LEBANESE AMERICAN UNIVERSITY

Electrical and Computer Engineering Dept

COE 431
Computer Networks

Spring 2013

Project I

Objective

For this project, you will design two Java classes called respectively **ReliableUDPSender.java** and **ReliableUDPReceiver.java**, which will be used to exchange data reliably over UDP. To achieve this objective, you will need to devise a protocol that behaves similarly to TCP in the sense that the recipient of data sends acknowledgments back to the sender to indicate that the data has been received correctly. Because your experiments may be in a network environment with no loss of IP packets, you will need to simulate packet loss.

Requirements

A file is to be transferred from the client side (namely, ReliableUDPSender.java) to the server side of the application. Given that the file may be very large, the content of the file may have to be fragmented into multiple datagram packets. The client should grab the resulting packets and send them to the server. After sending each packet, the client waits for a certain amount of time to receive an acknowledgement. If the server's acknowledgment is not received within that time, the packet is retransmitted.
Your code must meet the following requirements:

- The client should be able to transfer the **file name reliably**.
- The client should be able to transfer the **file content reliably**.
- The server must **write** the received content **to a file** with the appropriate file name.
- Your code must be able to transfer a file correctly in the presence of any number of packets **dropped, duplicated, or delayed.**

Feel free to use any of the reliable data transfer mechanisms described in chapter 3 (specifically, section 3.4.1 "Building a Reliable Data Transfer Protocol") in order to match all of the above listed requirements. However, more sophisticated implementations will be rewarded generously!!
You should keep in mind that the following criteria will be used when grading your implementation:

- How long does it take your client to transfer a given file successfully to the server?
- The amount of overhead incurred by your implementation in terms of retransmissions, acknowledgments, etc…

When testing your code, these two performance metrics will be taken into consideration. This goes without saying that better performance in this regard will result in a higher grade.

Finally, note that your client should be able to handle the simulated packet loss and large transmission delays gracefully and recover from these events without crashing.

Simulating Packet Loss and Transmission Delay

Because packet loss is rare or even non-existent in typical campus network, you are given below an implementation for a server that injects artificial loss to simulate the effects of network packet loss. The server has a parameter called **LOSS_RATE** that determines what percentage of packets should be lost. Moreover, the server has another parameter called **AVERAGE_DELAY** that is used to simulate transmission delay from sending a packet across the Internet. You should try to set AVERAGE_DELAY to a large value when testing your client and server on the same machine. Study the supplied code carefully, as it will help you write your own **ReliableUDPReceiver** code.

```
import java.io.*;
import java.net.*;
import java.util.*;
//Server to process ping requests over UDP.
public class PingServer {
   private static final double LOSS_RATE = 0.3;
   private static final int AVERAGE_DELAY = 100; // ms
   public static void main(String[] args) throws Exception {
      // read command line argument.
      if (args.length != 1) {
         System.out.println("Required arguments: port");
         return;
      }
      int port = Integer.parseInt(args[0]);
      // Create random number generator to simulate
      // packet loss and network delay.
      Random random = new Random();
      // Create a datagram socket for receiving/receiving
      // UDP segments through the obtained port number.
      DatagramSocket socket = new DatagramSocket(port);
      while (true) {
         // Create a packet to hold received UDP segments
         DatagramPacket request =
               new DatagramPacket(new byte[1024], 1024);
         // idle until receiving a UDP packet.
         socket.receive(request);
         // Print received data.
         printData(request);
         // Decide whether to reply,
         // or simulate packet loss.
         if (random.nextDouble() < LOSS_RATE) {
            System.out.println("   Reply not sent.");
            continue;
         }
         // Simulate random network delay.
         Thread.sleep(
            (int)(random.nextDouble() * 2 * AVERAGE_DELAY));
           // Send reply.
           InetAddress clientHost = request.getAddress();
           int clientPort = request.getPort();
           byte[] buf = request.getData();
           DatagramPacket reply = new DatagramPacket(buf,
            buf.length, clientHost, clientPort);
           socket.send(reply);
           System.out.println("   Reply sent.");
        }
      }
      //Print data to the standard output stream.
      private static void printData(DatagramPacket request) throws
      Exception {
            // get references to the packet's array of bytes.
            byte[] buf = request.getData();
            // Wrap the bytes in a byte array input stream,
            // so that you read the data as stream of bytes.
            ByteArrayInputStream bais = new
            ByteArrayInputStream(buf);
           // Wrap the byte array output stream in an input
            // stream reader, so that you read the data as a
            // stream of characters.
            InputStreamReader isr = new
                                 InputStreamReader(bais);
           // Wrap the input stream reader in a buffered
            // reader, so that you read the character data one
            // line at a time. (A line is a sequence of chars
            // terminated by any combination of \r and \n.)
            BufferedReader br = new BufferedReader(isr);
            //The message data is contained in a single
            // line, so read this line.
            String line = br.readLine();
            // Print host address and data received from it.
            System.out.println(
               "Received from " +
               request.getAddress().getHostAddress() +
               ": " +
               new String(line));
         }
      }
```

As you may have noticed, the supplied server code uses an infinite loop for the processing of the incoming packets. When a packet arrives at the server, the server simply sends its encapsulated data back to the client side of the application.
Because UDP is an unreliable protocol, some of the packets sent to the server may be lost. For this reason, the client cannot wait indefinitely for a reply to his lost message. You should have your **ReliableUDPSender** client wait for a certain amount of time for a reply; if no reply is received within that time, the client should assume that the packet was lost during transmission across the network. You will need to research the API for the **DatagramSocket** class to find out how to set the timeout value on a datagram socket. **Bear in mind that a timeout event can be detected via the “SocketTimeoutException” class. More specifically, after setting the timeout interval for your socket by means of the “setSoTimeout” method, the “receive” method throws a “SocketTimeoutException” when the preset timer expires.**
When developing your code, you should run both the client side and the server side of your application on your machine, and test your client by transferring a file to **localhost** (or, 127.0.0.1). After you have fully debugged your code, you should see how your application communicates across the network with a server running on a separate machine that belongs to another member of the class. Students that will go that extra mile will also be rewarded.

What you should do

Your job, should you choose to accept it ☺, is to write two programs: 1) a sending program that sends the file across the network and that you are required to call **ReliableUDPSender.java** and 2) a receiving program that receives the file and stores in its local disk storage. You may not use TCP as your transport layer protocol. This means that you have to construct the packets and acknowledgments yourself and interpret the incoming packets yourself.

You should write the client so that it starts with the following command:

```
java ReliableUDPSender -r <recv_host>:<recv_port> -f <filename>
```

Where `recv_host` is the name of the computer that houses the server, `recv_port` is the UDP port number the server is listening to, and `filename` is the name of the file to send.

To make the grading and debugging easier, your client program should produce the following messages:

- Each time a sender sends a new packet, it should print the following message out: `[send data]: start (length)`, where start is the beginning offset of the file sent in the packet, and length is amount of data sent in that packet.
- You may also print some messages of your own to indicate that acknowledgment messages were received, time out event occurred, etc… these messages will eventually depend on the reliability mechanism that you decide to use. But, please make your messages pointed, concise, and above all readable.

The command line syntax for launching the server side of the application should be as follows:

```
java ReliableUDPSender –p <recv_port>
```

Where `recv_port` is the UDP port the server is listening to.

Again, to aid in the grading and debugging, your receiving program should print out the following messages to the screen:

- When the receiver receives a valid data packet, it should print: `[recv data] start (length) status`, where status is either `ACCEPTED (in-order)`, or `IGNORED`.
- Similarly to the client side, you may add your own output messages.
- The receiver should use a different extension to store the file as this will allow you to use the same directory for both the receiver and the sender.

Finally, both the sender and the receiver should print out a message to indicate that file transfer is complete, and then exit:

```
[completed]
```

Other Resources

You must not copy any code from the internet and you must make sure to be explicit about what resources you use, including web tutorials and any other web resources.

What to turn in?

The project is due at the beginning of class on the due date. You have to turn in the following material in both hard and soft copies.

Criteria	Percentage
Documentation of your solution including explanations and illustrations in one or two pages along with short write-up of questions and/or problems that you encountered while doing this assignment.	2 pts (10%)
Source code that contains an appropriate amount of comments. Well-organized and correct code receives 16 pts, messy yet working code receives 12 pts, code with bugs receives 4 pts, and incomplete code receives 1 pt.	16 pts (80 %)
Execution output such as a snapshot of the contents of standard output. A correct output receives 2 pts, the one with minor errors receives 1 pts, and an incomplete output receives 0 pts.	2 pts (10%)
Total	20pts (100%)