**SOCKET PROGRAMMING IN JAVA:**

TRANSMITTER:

**package** Transmitter;

**import** java.net.DatagramPacket; **import** java.net.DatagramSocket; **import** java.net.InetAddress; **import** java.util.Random;

**public class** Transmitter **implements** Runnable {

**static int** *SENDER\_PORT\_NUM* = 4004; //to send the packet **static int** *RECEIVER\_PORT\_NUM* = 5004; //to receive the packet **static int** *msg\_size* = 30; //MPS

**static int** *pktreceived*=0; //initialising to 0 Thread t;

Transmitter()

{

t=**new** Thread(**this**); t.start();//start the timer and run

}

**public void** run()

{

**int** i=0;

**while**(i<4)

{

# try {

Thread.*sleep*(1000\*(**int**)Math.*pow*(2,i));//double the timer each time

} **catch** (InterruptedException e) {

e.printStackTrace();

}

**if**(*pktreceived*==1)

{

# break;

}

**else if**(*pktreceived*==0 && i==3)

{ System.*out*.println("Communication Error"); //when there is no

communication, give this error message

System.*exit*(0);

} i++;

}

}

**public static void** main(String[] args) **throws** Exception{ Random randGen = **new** Random();

**byte**[] sentMessage = **new byte**[500]; //Random number generation randGen.nextBytes(sentMessage);

**int** bytesSent = 0;

**int** receivedAdv = 30;

// Create the socket to communicate with receiver

DatagramSocket clientSocket = **new** DatagramSocket(*SENDER\_PORT\_NUM*);

**while**(bytesSent < 500) {

/\*creating Transmitter Message first len equal to 30 and adjusted based on value given by Receiver \*/ **byte** [] tx\_msg =**new byte**[receivedAdv];

**for**(**int** i =0; i<tx\_msg.length; i++){ tx\_msg[i] = 0;

}

// filling the data part

**for**(**int** i = bytesSent, j=8; i<bytesSent+receivedAdv-8 && i<500; i++, j++) tx\_msg[j] = sentMessage[i];

// msg type is modified only if its the last packet.

**if**(bytesSent > 500)

tx\_msg[1] = 1;

// filling the sequence number

tx\_msg[2] = (**byte**)((bytesSent & 0xFF00) >> 8); tx\_msg[3] = (**byte**)(bytesSent & 0xFF);

// filling the length

**int** lengthOfDataPart = receivedAdv-8;

tx\_msg[6] = (**byte**)((lengthOfDataPart & 0xFF00) >> 8); tx\_msg[7] = (**byte**)(lengthOfDataPart & 0xFF);

// filling the checksum

**byte** [] temp\_check = (receivedAdv%2==0)?**new byte**[receivedAdv]:**new byte**[receivedAdv+1];

System.*arraycopy*(tx\_msg, 0, temp\_check, 0, tx\_msg.length);

**int** checkSum = *calculateCheckSum*(temp\_check, receivedAdv-8); tx\_msg[4] = (**byte**)((checkSum & 0xFF00) >> 8);

tx\_msg[5] = (**byte**)(checkSum & 0xFF);

System.*out*.println("in transmitter checksum is "+checkSum); bytesSent = bytesSent + lengthOfDataPart;

**for**(**int** temp=0;temp<tx\_msg.length;temp++) System.*out*.print((**int**)tx\_msg[temp]+" "); System.*out*.println();

/\* send tx\_msg in a datagram to receiver

here am assuming the receiver is at the same machine but at diff port. will be useful for testing.\*/

DatagramPacket sentPacket = **new** DatagramPacket(tx\_msg, tx\_msg.length, InetAddress.*getLocalHost*(), *RECEIVER\_PORT\_NUM*); System.*out*.println("packet sending");

clientSocket.send(sentPacket); System.*out*.println("packet sent");

//creating a new thread that takes care of timeout mechanism.

**new** Transmitter();

// receive reply from receiver

**byte**[] receivedMessage = **new byte**[8];//8- we are just receiving the header back right? - Yes

DatagramPacket receivedPacket = **new** DatagramPacket(receivedMessage,receivedMessage.length); clientSocket.receive(receivedPacket);

*pktreceived*=1; receivedMessage=receivedPacket.getData(); System.*out*.println("received acknowledgment");

**if**( *validate\_ACK*(receivedMessage)==**true**){

// from the message extract the last 2 bytes alone and assign it to

receiverAdv.

}

**byte** rec\_adv0 = receivedMessage[6];

**byte** rec\_adv1 = receivedMessage[7]; receivedAdv = ((**int**)rec\_adv0 << 8) | rec\_adv1;

}

**for**(**int** ind=0;ind<500;ind++)

System.*out*.print(sentMessage[ind]+ " ");

}

/\* This function Validates ACknowledgment Packet by

1. checking the packet type in the header
2. using Internet checksum and confirming that the packet is free of Errors.

\*/

**static boolean** validate\_ACK(**byte**[] receivedACK){

**boolean** hdr\_match,checksum\_match;

//verifying the packet type in header: hdr\_match=**new** Byte("2").equals(receivedACK[1]);

System.*out*.println("(ACK)header match ?"+hdr\_match);

/\*confirming the packet is error free by

* 1. creating a temporary array , so that original Message is not modified when length of Array is odd

when verifyCheckSum function is called

* 1. checking whether verifyCheckSum() returned all 1's

\*/

**int** receivedACKLen=receivedACK.length;

**byte** [] temp\_check = (receivedACKLen%2==0)?**new byte**[receivedACKLen]:**new byte**[receivedACKLen+1]; System.*arraycopy*(receivedACK,0,temp\_check,0,receivedACKLen);

**int** checksum= *verifyCheckSum*(temp\_check, 0); // passing 0 as there is no data System.*out*.println("(ACK)calcaulated checksum for ACK in

transmitter"+checksum);

checksum\_match=(**boolean**)(checksum==65535);

**return** (checksum\_match &hdr\_match);

}

// this function calculates InternetChecksum

**static int** calculateCheckSum(**byte**[] fullPacket, **int** lengthOfData) {

**int** checksum = 0;

**int** dataLength = lengthOfData;

**if**(dataLength % 2 != 0) { dataLength++; fullPacket[dataLength+8-1] = 0;

}

// correct packet - even number of bytes

// calculating the checksum

**long** sum = 0; // to make sure there is no overflow

**for**(**int** ind = 0; ind < (dataLength+8)/2; ind++) {

**if**(ind!=2){ // this is done to prevent the checksum calculation to not include the checksum field.

// the MSB and LSB data in byte form **byte** msbByte = fullPacket[2\*ind]; **byte** lsbByte = fullPacket[2\*ind+1];

// converting them to int (byte is signed!)

**int** msb = msbByte >= 0 ? msbByte : msbByte + 256;

**int** lsb = lsbByte >= 0 ? lsbByte : lsbByte + 256;

// accumulating the checksum (this is two's complement sum!) sum += (msb << 8) + lsb;

}

}

// one's complement sum: adding the carries to the MSBs

// (this may have to be done more than once for a long data packet)

**while**(sum >> 16 != 0)

sum = (sum >> 16) + (sum & 0xffff);

// final checksum value: one's complement of the one's complement sum checksum = (**int**) ~sum;

**return** checksum;

}

//this function returns all 1's if the packet is error free

**static int** verifyCheckSum(**byte**[] fullPacket, **int** lengthOfData) {

**int** checksum = 0;

**int** dataLength = lengthOfData;

**if**(dataLength % 2 != 0) { dataLength++; fullPacket[dataLength+8-1] = 0;

}

// correct packet - even number of bytes

// calculating the checksum

**long** sum = 0; // to make sure there is no overflow

**for**(**int** ind = 0; ind < (dataLength+8)/2; ind++) {

// the MSB and LSB data in byte form

**byte** msbByte = fullPacket[2\*ind];

**byte** lsbByte = fullPacket[2\*ind+1];

// converting them to int (byte is signed!)

**int** msb = msbByte >= 0 ? msbByte : msbByte + 256;

**int** lsb = lsbByte >= 0 ? lsbByte : lsbByte + 256;

// accumulating the checksum (this is two's complement sum!) sum += (msb << 8) + lsb;

}

// one's complement sum: adding the carries to the MSBs

// (this may have to be done more than once for a long data packet)

**while**(sum >> 16 != 0)

sum = (sum >> 16) + (sum & 0xffff); checksum =(**int**) (sum & 0xffff);

**return** checksum;

}

}

RECEIVER:

**package** Receiver;

**import** java.net.DatagramPacket; **import** java.net.DatagramSocket; **import** java.net.InetAddress; **import** java.util.Random;

**public class** Receiver {

**static int** *receiveBufferSize* = 30;//the buffer size to get the bytes from transmitter

**static int** *SENDER\_PORT\_NUM* = 4004; **static int** *RECEIVER\_PORT\_NUM* = 5004; **static int** *msg\_size* = 30;

**public static void** main(String[] args) **throws** Exception {

/\*

* Create a datagram socket to receive data

\*/

DatagramSocket serverSocket = **new** DatagramSocket(*RECEIVER\_PORT\_NUM*);

**final int** ack\_packet = 8; //sending acknowledgement packet

**final int** sender\_port\_number = 5004; //the receiver port number from TX code

**int** rec\_msg = 30; //receive message size

//storing the message from the transmitter in an array

**byte**[] sentMessage = **new byte**[ack\_packet];

//setting the packet type sentMessage[0] = 0x00; sentMessage[1] = 0x02;

**byte**[] rx\_msg = **new byte**[500];

**int** expectedSequenceNumber = 0;//initialising

**int** bytesReceived = 0;//initialising

**while**(bytesReceived < 500) { // till 500 bytes have been received

**byte**[] receivedMessage = **new byte**[*receiveBufferSize*]; DatagramPacket receivePacket = **new**

DatagramPacket(receivedMessage,receivedMessage.length);

// receive a packet using socket.receive() in receivePacket System.*out*.println("waiting for packet"); serverSocket.receive(receivePacket); System.*out*.println("packet received"); receivedMessage=receivePacket.getData();

/\*

\* calculate checksum for the whole packet. That checksum should be all 1's, not the checksum field in the

* + received message directly. just create a method like validatePacket() that does all the
  + operations of validating the packet.

\*/

**boolean** validation = *validatePacket*(receivedMessage);

**if**(validation == **true**) {

System.*out*.println("length of received message="+receivedMessage.length);

**for**(**int** temp=0;temp<receivedMessage.length;temp++) System.*out*.print((**int**)receivedMessage[temp]+" ");

// updating the next expected byte ; expectedSequenceNumber=(receivedMessage.length-8)+1;

//int receivedSequenceNumber = ((int)(receivedMessage[2]<<8)) | receivedMessage[3];

**int** receivedDataLength = ((**int**)(receivedMessage[6]<<8)) | receivedMessage[7];

//expectedSequenceNumber = receivedSequenceNumber + receivedDataLength;

sentMessage[2] = (**byte**)((expectedSequenceNumber& 0xFF00) >> 8); sentMessage[3] = (**byte**)(expectedSequenceNumber & 0xFF);

// having zeroes in checksum field, as it is not going to affect value: sentMessage[4]=0;

sentMessage[5]=0;

/\*

* + - update bytesReceived by adding the length field of receivedMessage
    - update receiveBufferSize with the random number

\*/

// System.arraycopy(receivedMessage,8,rx\_msg,bytesReceived,receivedMessage.length-8);

**for**(**int** ind=bytesReceived ,ind1=8;ind

<bytesReceived+receivedMessage.length-8 & ind <500;ind++,ind1++) rx\_msg[ind]=receivedMessage[ind1];

bytesReceived =bytesReceived+receivedDataLength; Random r=**new** Random(); *receiveBufferSize*=r.nextInt(15)+15;

System.*out*.println("receivebufferssize=="+*receiveBufferSize*); sentMessage[6] = (**byte**)((*receiveBufferSize* & 0xFF00) >> 8);

sentMessage[7] = (**byte**)(*receiveBufferSize* & 0xFF);

no data in ack.

// calculating checksum

**int** checkSum = *calculateCheckSum*(sentMessage, 0); // 0 cos there is

sentMessage[4] = (**byte**)((checkSum & 0xFF00) >> 8); sentMessage[5] = (**byte**)(checkSum & 0xFF);

// sending ACK

DatagramPacket ack = **new** DatagramPacket(sentMessage, sentMessage.length, InetAddress.*getLocalHost*(),*SENDER\_PORT\_NUM* );

serverSocket.send(ack);

}

}

**for**(**int** i=0;i<rx\_msg.length;i++) System.*out*.print((**int**)rx\_msg[i]+" ");

}

**static boolean** validatePacket(**byte**[] receivedMessage) {

**boolean** checksum\_match, hdr\_match , len\_match;

**int** receivedMessageLen=receivedMessage.length;

/\*confirming the packet is error free by

* + 1. creating a temporary array , so that original Message is not modified when length of Array is odd

when verifyCheckSum function is called

* + 1. checking whether verifyCheckSum() returned all 1's

\*/

**byte** [] temp\_check = (receivedMessageLen%2==0)?**new byte**[receivedMessageLen]:**new byte**[receivedMessageLen+1]; System.*arraycopy*(receivedMessage,0,temp\_check,0,receivedMessageLen);

**int** checksum= *verifyCheckSum*(temp\_check, receivedMessage.length-8); checksum\_match=(**boolean**)(checksum==65535);

**if**(!checksum\_match) { // adding these would help debug System.*out*.println("check sum did not match"); **return false**;

}

//verifying the packet type in header:

hdr\_match=**new** Byte("1").equals(receivedMessage[1]) || **new**

Byte("0").equals(receivedMessage[1]);

hdr\_match= hdr\_match && **new** Byte("0").equals(receivedMessage[0]);

**if**(!hdr\_match) {

System.*out*.println("header did not match");

# return false;

}

//verifying length field in header

**int** length = ((**int**)receivedMessage[6] << 8) | receivedMessage[7];

//len\_match=(boolean)(receivedMessageLen==length+8);

len\_match = ((length+8) <= *receiveBufferSize*); // this is the 3rd condition for validation

**if**(!len\_match) {

System.*out*.println("length did not match");

# return false;

}

**return** (checksum\_match & hdr\_match & len\_match);

}

// this function calculates InternetChecksum

**static int** calculateCheckSum(**byte**[] fullPacket, **int** lengthOfData) {

**int** checksum = 0;

**int** dataLength = lengthOfData;

**if**(dataLength % 2 != 0) { dataLength++; fullPacket[dataLength+8-1] = 0;

}

// correct packet - even number of bytes

// calculating the checksum

**long** sum = 0; // to make sure there is no overflow

**for**(**int** ind = 0; ind < (dataLength+8)/2; ind++) {

**if**(ind!=2){ // this is done to prevent the checksum calculation to not include the checksum field.

// the MSB and LSB data in byte form **byte** msbByte = fullPacket[2\*ind]; **byte** lsbByte = fullPacket[2\*ind+1];

// converting them to int (byte is signed!)

**int** msb = msbByte >= 0 ? msbByte : msbByte + 256;

**int** lsb = lsbByte >= 0 ? lsbByte : lsbByte + 256;

// accumulating the checksum (this is two's complement sum!) sum += (msb << 8) + lsb;

}

}

// one's complement sum: adding the carries to the MSBs

// (this may have to be done more than once for a long data packet)

**while**(sum >> 16 != 0)

sum = (sum >> 16) + (sum & 0xffff);

// final checksum value: one's complement of the one's complement sum checksum = (**int**) ~sum;

**return** checksum;

}

//this function returns all 1's if the packet is error free

**static int** verifyCheckSum(**byte**[] fullPacket, **int** lengthOfData) {

**int** checksum = 0;

**int** dataLength = lengthOfData;

**if**(dataLength % 2 != 0) { dataLength++; fullPacket[dataLength+8-1] = 0;

}

// correct packet - even number of bytes

// calculating the checksum

**long** sum = 0; // to make sure there is no overflow

**for**(**int** ind = 0; ind < (dataLength+8)/2; ind++) {

// the MSB and LSB data in byte form

**byte** msbByte = fullPacket[2\*ind];

**byte** lsbByte = fullPacket[2\*ind+1];

// converting them to int (byte is signed!)

**int** msb = msbByte >= 0 ? msbByte : msbByte + 256;

**int** lsb = lsbByte >= 0 ? lsbByte : lsbByte + 256;

// accumulating the checksum (this is two's complement sum!) sum += (msb << 8) + lsb;

}

// one's complement sum: adding the carries to the MSBs

// (this may have to be done more than once for a long data packet)

**while**(sum >> 16 != 0)

sum = (sum >> 16) + (sum & 0xffff); checksum =(**int**) (sum & 0xffff);

**return** checksum;

}

}