Journal of Computing and Electronic Information Management
ISSN: 2413-1660 | Vol. 11, No. 3, 2023

The Design of P2P chat software in LAN

Yunzheng Ding *

School of Information Engineering, Jingdezhen Ceramic university, Jingdezhen 333403, China
* **Corresponding author Email:** jciddyz@163.com

Abstract: Nowadays, many units have set up their own LAN, some units do not allow staff to chat on the net while working, so only 80 ports are opened on the exit gateway of the LAN, which brings inconvenience to some staff who need to exchange information or transfer files when working. They need a chat software that can exchange information in the LAN. And now the college student dormitory computer popularity rate is very high, many of them only connected to the school's LAN, and did not pay for the Internet, so they also need a LAN chat software to exchange information.

Keywords: Java; P2P; Network programming; Multithreading.

1. Functional requirements

This program is a P2P(Peer-to-Peer, end to end) chat tool running in the LAN. By running the software, users in the same LAN can communicate with each other conveniently. At the same time, this network is dynamic, in other words, users can join and leave this network at any time.

The main functions are: Get the IP of the endpoint (host) in the P2P network, and update it dynamically; Can send text to any endpoint with a known IP address; Can receive text messages from other endpoints; It is possible to obtain information that certain endpoints have exited the network.

2. Design and implementing code

2.1. InfoQueue.java

This class performs the basic functions of a queue. The inheritance of packages, classes, and methods used in this class file is shown in Figure 1.

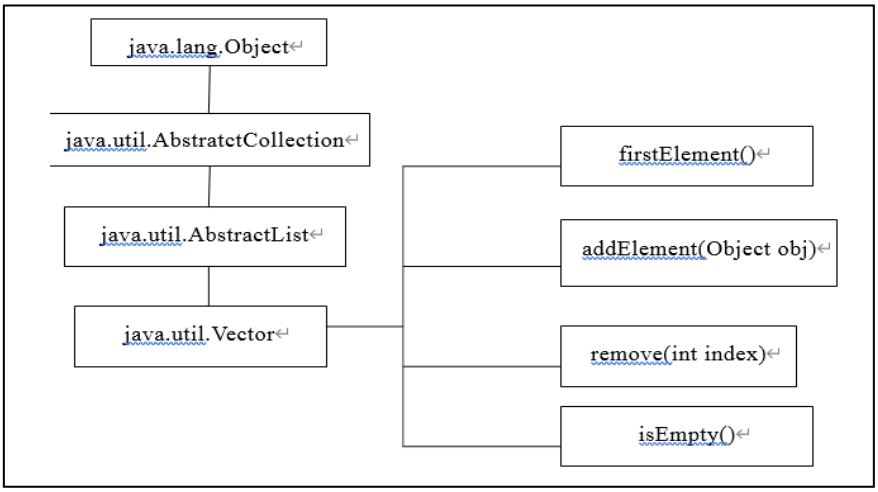

Fig.1 Inheritance diagram of the classes in the InfoQueue file

```
import java.util.Vector;
/** class InfoQueue */
public class InfoQueue{
// Define a Vector with an initial size of 10 and growth of
10

private Vector msgQueue=new Vector(10,10);
// Return the head of the line element
public synchronized String getElement(){
return msgQueue.firstElement() .toString() .trim() ;
}
// Add elements to the queue
public synchronized void addElement(String content){
msgQueue.addElement(content);
}
// Remove the queue element
public synchronized void delElement(){
msgQueue.remove(0);
}
// Test if the queue is empty
public synchronized boolean isEmpty(){
if(msgQueue.isEmpty() )
return true;
else
return false;
}
}
```

2.2. InfoResponse.java

This class completes the response to the user's request. It mainly includes: the user obtains the IP address, sends the information, exits the system. The main code as follows

```
// Stop the response
public void stopWork(){
```

```
isWork=false;
}
/** Update IP address list */
private synchronized void addIpList(String ip){
int i;
// If IP list is empty
if(p2pSys.modeIp .isEmpty() ){
// Add the IP address
p2pSys.modeIp .addElement(ip);
}
else {
// Otherwise, first check if this IP is duplicate with the existing IP
for(i=0; i&lt; p2pSys.modeIp .size(); i++ ){
if(p2pSys.modeIp .getElementAt(i).toString().trim() .equals(ip) )
// Exit if the IP already exists
break;
}

// Add the IP list if it doesn't exist
if(i&gt; =p2pSys.modeIp .size() ){
p2pSys.modeIp .addElement(ip);
}
}
}
/** Handle the IP address request */
private synchronized void getIp(){
try{
// Resolve the IP address
String ip=xmlParse .parse(msg,"ip").trim() ;
System.out.println(ip+"in infoResponse.java");
// Turn the text representation of an IP address into a real IP address.
InetAddress inetAddress=InetAddress.getByName(ip);
// Define send message
String sendString=inetAddress.toString() +" We're online!" ;
// Add the IP address to the IP address list
addIpList(ip);
// Create a DatagramSocket to send UDP packets
DatagramSocket ds=new DatagramSocket();
// Get the IP address of the local host
String sAddress=InetAddress.getLocalHost() .toString() ;
// Encapsulate the IP address
sAddress=sAddress.substring(sAddress.indexOf('/')+1 );
// Encapsulate the message you want to send
sendString=xmlFormat.xmlForm("p2p",sendString,sAddress ,"8888");
// Byte array to send the packet
byte[] buf = sendString.getBytes();
// Instantiate a datagram packet
DatagramPacket dp = new DatagramPacket(buf, buf.length,inetAddress, 8888);
// Send the datagram from the socket
ds.send(dp);
// Close datagram socket
ds.close();
}catch(Exception e){
// Display exception information
System.out.println("error in inforesponse.java "+e);
}
}
/** Handle incoming messages */
private synchronized void p2p(){
// Get the IP address
String ip= xmlParse.parse(msg,"ip");
ip=ip.substring(ip.indexOf('/')+1 ) ;
// Add the IP address to the IP list
addIpList(ip);
// Add sender's message to the message queue
p2pSys.modeMsg.addElement (" information from "+ip+ ":");
// Parse the message
String content=xmlParse.parse(msg,"content");
// Add the sender's message to the message queue
p2pSys.modeMsg .addElement(content);
}
/** Handle user exit from the system */
private synchronized void exitSys(){
// Resolve the IP address
String ip=xmlParse.parse(msg,"ip");
ip=ip.substring(ip.indexOf('/')+1 );
// Parse the message
String content=xmlParse.parse(msg,"content") ;
p2pSys.modeMsg.addElement (" Tip: "+ip+" "+content);
// In the IP list, remove the user's IP address if they are offline
int i;
for(i=0; i&lt; p2pSys.modeIp .size(); i++ ){
if(p2pSys.modeIp .getElementAt(i).toString().trim() .equals(ip) )
p2pSys.modeIp .remove(i);
}
if(p2pSys.ipTextField .getText() .toString() .trim() .equals(ip) )
// Empty the destination IP address text field if the current chat user is the one who just logged off
p2pSys.ipTextField .setText("");
}
/** Thread execution method */
public synchronized void run(){
while(isWork){
try{
sleep(100);
}catch(Exception e){
System.out.println("error"+e);
}
// Process the message queue
if(! p2pSys.infoQueue.isEmpty ()){// If there are messages left in the message queue
// Get the information
msg=p2pSys.infoQueue .getElement() .trim() ;
// Delete the message from the message queue
p2pSys.infoQueue .delElement() ;
// Parse the type of message sent

String parseResult=xmlParse .parse(msg,"type").trim() ;
// If the other party wants an IP address, respond to the request
if(parseResult.equals("getIp") ){
getIp();
continue;
}
// If the other party is already offline, process the request accordingly
if(parseResult.equals("exitSys") ){
exitSys();
```

```
continue;
}
// If it's a normal message, do something with it
if(parseResult.equals("p2p")){
p2p();
continue;
}
}
}
```

2.3. P2pSys.java

This is the main user interface, which completes the human-computer interaction interface, and the response to user requests. The inheritance of packages, classes, and methods used in this class file is shown in Figure 2.

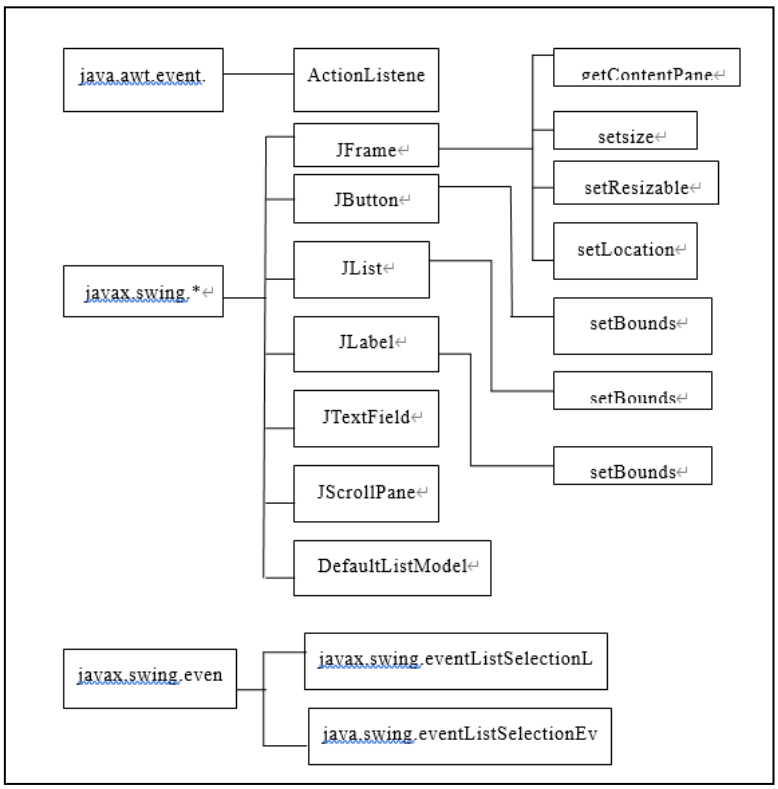

Fig.2 Inheritance relationships of classes and methods in P2pSys files

```
/** Import relevant packages */
import javax.swing.*;
import javax.swing.event.ListSelectionListener;
import javax.swing.event.ListSelectionEvent;
import java.awt.event.*;
import java.awt.*;
import java.util.Vector;
import java.net.InetAddress;
/** User interface */
public class P2pSys extends JFrame implements ActionListener{
// Store a list of IP addresses
Vector ipItem=new Vector(10,10);
// Store information
Vector messageItem=new Vector();
// Send message button
JButton sendButton;
// Get IP address button
JButton getIpButton;
// Message input field
JTextField messageTextField;
// Messages scroll panel
JScrollPane scrollMessage;
// Destination IP input field
JTextField ipTextField;
// Implement the list of IP addresses
DefaultListModel modeIp=new DefaultListModel();
// Implement information list
DefaultListModel modeMsg=new DefaultListModel();
// IP list
JList ipList;
// Information list
JList messageList;
// "Messages" TAB
JLabel messageLabel;
// "IP list" tag
JLabel ipLabel;
// "Destination IP address" tag
JLabel inputLabel;
// "Enter info" TAB
JLabel inputMessageLabel;
XmlFormat xmlFormat=new XmlFormat();
Sender sender=new Sender(P2pSys.this);
ReceiveMutiCast receiveMutiCast=new ReceiveMutiCast(P2pSys.this);
ReceiveP2p receiveP2p=new ReceiveP2p(P2pSys.this);
InfoQueue infoQueue=new InfoQueue();
InfoResponse infoResponse=new InfoResponse(P2pSys.this);
/** Constructor */
public P2pSys(){
// Main screen
setTitle(" LAN chat ");
// No preset layout
```

```
getContentPane().setLayout(null);
SetSize (500300);
setResizable(false);
SetLocation (200100);
// Display the "IP list" label
ipLabel=new JLabel("IP list ");
ipLabel.setBounds(370,10,80,20);
getContentPane().add(ipLabel);
// IP list
ipList=new JList(modeIp);
JScrollPane scrollIp=new JScrollPane(ipList);
scrollIp.setBounds(370,40,110,100);
getContentPane().add(scrollIp);
// When an IP address is selected, display it in the destination IP address
ListSelectionListener listSelectionListener = new ListSelectionListener() {
public void valueChanged(ListSelectionEvent e) {
ipTextField.setText(ipList.getSelectedValue() .toString() );
}
};
ipList.addListSelectionListener(listSelectionListener) ;
// "Chat history" TAB
messageLabel=new JLabel(" chat history: ");
messageLabel.setBounds(10,10,150,20);
getContentPane().add(messageLabel);
messageList= new JList(modeMsg);
scrollMessage = new JScrollPane(messageList);
scrollMessage.setBounds(10,40,350,140);
getContentPane().add(scrollMessage) ;
// The "About" button
aboutButton=new JButton(" about ");
aboutButton.setBounds(210,10,150,20);
getContentPane().add(aboutButton);
// Add an event listener for the button
aboutButton.addActionListener(this);
// "Get IP" button
getIpButton=new JButton(" Get the IP");
getIpButton.setBounds(385,150,80,30);
getContentPane().add(getIpButton);
// Add an event listener for the button
getIpButton.addActionListener(this);
// "Destination IP" tag
inputLabel=new JLabel(" destination IP: ");
inputLabel.setBounds(140,190,170,30);
getContentPane().add(inputLabel);
// "Enter message" label
inputMessageLabel=new JLabel(" Input message: ");
InputMessageLabel. SetBounds (10, 220, 80, 30);
getContentPane().add(inputMessageLabel);
// "Destination IP" text field
ipTextField=new JTextField("");
ipTextField.setBounds(230,190,130,30);
ipTextField.setForeground(Color.red );
getContentPane().add(ipTextField);
// "Enter message" text field
messageTextField=new JTextField("");
MessageTextField. SetBounds (80, 220, 280, 30);
getContentPane().add(messageTextField);
// "Send" button
sendButton=new JButton(" send ");
sendButton.setBounds(385,220,80,30);
getContentPane().add(sendButton);
// Add an event listener for the button
sendButton.addActionListener(this);
// Start the "Receive broadcast message" thread
receiveMutiCast.start() ;
// Start the "Receive P2P info" thread
receiveP2p.start() ;
// Start the "Message response" thread
infoResponse.start() ;
// Add a window event listener
addWindowListener(new WindowAdapter(){
// Close window event
public void windowClosing(WindowEvent e){
// When an endpoint exits, broadcast the endpoint down message
Stringstring=xmlFormat.xmlForm("exitSys", left ",getLocalIp(),"").trim();
sender.sendMutiCast(string);
// Close all running threads
receiveMutiCast.stopWork() ;
receiveP2p.stopWork() ;
infoResponse.stopWork() ;
// Exit the system
System.exit(0);
}
});
}
/** Handle button events */

public void actionPerformed(ActionEvent e){
Object object=e.getSource() ;
// If you click the Send button
if(object.equals(sendButton) ){
// if at least one of the destination IP and input message text fields has content
if(! (ipTextField.getText(). Equals ("") & & messageTextField.getText(). equals (""")) ){
// Add information to the message queue
modeMsg.addElement(" Message to "+ipTextField.getText()+ ");
modeMsg.addElement(messageTextField.getText() );
// Get the destination IP address
String sAddress=ipTextField.getText() .trim() ;
// Get the local IP address
String ip=getLocalIp() ;
// Get the message to send
String str=messageTextField.getText() .toString().trim() ;
// Encapsulate packet information
String string=xmlFormat.xmlForm("p2p",str,ip,"8888");
// Send message to destination IP chat endpoint
sender.sendP2p (string,sAddress );
}
}
// If click the Get IP button
if(object.equals(getIpButton) ){
// Clear the existing IP list
modeIp.clear() ;
// Set destination IP box to empty
ipTextField.setText("");
// Broadcast the message to get the IP list
String ip=getLocalIp() ;
String string=xmlFormat.xmlForm("getIp","",ip,"") .trim() ;
sender.sendMutiCast(string) ;
}
}
```

```
// If click Get About button
if(object.equals(aboutButton) ){
String title=" About LAN chat ";
String message=" made by Lu Jin ★Version_1.0";
int type=JOptionPane.PLAIN_MESSAGE;
JOptionPane.showMessageDialog(this,message,title,type,
new ImageIcon("about.jpg"));
}
/** Get native IP and wrap it in XML format */
private String  getLocalIp(){
String ip="";
try{
InetAddress inetAddress=InetAddress.getLocalHost() ;
ip=inetAddress.toString() ;
ip=ip.substring(ip.indexOf('/')+1 ) ;
}catch(Exception e){
System.out.println("in getLocalIp() "+e);
}
return ip;
}
/** Entry point to the program */
public static void main(String[] args){
JFrame p2pSys=new P2pSys();
p2pSys.show() ;
}
}
```

2.4. ReceiveMutiCast.java

This file implements the function of receiving group broadcast. The specified IP address for the broadcast group is: 234.5.6.7; The Socket port used is: 6789.

```
import java.net*;
/** Receive group broadcast */
public class ReceiveMutiCast extends Thread{
private P2pSys p2pSys;
private boolean isWork=true;
/** Constructor */
public ReceiveMutiCast(P2pSys p2pSys){
this.p2pSys=p2pSys;
}
/** Stop running */
public void stopWork(){
isWork=false;
}
/** Start thread */
public synchronized void run(){
try{
// Specify broadcast group socket port: 6789
MulticastSocket multicastSocket = new
MulticastSocket(6789);
// Specify broadcast group IP address: 234.5.6.7
InetAddress inetAddress = inetAddress.getByName
("234.5.6.7");
// Join a broadcast group
multicastSocket.joinGroup(inetAddress);
// Receive the group broadcast when the thread is running
normally
while(isWork){
// Specify a packet size of 1000 bytes to receive the
broadcast
byte [] b = new byte [1000];
DatagramPacket datagramPacket = new DatagramPacket(b,
b.length);
// Receive the group broadcast
multicastSocket.receive(datagramPacket);
// Get the broadcast message content
String mutiCastMsg=new String(b);
mutiCastMsg=mutiCastMsg.substring(0,mutiCastMsg.last
IndexOf('&gt; ') +1);
// Add the group broadcast message to the message queue
mutiCastMsg=mutiCastMsg.trim() ;
p2pSys.infoQueue .addElement (mutiCastMsg);
}
}catch(Exception e){
System.out.println("error in receivemuticast  "+e) ;
}
}
```

2.5. ReceiveP2p.java

This class implements the function of receiving the other party's message.

```
/** Import UDP package resources */
import java.net.DatagramSocket;
import java.net.DatagramPacket;
/** Receive P2P messages */
public class ReceiveP2p extends Thread{
private P2pSys  p2pSys;
// Specify an array size of 1000 bytes for receiving
messages
byte[] buf = new byte[1000];
// Define UDP socket
DatagramSocket ds;
// Define UDP datagram packets
DatagramPacket dp;
XmlFormat xmlFormat=new XmlFormat();
boolean isWork=true;
public void stopWork(){
isWork=false;
}
public ReceiveP2p(P2pSys p2pSys){
this.p2pSys=p2pSys;
}
/** Start thread */
public void run(){
try{
// Specify UDP socket port 8888
ds=new DatagramSocket(8888);
// Specify the datagram packet to receive information
dp=new DatagramPacket(buf,buf.length );
// when the thread is running normally
while(isWork){
// Receive information from an IP address
ds.receive(ip);
// Add received messages to the message queue
p2pSys.infoQueue .addElement (new String(ip.getData()) );
}
// Close the socket
ds.close() ;
}catch(Exception e){
System.out.println("error in receivep2p.java"+e) ;
}
}
}
```

2.6. Sender.java

This file implements the function of sending messages.

In the method of sending P2P messages, its UDP port is the port 8888 which is responsible for accepting datagram

packets by the peer. In the method of sending group broadcast messages, the specified UDP port is the port 6789 in the broadcast group that is responsible for receiving group broadcast, and the IP address of the broadcast group is the class D IP address: 234.5.6.7.

```
/** Import network package resources */
import java.net.*;
/** Send message */
public class Sender{
private P2pSys p2pSys;
/** Constructor */
public Sender(P2pSys p2pSys){
this.p2pSys=p2pSys;
}
/** Methods to send P2P messages */
public void sendP2p(String sendString,String destinationIp){
try{
// Convert text IP address to real IP address
InetAddress ia=InetAddress.getByName(destinationIp);
// Initialize the UDP socket
DatagramSocket ds=new DatagramSocket();
// Get the message to send
byte[] buf = sendString.getBytes() ;
// Instantiate UDP datagram packets
DatagramPacket dp = new DatagramPacket(buf, buf.length, ia, 8888);
// Send the datagram packet
ds.send(dp);
// Close UDP socket after sending
ds.close();
}catch(Exception e){

System.out.println("error in Sender.java"+e);
}
}
/** Method to send a group broadcast message */
public void sendMutiCast(String s){
try{// catch the exception
byte[] b=s.getBytes() ;
// Specify broadcast group IP address: 234.5.6.7
InetAddress inetAddress= inetAddress.getByName ("234.5.6.7");
// Instantiate datagram packet with port: 6789
DatagramPacket datagramPacket =new DatagramPacket(b, b.length, inetAddress, 6789);
// Create broadcast group socket
MulticastSocket multicastSocket = new MulticastSocket();
// Send a group broadcast message
multicastSocket.send(datagramPacket);
// Close the group broadcast socket after sending
multicastSocket.close() ;
}catch(Exception e){// Exception handling
p2pSys.modeMsg .addElement("Note: getting ip is failed Please click getIp button again");
System.out.println("error in Sender.java's sendMutiCast"+e);
}
}
```

2.7. XmlFormat.java

This class implements the encapsulation of XML information. The information is encapsulated in XML format, and after the XML is parsed, we can quickly find the information in a specific part of the information. Among them, the encapsulated information consists of four parts: message type, message content, IP address, Socket port.

There are three types of messages: "getIP", "P2P", "exitSys". When the message arrives, the receiver can process it according to the type of message.

```
/** Information encapsulation */
public class XmlFormat{
// Message to send
private String sendString;
/** No argument constructor */
public XmlFormat(){
}
/** Information wrapper method with 4 arguments */
public String xmlForm(String type,String content,String ip,String port){
sendString="&lt; ? xml version=\"1.0\" encoding=\"UTF-8\"? &gt;"
+"&lt; msg&gt;"
// Message type
+"&lt; type&gt;" +type+"&lt; /type&gt;"
// Message content
+"&lt; content&gt;" +content+"&lt; /content&gt;"
// IP address
+"&lt; ip&gt;" +ip +"&lt; /ip&gt;"
// Port address
+"&lt; port&gt;" +port +"&lt; /port&gt;"
+"&lt; /msg&gt;" ;
// This class returns the wrapped information
return sendString;
}
}
```

2.8. XmlParse.java

This class implements the parsing of XML information. XML parsing and XML encapsulation are two completely opposite processes. It is worth mentioning that <table> and </table> are used in message parsing as a flag to retrieve a part of the message. By assigning "type", "content", "ip", and "port" to the table, we get the contents of one of the four parts of the message.

```
/** Parsing information */
public class XmlParse{
/** Message parsing methods */
public String parse(String xml,String tab){
String s="";
int startIndex=-1;
int endIndex=-1;
startIndex=xml.indexOf( "&lt;" +tab+"&gt;" );
endIndex=xml.indexOf("&lt; /"+tab+"&gt;" );;
if(startIndex! = 1) {
// The starting point for the desired content
int contentStart=xml.indexOf('&gt; ',startIndex)+1;
// Get what you want
s= xml.substring(contentStart,endIndex);
}
// This class returns what you want
return s;
}
}
```

3. Program debugging

In the process of debugging the program, the program is run twice on a single machine to simulate P2P communication

between two hosts, and the problem shown in Figure 3 occurs.Because this is a single machine test, only the IP address of the local machine is obtained. Figure 3 is the case of two run results A and B simulating communication. Whether A or B sends A message, only A can receive it.

This happens because of the result of the first run, A, which binds the native IP address and UDP port (8888 is specified). When run again, result B throws the following exception:

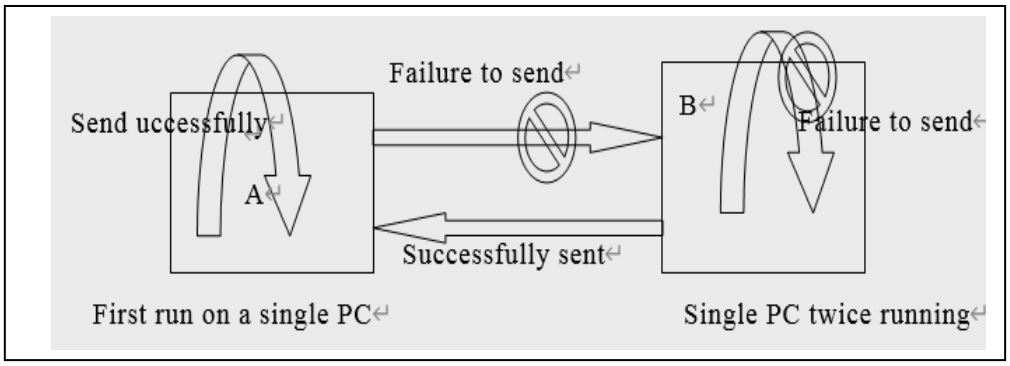

Fig.3 Simulate P2P on single PC

Java.net.bindexception. this exception is thrown if an error occurred while attempting to bind a socket to a local address and port. These errors usually occur when the port is in use or when the requested local address cannot be allocated.

Therefore, the only way to get the correct results as expected is to run the test on multiple (at least two) PCs.

References

[1] Yang Hongbo ,Wang Zhishun, “J2SE evolution history”, Programmers, 2005(07), P50-52.

[2] LIU Xiao-zheng, “Analysis of Java GUI programming tool set”, SCIENCE & TECHNOLOGY INFORMATION, 2012(35), P596-597.

[3] Wang chun-li, “Network programming using Java”, CHINA SCIENCE AND TECHNOLOGY INFORMATION, 2006(04), P186-187.

Li rui-ge,” Computer software development of Java programming language and applications”, Programmers, 2022(06), P15-16.