**Cleveland State University**

**Department of Electrical Engineering and Computer Science**

**CIS454/554 Data Communication Networks**

**Term Project#1:**

**Implementing a Reliable Data Transfer Protocol**

**Reference website:**

1. <https://academic.csuohio.edu/zhao_w/teaching/CNProjects/RDT/cis454proj1.html>
2. <https://academic.csuohio.edu/zhao_w/teaching/CIS454-S20/cis454.html>

In this project, you will be implementing the rdt3.0 protocol covered in lecture 7. In this project description, we refer this protocol as PAR (Positive Acknowledgement with Retransmission) protocol.

The execution of network protocols is typically event-driven. Such events include the arrival of a message from the lower or higher level, and the expiration of a timer. The event is often called an interrupt.

Consider a layer n and its adjacent layers n+1 and n-1, as shown in the figure above. When layer n+1 has a message to send, it invokes the send interface provided by the layer n and passes the message to layer n. Layer n maintains a send buffer to store message to be sent (and to be retransmitted). If the buffer is already full when layer n+1 tries to send a message, an error will be returned. The send call will trigger an event so that the thread that is executing the layer n protocol can fetch the message in the send buffer and passes it to the lower layer (n-1) afternoon adding a layer n header.

Similarly, when a message arrives at layer n-1, an event is generated and the layer n protocol thread is woken up to handle the message. In general, the message is initially stored in a receive buffer of layer n before it is delivered to layer n+1, for a number of reasons (e.g., to ensure source ordering in TCP). In our PAR protocol, this buffer is not used because the message is delivered right away.

Since layer n-1 might be lossy, layer n must implements a retransmission mechanism to retransmit the messages to be sent, if the acknowledgement does not arrive in time. To do this, the protocol starts a timer each time it transmits (or retransmits) a message. If the timer expires, an event is generated and the protocol thread is woken up to perform the retransmission operation.

Event-driven programming is not trivial. Therefore, you are provided with Java skeleton code and the binary (Java byte code) files for a reference implementation for this project. In the reference implementation, the main thread executes the PAR protocol, and a separate thread is used to pick up messages arrived from UDP in the simulated network layer. The timer is implemented using the facility provided by Java (java.util.Timer and java.util.TimerTask). You are strongly encouraged to build your project based on the skeleton Java code. However, you are free to choose a different programming language if you wish, in which case, you will have to implement the simulation environment as well as the PAR protocol from scratch.

For completeness, the PAR finite state machine specification for the sender and receiver (partial) is shown below:



**Partial finite state machine for the receiver (it is the same as that for rdt2.2).**



**Finite state machine specification for the sender.**

The reference implementation consists of the following files:

[ByteArrayUtils.java](https://academic.csuohio.edu/zhao_w/teaching/CNProjects/RDT/ByteArrayUtils.java) � implements utility methods to convert byte array to integer and vice versa.

/\*

 \* Code taken from

 \* http://www.experts-exchange.com/Programming/Programming\_Languages/

 \* Java/Q\_20273040.html

 \* Posted by PingBanon on 03/11/2005 06:41AM PST

 \*/

public class ByteArrayUtils {

 public static final int NUMBER\_OF\_BITS\_IN\_A\_BYTE = 8;

 public static final short MASK\_TO\_BYTE = 0xFF;

 public static final int SIZE\_OF\_AN\_INT\_IN\_BYTES = 4;

 public static void writeInt( byte[] p\_dest,

 int p\_toWrite )

 {

 assert( p\_dest.length >= SIZE\_OF\_AN\_INT\_IN\_BYTES )

 : "Programming error: p\_dest is too short to hold an int";

 // unrolled loop of 4 iterations

 p\_dest[0] = (byte)(p\_toWrite & MASK\_TO\_BYTE);

 p\_toWrite >>= NUMBER\_OF\_BITS\_IN\_A\_BYTE;

 p\_dest[1] = (byte)(p\_toWrite & MASK\_TO\_BYTE);

 p\_toWrite >>= NUMBER\_OF\_BITS\_IN\_A\_BYTE;

 p\_dest[2] = (byte)(p\_toWrite & MASK\_TO\_BYTE);

 p\_toWrite >>= NUMBER\_OF\_BITS\_IN\_A\_BYTE;

 p\_dest[3] = (byte)(p\_toWrite & MASK\_TO\_BYTE);

 }

 public static int readInt( byte[] p\_src ) // must be of size 4

 {

 assert( p\_src.length >= SIZE\_OF\_AN\_INT\_IN\_BYTES )

 : "Programming error: p\_src is too short to hold an int";

 // unrolled loop of 4 iterations

 int result = ( p\_src[ 0 ] & MASK\_TO\_BYTE );

 result |= ( ( p\_src[ 1 ] & MASK\_TO\_BYTE ) << 8 );

 result |= ( ( p\_src[ 2 ] & MASK\_TO\_BYTE ) << 16 );

 result |= ( ( p\_src[ 3 ] & MASK\_TO\_BYTE ) << 24 );

 return result;

 }

}

[TransportLayer.java](https://academic.csuohio.edu/zhao_w/teaching/CNProjects/RDT/TransportLayer.java) � An abstract class that encapsulates the common data structures and methods for PAR sender and receiver. The abstract method run() is to be extended by the ParSender and ParReceiver classes.

/\*

 \* Author: Wenbing Zhao

 \* Last Modified: 10/4/2009

 \* For EEC484 Project

 \*/

import java.util.\*;

public abstract class TransportLayer {

 public static final int MAX\_SEQ = 1;

 public static final int EVENT\_PACKET\_ARRIVAL = 0;

 public static final int EVENT\_TIMEOUT = 1;

 public static final int EVENT\_MESSAGE\_TOSEND = 2;

 public static final long TIMEOUT = 1000; // in millisecond

 java.util.Timer m\_timer;

 SendTimerTask m\_timerTask = new SendTimerTask();

 // state variable to indicate if we should process an event

 boolean m\_wakeup = false;

 // state variable to indicate the type of event that occurred

 int m\_event = -1;

 LossyChannel m\_lossyChannel = null;

 public TransportLayer(LossyChannel lc) {

 m\_lossyChannel = lc;

 }

 // to be implemented by sub-classes for actual sending side

 // and receiving side protocol

 public abstract void run();

 // simulate 1-bit sequence increment operation

 byte increment(byte seq) {

 byte newseq;

 if(seq < MAX\_SEQ)

 newseq = 1;

 else // seq number wrap around

 newseq = 0;

 return newseq;

 }

 void sendToLossyChannel(Packet p) {

 m\_lossyChannel.send(p.toBytes());

 }

 Packet receiveFromLossyChannel() {

 byte[] receivedData = m\_lossyChannel.receive();

 Packet packet = new Packet(receivedData);

 return packet;

 }

 void startTimer() {

 try {

 m\_timer = new java.util.Timer();

 m\_timer.schedule(new SendTimerTask(), TIMEOUT);

 } catch(Exception e) {}

 }

 void stopTimer() {

 try {

 m\_timer.cancel();

 } catch(Exception e) {}

 }

 public synchronized int waitForEvent() {

 while(!m\_wakeup) {

 try {

 wait();

 } catch (InterruptedException e) {}

 }

 m\_wakeup = false;

 return m\_event;

 }

 public synchronized void onPacketArrival() {

 m\_wakeup = true;

 m\_event = EVENT\_PACKET\_ARRIVAL;

 notifyAll();

 }

 public synchronized void onTimeout() {

 m\_wakeup = true;

 m\_event = EVENT\_TIMEOUT;

 notifyAll();

 }

 public class SendTimerTask extends TimerTask {

 public void run() {

 onTimeout();

 }

 }

}

[Packet.java](https://academic.csuohio.edu/zhao_w/teaching/CNProjects/RDT/Packet.java) � The class that defines the structure of the packet, i.e., the transmission unit of the PAR protocol. Encoding and decoding methods are also provided, together with a simple frame verification method.

/\*

 \* Author: Wenbing Zhao

 \* Last Modified: 10/4/2009

 \* For EEC484 Project

 \*/

// This class describes the packet structure used by the par protocol

public class Packet {

 public static final byte[] PREAMBLE =

 {'E','E','C','4','8','4','F','A'};

 public static final int MAX\_PACKET\_PAYLOAD = 1024;

 public static final int MAX\_PACKET\_SIZE = MAX\_PACKET\_PAYLOAD + 6 +

 PREAMBLE.length; // payload length + header fields + preamble len

 public byte seq; // sequence number for duplicate detection, actually needed 1 bit

 public byte ack; // acked sequence number, actually needed 1 bit

 public int length; // payload length, really need 4 bytes

 public byte[] payload = new byte[MAX\_PACKET\_PAYLOAD];

 private boolean m\_isValid = true;

 public Packet() {

 seq = -1;

 ack = -1;

 length = 0;

 }

 public Packet(byte[] receivedData) {

 // Does the packet carries the right preamble?

 m\_isValid = verifyPacket(receivedData);

 int index = PREAMBLE.length;

 if(m\_isValid) {

 //System.out.println("Valid packet");

 // set seq number

 seq = receivedData[index++];

 // set ack sequence number

 ack = receivedData[index++];

 //System.out.println("b1: "+receivedData[index]);

 //System.out.println("b2: "+receivedData[index+1]);

 //System.out.println("b3: "+receivedData[index+2]);

 //System.out.println("b4: "+receivedData[index+3]);

 // payload length

 byte[] intArray = {receivedData[index], receivedData[index+1],

 receivedData[index+2], receivedData[index+3]};

 index += 4;

 length = ByteArrayUtils.readInt(intArray);

 // the rest is the application payload

 //System.out.println("payload len: "+length);

 for(int i=0; i<length; i++) {

 payload[i] = receivedData[index+i];

 }

 //String recvd = new String(payload);

 //System.out.println("Packet::Received: "+recvd);

 //System.out.println("Got ack, try get another message to send");

 }

 }

 public byte[] toBytes() {

 // we want to construct a byte array consisting

 // a preamble, a sequence number, an ack,

 // a length field (4 bytes), and the payload

 int totalLen = length + 2 + 4 + PREAMBLE.length;

 byte[] data = new byte[totalLen];

 //System.out.println("Send total length: "+totalLen);

 int i = 0;

 for(i=0; i<PREAMBLE.length; i++)

 data[i] = PREAMBLE[i];

 // note that our sequence number is 1-bit, so it fits one byte

 data[i++] = seq;

 data[i++] = ack;

 byte[] intArray = new byte[4];

 ByteArrayUtils.writeInt(intArray, length);

 int k = 0;

 data[i++] = intArray[k++];

 data[i++] = intArray[k++];

 data[i++] = intArray[k++];

 data[i++] = intArray[k++];

 //System.out.println("Packet::toBytes: payload len="+length);

 for(int j=0; j<length; j++,i++)

 data[i] = payload[j];

 // self test on marshalling

 //System.out.println("----");

 //Packet test = new Packet(data);

 //System.out.println("----");

 return data;

 }

 private boolean verifyPacket(byte[] data) {

 boolean verified = true;

 for(int i=0; i<PREAMBLE.length; i++)

 if(data[i] != PREAMBLE[i]) {

 verified = false;

 break;

 }

 return verified;

 }

 public boolean isValid() {

 return m\_isValid;

 }

}

[ParReceiver.java](https://academic.csuohio.edu/zhao_w/teaching/CNProjects/RDT/ParReceiver.java) � The class that implements the PAR receiver protocol. It is a subclass of TransportLayer class. For simplicity, the application sender is integrated with this class.

/\*

 \* Author: Wenbing Zhao

 \* Last Modified: 10/4/2009

 \* For EEC484 Project

 \*/

public class ParReceiver extends TransportLayer{

 public static final int RECEIVER\_PORT = 9888;

 public static final int SENDER\_PORT = 9887;

 public ParReceiver(LossyChannel lc) {

 super(lc);

 }

 public void run() {

 byte nextPacketExpected = 0;

 Packet packetReceived = new Packet();

 Packet packetToSend = new Packet();

 System.out.println("Ready to receive: ");

 while(true) {

 int event = waitForEvent();

 if(EVENT\_PACKET\_ARRIVAL == event) {

 packetReceived = receiveFromLossyChannel();

 // To be completed for task#2

 // PAR protocol implementation: receiver side

 }

 }

 }

 // To be modified for task#5

 //

 // We simply extract the payload and display it as a string in stdout

 void deliverMessage(Packet packet) {

 byte[] payload = new byte[packet.length];

 for(int i=0; i<payload.length; i++)

 payload[i] = packet.payload[i];

 String recvd = new String(payload);

 System.out.println("Received "+packet.length+" bytes: "

 +new String(payload));

 //System.out.println("received payload len: "+recvd.length());

 }

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

 LossyChannel lc = new LossyChannel(RECEIVER\_PORT, SENDER\_PORT);

 ParReceiver receiver = new ParReceiver(lc);

 lc.setTransportLayer(receiver);

 receiver.run();

 }

}

[ParSender.java](https://academic.csuohio.edu/zhao_w/teaching/CNProjects/RDT/ParSender.java) � The class that implements the PAR sender protocol. It is a subclass of TransportLayer class. For simplicity, the application receiver is integrated with this class.

/\*

 \* Author: Wenbing Zhao

 \* Last Modified: 10/4/2009

 \* For EEC484 Project

 \*/

import java.io.\*;

public class ParSender extends TransportLayer{

 public static final int RECEIVER\_PORT = 9888;

 public static final int SENDER\_PORT = 9887;

 public ParSender(LossyChannel lc) {

 super(lc);

 }

 public void run() {

 byte nextPacketToSend = 0;

 Packet packet = new Packet();

 byte[] msgToSend = getMessageToSend();

 if(null == msgToSend)

 return;

 while(true) {

 // To be completed for task#2

 // populate the packet fields

 sendToLossyChannel(packet);

 m\_wakeup = false;

 // To be completed for task#2

 // start timer for retransmission

 int event = waitForEvent();

 if(EVENT\_PACKET\_ARRIVAL == event) {

 packet = receiveFromLossyChannel();

 // To be completed for task#2

 // PAR protocol implementation: sender side

 }

 }

 }

 // To be modified for task#4

 //

 // We get message to send from stdin

 byte[] getMessageToSend() {

 System.out.println("Please enter a message to send: ");

 try {

 BufferedReader inFromUser =

 new BufferedReader(new InputStreamReader(System.in));

 String sentence = inFromUser.readLine();

 if(null == sentence)

 System.exit(1);

 System.out.println("Sending: "+sentence);

 return sentence.getBytes();

 } catch(Exception e) {

 System.out.println("IO error: "+e);

 return null;

 }

 }

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

 LossyChannel lc = new LossyChannel(SENDER\_PORT, RECEIVER\_PORT);

 ParSender sender = new ParSender(lc);

 lc.setTransportLayer(sender);

 sender.run();

 }

}

[LossyChannel.java](https://academic.csuohio.edu/zhao_w/teaching/CNProjects/RDT/LossyChannel.java) � This class simulates the noisy and lossy channel using UDP.

/\*

 \* Author: Wenbing Zhao

 \* Last Modified: 10/4/2009

 \* For EEC484 Project

 \*/

import java.io.\*;

import java.net.\*;

import java.util.\*;

public class LossyChannel {

 private InetAddress m\_IPAddress;

 private DatagramSocket m\_socket = null;

 private int m\_localport = 0;

 private int m\_remoteport = 0;

 private byte[] m\_receiveBuffer = new byte[Packet.MAX\_PACKET\_SIZE];

 private TransportLayer m\_transportLayer = null;

 public LossyChannel(int localport, int remoteport) {

 m\_localport = localport;

 m\_remoteport = remoteport;

 try {

 m\_IPAddress = InetAddress.getByName("localhost");

 m\_socket = new DatagramSocket(localport);

 } catch(Exception e) {

 System.out.println("Cannot create UDP socket: "+e);

 }

 // start the reading thread

 new ReadThread().start();

 }

 public void setTransportLayer(TransportLayer tl) {

 m\_transportLayer = tl;

 }

 public void send(byte[] payload) {

 //

 // To be modified for task#3

 //

 // simulate random loss of packet

 Random rand = new Random();

 int randnum = rand.nextInt(10); // range 0-10

 if(randnum < 3)

 return; // simulate a loss

 try {

 DatagramPacket p =

 new DatagramPacket(payload, payload.length,

 m\_IPAddress, m\_remoteport);

 //System.out.println("LossyChannel::send: "+new String(p.getData()));

 m\_socket.send(p);

 } catch(Exception e) {

 System.out.println("Error sending packet: "+e);

 }

 }

 // Interface provided to the transport layer to pick up message received

 public byte[] receive() {

 return m\_receiveBuffer;

 }

 // Thread to read packets arrived from the network and to notify

 // the transport layer

 public class ReadThread extends Thread {

 public void run() {

 while(true) {

 DatagramPacket p =

 new DatagramPacket(m\_receiveBuffer,

 m\_receiveBuffer.length);

 try {

 m\_socket.receive(p);

 } catch(Exception e) {

 System.out.println("Cannot receive from socket: "+e);

 }

 byte[] receivedData = p.getData();

 //System.out.println("Received packet: "+ receivedData[0]);

 if(m\_transportLayer != null)

 m\_transportLayer.onPacketArrival();

 }

 }

 }

}

**How to run the reference implementation binaries:**

* Download all the jar file ([par.jar](https://academic.csuohio.edu/zhao_w/teaching/CNProjects/RDT/par.jar)) and expand it in your working directory: jar xf par.jar
* First, start ParReceiver: java ParReceiver
* Then, start ParSender: java ParSender
* The ParSender will prompt you to enter a new message when it is ready. Best in mind that the message length is limited to 1024 bytes.

**Tasks:**

To keep the difficulty of the project in the reasonable level, most of the files above will be provided with full implementation. Your tasks include:

1. Thoroughly read and understand the code provided (This task does not apply if you choose to implement this project in a different language)
2. Complete the main protocol loop in the run() method in RarReceiver and ParSender classes.
3. Modify the send() method defined in the LossyChannel class so that you can control the loss rate dynamically. The reference implementation has a hard coded loss rate of 30%, i.e., 3 packets will be lost for every 10 packets sent. There are a number of ways you can use to convey the loss rate information dynamically (i.e., at runtime) to the program, e.g., through command line, environment variables and Java Properties. Please consult with the Java Tutorial for detail: <http://java.sun.com/docs/books/tutorial/essential/environment/config.html>
4. Modify the getMessageToSend() method in ParSender class so that it can fetch packets to send from a file (ASCII file would be fine).
5. Modify the deliverPacket() method in ParReceiver class so that it extracts the packet from  the frame received and write the payload into a file.
6. The above two tasks enable the measurement of the performance of the PAR protocol. You need to instrument the ParSender class to measure the total time required to transmit a big file. If you choose to use an ASCII file, it should include at least 1,000 lines. You need to perform the latency measurement under the following configurations: 0 loss, 10% loss, 20%, and 30%. You can take the measurement on the same machine, or on two computers. The measurement result can be reported in a table or in a figure.

**Deliverables:**

* Source code.
* Your input file for the performance test, and proof for transmission correctness (comparison result for your input and output files).
* Project report describing your implementation and performance measurement results.
* Demonstration.

**Extra credit (3% of the course):**

The reference implementation has only a simple verification test on the correctness of the frames received. It includes a preamble with each frame sent and check the validity of the preamble. This check does not work if the transmission error occurs after the preamble part in a frame. To improve the current implementation, you need to compute a checksum for each frame and include the checksum as a trailer in each frame sent, and verify the checksum of each frame received.

Because UDP ensures no corruption for packets delivered, you need to instrument the LossyChannel class�s send() method to inject corruption into packets to be sent. Again, the corruption incidence should happen randomly with controllable probabilities, just like the segment loss rate.

**Appendix: How to run the reference implementation binaries in Eclipse:**

* Download and install the eclipse on your machine. Please go to eclipse.org and find the right version for you.
* Open eclipse by double click the icon under the installation directory.
* When the Workbench is launched, the first thing you see is a dialog that allows you to select where the workspace should be located. The workspace is the directory where your work will be stored. You can choose the default one or create a new directory by yourself.
* After the workspace location is chosen, the main interface of Eclipse SDK should show up subsequently.
* Then create a new project and select Java Project in the dialog. And click next.





* Type in your project name and click finish. Choose Yes when you see Open Associated Perspective.
* Now you should see the project in the Package Explore on the left side.



* Download the jar file (par.jar) and put it in your workspace -> your project home directory.
* Right click the project name and refresh it, you should see the par.jar under your project name from the Package Explorer.
* Right click the project name and choose Build Path -> Configure Build Path. Then you should see a window of Properties for Project. Then choose Libraries tab, you should see like this



* Click Add JARs if your jar file is under the Project home directory. And find the par.jar file click it and choose OK to go back to the Build Path window. Click OK to finish add jar file.
* Now you should see the par.jar file under Referenced Libraries.
* Click par.jar file and then click RUN button and choose run as Java Application. Or right click par.jar file and choose Run As -> Java Application





* Then it is the time to choose which java application you want to run.
* First, start ParReceiver. You should see the result in the Console (bottom side of the window).
* Find the New Console button, click the drop down list and choose New Console View.





* Right click the Console and choose the Detached for at least one of two consoles. Now you will see both of the consoles with the same thing.



* Right click the par.jar, Run As -> Java Application. Or drop down list of run button (don�t click run button here). Run ParSender. You will see both consoles change to the ParSender. Click Display Select Console for one console. You should see both of ParReceiver and ParSender now.



* The ParSender will prompt you to enter a new message when it is ready. Best in mind that the message length is limited to 1024 bytes.