Programming Assignment 3: UDP Pinger Lab

In this lab, you will study a simple Internet ping server written in the Java language, and implement a corresponding client. The functionality provided by these programs are similar to the standard ping programs available in modern operating systems, except that they use UDP rather than Internet Control Message Protocol (ICMP) to communicate with each other. (Java does not provide a straightforward means to interact with ICMP.)

The ping protocol allows a client machine to send a packet of data to a remote machine, and have the remote machine return the data back to the client unchanged (an action referred to as echoing). Among other uses, the ping protocol allows hosts to determine round-trip times to other machines.

You are given the complete code for the Ping server below. Your job is to write the Ping client.

Server Code

The following code fully implements a ping server. You need to compile and run this code. You should study this code carefully, as it will help you write your Ping client.

```
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;  // milliseconds

  public static void main(String[] args) throws Exception
  {
    // Get command line argument.
    if (args.length != 1) {
      System.out.println("Required arguments: port");
```

```
      return;
    }
    int port = Integer.parseInt(args[0]);

    // Create random number generator for use in simulating
    // packet loss and network delay.
    Random random = new Random();

    // Create a datagram socket for receiving and sending UDP packets
    // through the port specified on the command line.
    DatagramSocket socket = new DatagramSocket(port);

    // Processing loop.
    while (true) {
      // Create a datagram packet to hold incomming UDP packet.
      DatagramPacket request = new DatagramPacket(new byte[1024], 1024);

      // Block until the host receives a UDP packet.
      socket.receive(request);

      // Print the recieved data.
      printData(request);

      // Decide whether to reply, or simulate packet loss.
      if (random.nextDouble() < LOSS_RATE) {
        System.out.println("   Reply not sent.");
        continue;
      }

      // Simulate 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 ping data to the standard output stream.
   */
  private static void printData(DatagramPacket request) throws Exception
  {
    // Obtain references to the packet's array of bytes.
    byte[] buf = request.getData();

    // Wrap the bytes in a byte array input stream,
    // so that you can read the data as a stream of bytes.
    ByteArrayInputStream bais = new ByteArrayInputStream(buf);

    // Wrap the byte array output stream in an input stream reader,
    // so you can read the data as a stream of characters.
    InputStreamReader isr = new InputStreamReader(bais);

    // Wrap the input stream reader in a bufferred reader,
    // so you can read the character data a 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) );
  }
}
```

The server sits in an infinite loop listening for incoming UDP packets. When a packet comes in, the server simply sends the encapsulated data back to the client.

Packet Loss

UDP provides applications with an unreliable transport service, because messages may

get lost in the network due to router queue overflows or other reasons. In contrast, TCP provides applications with a reliable transport service and takes care of any lost packets by retransmitting them until they are successfully received. Applications using UDP for communication must therefore implement any reliability they need separately in the application level (each application can implement a different policy, according to its specific needs).

Because packet loss is rare or even non-existent in typical campus networks, the server in this lab injects artificial loss to simulate the effects of network packet loss. The server has a parameter LOSS_RATE that determines which percentage of packets should be lost.

The server also has another parameter AVERAGE_DELAY that is used to simulate transmission delay from sending a packet across the Internet. You should set AVERAGE_DELAY to a positive value when testing your client and server on the same machine, or when machines are close by on the network. You can set AVERAGE_DELAY to 0 to find out the true round trip times of your packets.

Compiling and Running Server

To compile the server, do the following:

```
javac PingServer.java
```

To run the server, do the following:

```
java PingServer port
```

where port is the port number the server listens on. Remember that you have to pick a port number greater than 1024, because only processes running with root (administrator) privilege can bind to ports less than 1024.

Note: if you get a class not found error when running the above command, then you may need to tell Java to look in the current directory in order to resolve class references. In this case, the commands will be as follows:

```
java -classpath . PingServer port
```

Your Job: The Client

You should write the client so that it sends 10 ping requests to the server, separated by approximately one second. Each message contains a payload of data that includes the keyword PING, a sequence number, and a timestamp. After sending each packet, the client waits up to one second to receive a reply. If one seconds goes by without a reply from the server, then the client assumes that its packet or the server's reply packet has been lost in the network.

Hint: Cut and paste PingServer, rename the code PingClient, and then modify the code.

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

```
java PingClient host port
```

where host is the name of the computer the server is running on and port is the port number it is listening to. Note that you can run the client and server either on different machines or on the same machine.

The client should send 10 pings to the server. Because UDP is an unreliable protocol, some of the packets sent to the server may be lost, or some of the packets sent from server to client may be lost. For this reason, the client can not wait indefinitely for a reply to a ping message. You should have the client wait up to one second for a reply; if no reply is received, then the client should assume that the packet was lost during transmission across the network. You will need to research the API for DatagramSocket to find out how to set the timeout value on a datagram socket.

When developing your code, you should run the ping server on your machine, and test your client by sending packets 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 ping server run by another member of the class.

Message Format

The ping messages in this lab are formatted in a simple way. Each message contains a sequence of characters terminated by a carriage return character (r) and a line feed character (n). The message contains the following string:

```
PING sequence_number time CRLF
```

where sequence_number starts at 0 and progresses to 9 for each successive ping message sent by the client, time is the time when the client sent the message, and CRLF represent the carriage return and line feed characters that terminate the line.

Optional Exercises

When you are finished writing the client, you may wish to try one of the following exercises.

1) Currently the program calculates the round-trip time for each packet and prints them out individually. Modify this to correspond to the way the standard ping program works. You will need to report the minimum, maximum, and average RTTs. (easy)

2) The basic program sends a new ping immediately when it receives a reply. Modify the program so that it sends exactly 1 ping per second, similar to how the standard ping program works. Hint: Use the Timer and TimerTask classes in java.util. (difficult)

3) Develop two new classes ReliableUdpSender and ReliableUdpReceiver, which are used to send and receive data reliably over UDP. To do this, you will need to devise a protocol (such as TCP) in which the recipient of data sends acknowledgements back to the sender to indicate that the data has arrived. You can simplify the problem by only providing one-way transport of application data from sender to recipient. Because your experiments may be in a network environment with little or no loss of IP packets, you should simulate packet loss. (difficult)