# COSC264 Assignment Report

Yuan Cui(63483319) / Weiling Xie(83743927)

1. The protocol between **sender** and **receiver** as described above has (at least) one

weakness: it has a **deadlock**. Please explain the notion of a deadlock in the

context of networking protocols and describe the particular deadlock situation in

our case. A guiding question is: what can go wrong and when in case certain

packets are lost?

**Deadlock happens when the last acknowledge packet from receiver to sender is dropped or modified datalen by the channel, sender will keep sending this last data packet without response from receiver forever.**

2. What is the **magicno** field good for?

**Magicno may refer to the protocol we implement in this program. Each data packet transfer through this protocol should use the same magicno. For example, in the packet format, there is a protocol field which include a hexadecimal number corresponding to a special protocol, such as 0x06 represents TCP protocol while 0x11 represents UDP.**

3. How have you solved the issue with the **bit errors**? Please explain what you have

added to the packet and to the **sender** and **receiver** modules.

**Checksum was added into the packet as an integer in the program. In the sender module, before sending the packet, myCheckSum function will calculate the checksum value and add it to packet. In receive model, myCheckSum function will calculate another checksum value through the received packet and according to the comparing result by two checksum values to decide whether to keep the packet or drop it.**

4. Please explain what the **select()** function is doing and why it is useful for the

channel (and in another way for the **sender**).

**Select() enable program to wait for multiple sockets. If there is no available input, the process will be blocked which can save CPU resource.**

**In channel part of this program, for example, if there are two packets coming together. The last coming one did not need to wait for first one finishing its read() function and logic statement. Theoretically, the two packets can be read at the same time, which a little bit like using threads.**

**In sender part, because the read socket is non-blocking and write socket is blocking, selector() can implement the delay sending by select(TIMEOUT), which means if there is an incoming packet, after logic judgement, write socket can send another packet to channel without time delay. However, if no incoming packet, the write will be blocked and retransmit after TIMEOUT ms.**

5. Please explain how you have checked whether or not the file was transferred correctly (i.e. the receivers copy is identical to the transmitters copy).

**We can use md5sum tool to generate md5 values for both original file and the file we received, and compare the md5 value, if they are the same, it means the file was transferred correctly, or it was transferred incorrectly. For checking txt file, we can use diff as well.**

6. We consider different packet loss probabilities of P ∈ {0:0; 0:01; 0:05; 0:1; 0:2; 0:3}

and a source file of length M = 512 \* 100 = 51,200 bytes (you need to create such

a file). For each value of P make ten repetitions of the file transfer and for each

repetition record how many packets the **sender** has sent in total. Draw a graph

that shows the different values of P on the x-axis and for each such value the

*average* number of total packets (the average being taken over the ten repetitions)

on the y-axis. Explain the results.

Note: To produce graphs, the tool gnuplot can be useful under Linux. Its main

advantage is that it allows for script-based (i.e. non-interactive) creation of graphs,

but admittedly its command syntax needs some getting used to. However, you are

free to use any tool you like (including Excel, Matlab, etc.) for producing graphs.

**Under all circumstances you need to make sure that axes and curves**

**are properly labeled**. You will lose marks otherwise.



**As can be seen from the bar chart above, with the increase of the packet loss rate, the average number of total packets rises as well. Because if we ignore the datalen modification by channel and the expected average successfully transfer rate pr(S) can be calculated as (1-P)2, which means it won’t be a linear increasing tendency.**

7. Assume the following:

* The probability to lose an individual packet (either a dataPacket or an

acknowledgementPacket) is P,

* Packet loss events are statistically independent of each other.
* The size of the file to be transmitted requires N packets.

Please derive and justify an expression for the average total number of packets that need to be sent (including retransmissions) to transmit the entire file. Compare this to the (average) total number of packets you have observed in your experiments.

**Let the mean total number of packets be E(T).**

**N = E(T) \* Pr(one packet successfully transfer)**

**E(T) = N / Pr(one packet successfully transfer)**

**Because packet loss events are independent of each other and there are four opportunities of packets loss which are P in packet transfer from sender to receiver, 10% in receiver dropping the packet as modified datalen, P in acknowledge packet transfer from receiver to sender and 10% in sender dropping the packet as modified datalen.**

**The Pr(one packet successfully transfer) is (1-P) \* (1-P) \* (1-10%) \* (1-10%) = 0.81 \* (1-P)2 , we can use the expression E(T) = N/(0.81\*(1-P)2) to predict the average number of total packets.**

**If we take P ∈ {0.0; 0.01; 0.05; 0.1; 0.2; 0.3}, and N = 100 to the formula above,**

|  |  |  |
| --- | --- | --- |
| Packet Loss Rate | mean | E(T) |
| 0 | 123.7 | 123.5 |
| 0.01 | 126.6 | 126.0 |
| 0.05 | 136.6 | 136.8 |
| 0.1 | 154.1 | 152.4 |
| 0.2 | 190.3 | 192.9 |
| 0.3 | 252.2 | 252.0 |

**Comparing with ten times average number of total packets (mean) and prediction E(T), we can find they are quite closed, which proved the correctness of this program in the other side.**

**Source Code with Java:**

**Sender.class**

package TCP\_Protocol;

import java.io.File;

import java.io.IOException;

import java.io.RandomAccessFile;

import java.net.InetSocketAddress;

import java.nio.ByteBuffer;

import java.nio.channels.SelectionKey;

import java.nio.channels.Selector;

import java.nio.channels.ServerSocketChannel;

import java.nio.channels.SocketChannel;

import java.util.Iterator;

import java.util.Scanner;

import java.util.Set;

/\*\*

\*

\* @author Yuan & Xie

\* Sender module

\*/

public class Sender {

public static final int DATASIZE = 512;

public static final int TIMEOUT = 200;

static Scanner scan = new Scanner(System.in);

int next = 0;

public Sender(File file, int sinPort, int soutPort, int csinPort) throws IOException{

ServerSocketChannel sscFC = null;

SocketChannel scFC = null;

SocketChannel scTC = null;

Packet[] packets = null;

Packet packet = null;

Packet acknowledgementPacket = null;

Selector selector = null;

selector = Selector.open();

int packetNum = 0;

int totalPacketSendNo = 0;

boolean exitFlag = true;

packets = splitFile(file);

sscFC = ServerSocketChannel.open();

sscFC.bind(new InetSocketAddress("localhost", sinPort));

System.out.println("SCFC is listening on Port " + sinPort);

scTC = SocketChannel.open();

scTC.bind(new InetSocketAddress(soutPort));

scTC.connect(new InetSocketAddress("localhost", csinPort));

System.out.println("SCTC is connecting to channel");

if(scTC.finishConnect()){

System.out.println("SCTC has connected to channel");

}

scFC = sscFC.accept();

System.out.println("SCFC has accepted the request from channel");

scFC.configureBlocking(false);

scFC.register(selector, SelectionKey.OP\_READ);

while(exitFlag){

selector.select(TIMEOUT);

Set<SelectionKey> selectionKeys = selector.selectedKeys();

Iterator<SelectionKey> it = selectionKeys.iterator();

while(it.hasNext()){

SelectionKey sk = it.next();

it.remove();

if(sk.isReadable()){

ByteBuffer bb = ByteBuffer.allocate(1000);

int result = scFC.read(bb);

if(result != 0){

acknowledgementPacket = UtilFunction.bToP(bb.array());

if(checkResponse(packet, acknowledgementPacket)){

packetNum++;

if(packetNum == packets.length){

exitFlag = false;

break;

}

next = 1-next;

packets[packetNum].setSeqno(next);

}

}

}

}

if(exitFlag){

packet = packets[packetNum];

packet.setChecksum(UtilFunction.myCheckSum(packet));

int i = UtilFunction.sendPacket(scTC, packet);

totalPacketSendNo++;

System.out.println(totalPacketSendNo);

}

}

if(selector!=null)

selector.close();

if(sscFC!=null)

sscFC.close();

if(scFC!=null)

scFC.close();

if(scTC!=null)

scTC.close();

System.out.println("total sending packet number is " + totalPacketSendNo);

}

public Packet[] splitFile(File file) throws IOException {

long fileLength = file.length();

int count = 0;

int numOfPackets = (int) Math.ceil((double) fileLength / DATASIZE);

Packet[] packets = new Packet[numOfPackets + 1];

RandomAccessFile raf = new RandomAccessFile(file, "r");

int result = 0;

while (true) {

byte[] b = new byte[DATASIZE];

if((count+1) \* DATASIZE > fileLength && count\*DATASIZE != fileLength){

b = new byte[(int) (fileLength%DATASIZE)];

}

result = raf.read(b);

if(result == -1){

break;

}

packets[count] = new Packet(Packet.PTYPE\_DATA, 0, DATASIZE);

packets[count].setDataLen(result);

packets[count].setData(b);

packets[count].setNo(count);

System.out.println(packets[count]);

count++;

}

packets[numOfPackets] = new Packet(Packet.PTYPE\_DATA, 0, 0);

packets[numOfPackets].setNo(numOfPackets);

if(raf!=null)

raf.close();

return packets;

}

public boolean checkResponse(Packet packet, Packet acknowledgementPacket) {

if(acknowledgementPacket.getMagicno() == 0x497E && acknowledgementPacket.getType()== Packet.PTYPE\_ACK && acknowledgementPacket.getDataLen() == 0 && acknowledgementPacket.getSeqno() == next){

return true;

}

return false;

}

}

**Channel.class**

package TCP\_Protocol;

import java.io.IOException;

import java.net.InetSocketAddress;

import java.nio.ByteBuffer;

import java.nio.channels.SelectionKey;

import java.nio.channels.Selector;

import java.nio.channels.ServerSocketChannel;

import java.nio.channels.SocketChannel;

import java.util.Iterator;

import java.util.Random;

import java.util.Set;

/\*\*

\*

\* @author Yuan & Xie

\* Channel module

\*

\*/

public class Channel {

SocketChannel scTS = null;

SocketChannel scFS = null;

SocketChannel scTR = null;

SocketChannel scFR = null;

ServerSocketChannel sscFR = null;

ServerSocketChannel sscFS = null;

Packet packet = null;

Packet ackPacket = null;

Packet unknownPacket = null;

Selector selector = null;

ByteBuffer bfTS = null;

ByteBuffer bfTR = null;

Random rand = new Random();

public Channel(int csinPort, int csoutPort, int rinPort, int croutPort, int crinPort, int sinPort, double packetLossRate)

throws IOException {

selector = Selector.open();

// Channel from Sender

sscFS = ServerSocketChannel.open();

sscFS.bind(new InetSocketAddress("localhost", csinPort));

System.out.println("SCFS is listening on Port " + csinPort);

scFS = sscFS.accept();

System.out.println("SSFS has accepted the request from sender");

scFS.configureBlocking(false);

scFS.register(selector, SelectionKey.OP\_READ);

// Channel to Receiver

scTR = SocketChannel.open();

scTR.bind(new InetSocketAddress(croutPort));

scTR.connect(new InetSocketAddress("localhost", rinPort));

System.out.println("SCTR is connecting to Receiver");

if (scTR.finishConnect()) {

System.out.println("SCTR has connected to receriver");

}

// Channel from Receiver

sscFR = ServerSocketChannel.open();

sscFR.bind(new InetSocketAddress("localhost", crinPort));

System.out.println("sscFR is listening on Port " + crinPort);

scFR = sscFR.accept();

System.out.println("SCFR has accepted the request from receiver");

scFR.configureBlocking(false);

scFR.register(selector, SelectionKey.OP\_READ);

// Channel to Sender

scTS = SocketChannel.open();

scTS.bind(new InetSocketAddress(csoutPort));

scTS.connect(new InetSocketAddress("localhost", sinPort));

System.out.println("SCTS is connectiog to sender");

if (scTS.finishConnect()) {

System.out.println("SCTS has connected to sender");

}

while (!scFS.isConnected()||!scFR.isConnected()||scTS.isConnected()||scTR.isConnected()) {

selector.select();

Set<SelectionKey> selectionKeys = selector.selectedKeys();

Iterator<SelectionKey> it = selectionKeys.iterator();

while (it.hasNext()) {

SelectionKey sk = it.next();

it.remove();

if (sk.isReadable()) {

ByteBuffer bb = ByteBuffer.allocate(1000);

int result = ((SocketChannel) sk.channel()).read(bb);

if (result != 0) {

unknownPacket = UtilFunction.bToP(bb.array());

//data packet

if (unknownPacket.getType() == Packet.PTYPE\_DATA) {

packet = unknownPacket;

if(packet.getMagicno() != 0X497E || rand.nextDouble() < packetLossRate){

break;

}else if (rand.nextDouble() < 0.1){

packet.setDataLen(packet.getDataLen() + rand.nextInt(10) + 1);

}

int size1 = UtilFunction.sendPacket(scTR, packet);

System.out.println("packet from channel to receiver, size is "+ size1);

}else if (unknownPacket.getType() == Packet.PTYPE\_ACK){

ackPacket = unknownPacket;

if(ackPacket.getMagicno() != 0X497E || rand.nextDouble() < packetLossRate){

break;

}else if (rand.nextDouble() < 0.1){

ackPacket.setDataLen(ackPacket.getDataLen() + rand.nextInt(10) + 1);

}

int size2 = UtilFunction.sendPacket(scTS, ackPacket);

System.out.println("packet from channel to sender, size is "+ size2);

}

}

}

}

}

if(sscFR!=null)

sscFR.close();

if(sscFS!=null)

sscFS.close();

if(scFR!=null)

scFR.close();

if(scTR!=null)

scTR.close();

if(scFS!=null)

scFS.close();

if(scTS!=null)

scTS.close();

if(selector!=null)

selector.close();

}

}

**Receiver.class**

package TCP\_Protocol;

import java.io.IOException;

import java.net.InetSocketAddress;

import java.nio.channels.ServerSocketChannel;

import java.nio.channels.SocketChannel;

import java.util.ArrayList;

/\*\*

\*

\* @author Yuan & Xie

\* Receiver Module

\*

\*/

public class Receiver {

ServerSocketChannel sscFC = null;

SocketChannel scFC = null;

SocketChannel scTC = null;

Packet packet = null;

Packet ackPacket = null;

ArrayList<Packet> packets = new ArrayList<Packet>();

int expected = 0;

int num = -1;

public Receiver(int rinPort, int routPort, int crinPort, String fileName) throws IOException{

//Receiver from Channel

sscFC = ServerSocketChannel.open();

sscFC.bind(new InetSocketAddress("localhost", rinPort));

System.out.println("SCFC is listening on port "+ rinPort);

scFC = sscFC.accept();

System.out.println("SCFC has accepted the request from channel");

//Receiver to Channel;

scTC = SocketChannel.open();

scTC.bind(new InetSocketAddress(routPort));

scTC.connect(new InetSocketAddress("localhost", crinPort));

System.out.println("SCTC is connecting to channel");

if(scTC.finishConnect()){

System.out.println("SCTC has connected to channel");

}

while(true){

packet = UtilFunction.getPacket(scFC);

if(packet.getDataLen() == 0){

ackPacket = UtilFunction.pToAck(packet);

UtilFunction.sendPacket(scTC, ackPacket);

break;

}else if(packet.getMagicno()!=0x497E || packet.getType() != Packet.PTYPE\_DATA || packet.getChecksum() != UtilFunction.myCheckSum(packet)){

continue;

}else if(packet.getSeqno() != expected){

ackPacket = UtilFunction.pToAck(packet);

UtilFunction.sendPacket(scTC, ackPacket);

}else{

System.out.println(num+","+packet.getNo());

if(num != packet.getNo()){

System.out.println(packet.getData().length);

packets.add(packet);

num = packet.getNo();

ackPacket = UtilFunction.pToAck(packet);

UtilFunction.sendPacket(scTC, ackPacket);

expected = 1- expected;

}

}

}

if(sscFC!=null)

sscFC.close();

if(scFC!=null)

scFC.close();

if(scTC!=null)

scTC.close();

UtilFunction.psToFile(packets, fileName);

}

}

**Packet.class**

package TCP\_Protocol;

import java.io.Serializable;

/\*\*

\*

\* @author Yuan & Xie

\*

\*/

public class Packet implements Serializable{

private static final long serialVersionUID = -7996935087676307689L;

private int magicno;

private int type;

private int seqno;

private int dataLen;

private byte[] data;

private int no = 0;

private int checksum = 0;

public static final int PTYPE\_DATA = 0;

public static final int PTYPE\_ACK = 1;

public int getMagicno() {

return magicno;

}

public void setMagicno(int magicno) {

this.magicno = magicno;

}

public int getType() {

return type;

}

public void setType(int type) {

this.type = type;

}

public int getSeqno() {

return seqno;

}

public void setSeqno(int seqno) {

this.seqno = seqno;

}

public int getDataLen() {

return dataLen;

}

public void setDataLen(int dataLen) {

this.dataLen = dataLen;

}

public byte[] getData() {

return data;

}

public void setData(byte[] data) {

this.data = data;

}

public int getNo() {

return no;

}

public void setNo(int no) {

this.no = no;

}

public int getChecksum() {

return checksum;

}

public void setChecksum(int checksum) {

this.checksum = checksum;

}

public Packet(int type, int seqno, int dataLen) {

setMagicno(0x497E);

this.type = type;

this.seqno = seqno;

//if data length < 0 or data length >512, drop packet

if(dataLen>=0 && dataLen<=512){

this.dataLen = dataLen;

}else{

System.out.println("data length must between 0 and 512");

}

data = new byte[dataLen];

}

@Override

public String toString() {

return "Packet [magicno=" + magicno + ", type=" + type + ", seqno=" + seqno + ", dataLen=" + dataLen + ", no="

+ no + ", checksum=" + checksum + "]";

}

}

**UtilFunction.class**

package TCP\_Protocol;

import java.io.ByteArrayInputStream;

import java.io.ByteArrayOutputStream;

import java.io.File;

import java.io.IOException;

import java.io.ObjectInputStream;

import java.io.ObjectOutputStream;

import java.io.RandomAccessFile;

import java.nio.ByteBuffer;

import java.nio.channels.SocketChannel;

import java.util.ArrayList;

/\*\*

\*

\* @author Yuan & Xie

\*

\*/

public class UtilFunction {

public static byte[] pToB (Packet packet) throws IOException{

byte[] bytes = null;

ByteArrayOutputStream bo = new ByteArrayOutputStream();

ObjectOutputStream oo = new ObjectOutputStream(bo);

oo.writeObject(packet);

oo.flush();

bytes = bo.toByteArray();

bo.close();

oo.close();

return bytes;

}

public static Packet bToP (byte[] bytes) throws IOException{

ByteArrayInputStream bis = new ByteArrayInputStream (bytes);

ObjectInputStream ois = new ObjectInputStream (bis);

Packet packet = null;

try {

packet = (Packet) ois.readObject();

} catch (ClassNotFoundException e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

ois.close();

bis.close();

return packet;

}

public static int sendPacket(SocketChannel sc, Packet packet) throws IOException{

byte[] bytes = pToB(packet);

ByteBuffer bb = ByteBuffer.wrap(bytes);

int result = sc.write(bb);

return result;

}

public static Packet getPacket(SocketChannel sc) throws IOException{

ByteBuffer bb = ByteBuffer.allocate(1000);

int result = sc.read(bb);

// System.out.println("read "+result+" bytes");

Packet packet = bToP(bb.array());

return packet;

}

public static Packet pToAck(Packet packet){

Packet ackPacket = new Packet(Packet.PTYPE\_ACK, packet.getSeqno(), 0);

return ackPacket;

}

public static void psToFile(ArrayList<Packet> packets, String fileName) throws IOException{

byte[] bytes;

File file = new File(fileName);

RandomAccessFile raf = new RandomAccessFile(file, "rw");

for(int i = 0; i<packets.size(); i++){

bytes = packets.get(i).getData();

raf.write(bytes);

}

if(raf!=null)

raf.close();

}

public static int myCheckSum(Packet packet){

return packet.getData().length+packet.getDataLen()+packet.getMagicno()+packet.getNo()+packet.getSeqno()+packet.getType();

}

}

**SenderThread.class**

package TCP\_Protocol;

import java.io.File;

/\*\*

\*

\* @author Yuan & Xie

\*

\*/

public class SenderThread implements Runnable{

File fileRead;

int sinPort, soutPort, csinPort;

public SenderThread(File fileRead, int sinPort, int soutPort, int csinPort) {

super();

this.fileRead = fileRead;

this.sinPort = sinPort;

this.soutPort = soutPort;

this.csinPort = csinPort;

}

@Override

public void run() {

// TODO Auto-generated method stub

try {

Thread.sleep(500);

} catch (InterruptedException e1) {

// TODO Auto-generated catch block

e1.printStackTrace();

}

try{

new Sender(fileRead, sinPort, soutPort, csinPort);

}catch(Exception e){

e.printStackTrace();

}

}

}

**ChannelThread.class**

package TCP\_Protocol;

/\*\*

\*

\* @author Yuan & Xie

\*

\*/

public class ChannelThread implements Runnable{

int csinPort;

int csoutPort;

int rinPort;

int croutPort;

int crinPort;

int sinPort;

double packetLossRate;

public ChannelThread(int csinPort, int csoutPort, int rinPort, int croutPort, int crinPort, int sinPort,

double packetLossRate) {

this.csinPort = csinPort;

this.csoutPort = csoutPort;

this.rinPort = rinPort;

this.croutPort = croutPort;

this.crinPort = crinPort;

this.sinPort = sinPort;

this.packetLossRate = packetLossRate;

}

@Override

public void run() {

try{

new Channel(csinPort, csoutPort, rinPort, croutPort, crinPort, sinPort, packetLossRate);

}catch(Exception e){

System.out.println("channel lose connectiong and closing!");

}

}

}

**ReceiverThread.class**

package TCP\_Protocol;

/\*\*

\*

\* @author Yuan & Xie

\*

\*/

public class ReceiverThread implements Runnable{

int rinPort, routPort, crinPort;

String filePathWrite;

public ReceiverThread(int rinPort, int routPort, int crinPort, String filePathWrite) {

this.rinPort = rinPort;

this.routPort = routPort;

this.crinPort = crinPort;

this.filePathWrite = filePathWrite;

}

@Override

public void run() {

try{

new Receiver(rinPort, routPort, crinPort, filePathWrite);

}catch(Exception e){

e.printStackTrace();

}

}

}

**RunMe.class**

package TCP\_Protocol;

import java.io.File;

import java.util.Scanner;

/\*\*

\*

\* @author Yuan & Xie

\*

\*/

public class RunMe {

public static void main(String[] args) {

int sinPort = 0;

int soutPort = 0;

int csinPort = 0;

int crinPort = 0;

int csoutPort = 0;

int croutPort = 0;

int rinPort = 0;

int routPort = 0;

double packetLossRate = 0.0;

String filePathRead = null;

String filePathWrite = null;

File fileRead = null;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*

\* code for testing

\*/

// sinPort = 7001;

// soutPort = 7002;

// csinPort = 7003;

// crinPort = 7004;

// csoutPort = 7005;

// croutPort = 7006;

// rinPort = 7007;

// routPort = 7008;

// packetLossRate = 0.3;

// filePathRead = "null.txt";

// filePathWrite = "1.txt";

// fileRead = new File(filePathRead);

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

Scanner scan = new Scanner(System.in);

System.out.println("please enter the sender in from channel port No. from 1024 to 64000");

String sinPortStr = scan.next();

while(true){

try{

sinPort = Integer.parseInt(sinPortStr);

if(sinPort<1024 || sinPort>64000){

throw new RuntimeException("Port number must be an integer and its range between 1024 and 64000!");

}

break;

}catch(Exception e){

System.out.println("Wrong Port Number! please enter the sender in from channel port No. from 1024 to 64000");

sinPortStr = scan.next();

}

}

System.out.println("please enter the sender out to channel port No. from 1024 to 64000");

String soutPortStr = scan.next();

while(true){

try{

soutPort = Integer.parseInt(soutPortStr);

if(soutPort<1024 || soutPort>64000){

throw new RuntimeException("Port number must be an integer and its range between 1024 and 64000!");

}

break;

}catch(Exception e){

System.out.println("Wrong Port Number! please enter the sender out to channel port No. from 1024 to 64000");

soutPortStr = scan.next();

}

}

System.out.println("please enter the channel in from sender port No. from 1024 to 64000");

String csinPortStr = scan.next();

while(true){

try{

csinPort = Integer.parseInt(csinPortStr);

if(csinPort<1024 || csinPort>64000){

throw new RuntimeException("Port number must be an integer and its range between 1024 and 64000!");

}

break;

}catch(Exception e){

System.out.println("Wrong Port Number! please enter the channel in from sender port No. from 1024 to 64000");

csinPortStr = scan.next();

}

}

System.out.println("please enter the channel in from receiver port No. from 1024 to 64000");

String crinPortStr = scan.next();

while(true){

try{

crinPort = Integer.parseInt(crinPortStr);

if(crinPort<1024 || crinPort>64000){

throw new RuntimeException("Port number must be an integer and its range between 1024 and 64000!");

}

break;

}catch(Exception e){

System.out.println("Wrong Port Number! please enter the channel in from receiver port No. from 1024 to 64000");

crinPortStr = scan.next();

}

}

System.out.println("please enter the channel out to sender port No. from 1024 to 64000");

String csoutPortStr = scan.next();

while(true){

try{

csoutPort = Integer.parseInt(csoutPortStr);

if(csoutPort<1024 || csoutPort>64000){

throw new RuntimeException("Port number must be an integer and its range between 1024 and 64000!");

}

break;

}catch(Exception e){

System.out.println("Wrong Port Number! please enter the channel out to sender port No. from 1024 to 64000");

csoutPortStr = scan.next();

}

}

System.out.println("please enter the channel out to receiver port No. from 1024 to 64000");

String croutPortStr = scan.next();

while(true){

try{

croutPort = Integer.parseInt(croutPortStr);

if(croutPort<1024 || croutPort>64000){

throw new RuntimeException("Port number must be an integer and its range between 1024 and 64000!");

}

break;

}catch(Exception e){

System.out.println("Wrong Port Number! please enter the channel out to receiver port No. from 1024 to 64000");

croutPortStr = scan.next();

}

}

System.out.println("please enter the reciever in from channel port No. from 1024 to 64000");

String rinPortStr = scan.next();

while(true){

try{

rinPort = Integer.parseInt(rinPortStr);

if(rinPort<1024 || rinPort>64000){

throw new RuntimeException("Port number must be an integer and its range between 1024 and 64000!");

}

break;

}catch(Exception e){

System.out.println("Wrong Port Number! please enter the reciever in from channel port No. from 1024 to 64000");

rinPortStr = scan.next();

}

}

System.out.println("please enter the reciever out to channel port No. from 1024 to 64000");

String routPortStr = scan.next();

while(true){

try{

routPort = Integer.parseInt(routPortStr);

if(routPort<1024 || routPort>64000){

throw new RuntimeException("Port number must be an integer and its range between 1024 and 64000!");

}

break;

}catch(Exception e){

System.out.println("Wrong Port Number! please enter the reciever out to channel port No. from 1024 to 64000");

routPortStr = scan.next();

}

}

System.out.println("please enter packet loss rate, it will be a float which between 0 and 1 and exclude 1");

String pLRStr = scan.next();

while(true){

try{

packetLossRate = Double.parseDouble(pLRStr);

if(packetLossRate<0.0 ||packetLossRate>=1.0){

throw new RuntimeException("Packet loss rate must be a float and between 0 and 1");

}

break;

}catch(Exception e){

System.out.println("please enter packet loss rate, it will be a float which between 0 and 1 and exclude 1");

pLRStr = scan.next();

}

}

System.out.println("please enter file path for sending");

filePathRead = scan.next();

while(true){

try{

fileRead = new File(filePathRead);

if(!fileRead.exists()){

throw new RuntimeException("File not existed");

}

break;

}catch(Exception e){

System.out.println("File not existed, please enter correct file path for sending");

filePathRead = scan.next();

}

}

System.out.println("please enter file path for receiving");

filePathWrite = scan.next();

Thread receiverThread = new Thread(new ReceiverThread(rinPort, routPort, crinPort, filePathWrite));

Thread channelThread = new Thread(new ChannelThread(csinPort, csoutPort, rinPort, croutPort, crinPort, sinPort, packetLossRate));

Thread senderThread = new Thread(new SenderThread(fileRead, sinPort, soutPort, csinPort));

receiverThread.start();

channelThread.start();

senderThread.start();

}

}