CS433/533 Homework Assignment Two: UDP Client/Server

This assignment gives you a chance to become familiar with the basic Java UDP socket programming interface, Java sleep or (java Timer), and programming using UDP for DNS.

Due 11:55 PM, Monday, Oct. 1, 2018 by uploading to Canvas.

Rubric

[Part 1: Coding; 60 points]

Many network applications need a capability for two clients to ping each other to determine the round-trip time, loss rate, and bandwidth between them. For example, a Skype client A pings another Skype client B before deciding if A should use B as a relay. There are multiple tools available (e.g., speedtest.net), either on desktops or mobile devices, that provide such a functionality. In this part, you will implement a ping server and a corresponding client, using UDP programming.

The ping protocol allows a client machine to send a sequence of packets of data to a remote machine, and have the remote machine return the data back to the client (an action referred to as echoing).

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

Server Code: The following code is your starting code for a ping server. You should study this code carefully, as it will help you write your Ping client and extend the server code.

```
// PingServer.java
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 receives a UDP packet.
         socket.receive(request);

         // Print the received data, for debugging
         printData(request);

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

         // Simulate prorogation 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.");
     } // end of while
   } // end of main

   /*
    * 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**: reader/writer handles
      // 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) );
          } // end of printData
        } // end of class
```

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

Packet Loss and Delay: 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 assignment injects artificial losses to simulate the effects of network packet losses. 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 the propagation delay. 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 example server, do the following:

```
javac PingServer.java
```

To run the example 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 the 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
```

Now we specify your job.

Message Format: The ping messages from a client to the server in this assignment are formatted in the following way:

```
PING sequence_number client_send_time passwd CRLF
```

where PING is the 4 ASCII characters, sequence_number is 2 bytes, and starts at 0 and increases by 1 for each successive ping message sent by the client; client_send_time is 8-byte local timestamp (in the format of Java currentTimeMill) encoding when the client sends the message; passwd is ASCII so that a server echoes only valid requests, and CRLF represents the carriage return and line feed characters that terminate the line.

The reply from a server to a ping message is formatted in the following way:

```
PINGECHO sequence_number client_send_time passwd CRLF
```

where PINGECHO is the 8 ASCII characters, sequence_number, client_send_time, and passwd are echoes of client message.

It is recommended that you read Chapter 4 of Java Network Programming on i/o issues.

Your Client: You should write the client so that it sends 10 ping requests to the server, at a pace of one request per second (Hint: you can Use the Timer and TimerTask classes in java.util). After sending each request, 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 either its request or the server's reply has been lost in the network.

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

```
java PingClient host port passwd
```

where host is the name of the computer that the server is running on and port is the port number that the server 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.

The output of your client should report the minimum, maximum, and average RTTs. It should also report the loss rate.

When developing your code, you may want to 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.

Your Server: Please modify the server program to check that the password is valid. The valid password should be provided when the server starts:

```
java PingServer port passwd [-delay delay] [-loss loss]
```

[Part 2: Design Questions; 20 points]

- **[2a; 8pt]** Please discuss how you tested and validated your program in Part 1. In particular, please describe:
    - How did you handle encoding/decoding to make sure that your code works across platforms?

- How may you extend the program (client and server) to measure the bandwidth (from client to server, and reverse)? A reference is to use packet pair to estimate bandwidth: http://www.usenix.org/event/usits01/full_papers/lai/lai_html/

- Is there any way you can extend the program to synchronize the clocks of the client and the server? What is the accuracy that you may get?

- **[2b; 7pt]** Please answer the following questions on DNS implementation:

  - We mentioned that each DNS packet includes an identification field to handle reply forwarding. One implementation is the following. Assume that a DNS server receives a DNS request R in UDP with src IP being h, src port being p, and identification being id. The server needs to generate a query to another server to answer R. Then the server (1) stores in a hash map so that id maps to (h, p), and (2) uses the identification field id specified in R for the new query. Does this work? Please justify your answer.

  - Assume that a DNS server needs to wait for, on average, 300 ms for the reply of its query that it forwarded. One concern is that the identification numbers may become exhausted when a new query that needs to be forwarded arrives. What is an expression to compute the exhausting likelihood?

  - DNS servers depend heavily on caching to improve scalability. Given a DNS name a.b.c.d (e.g., cicada.zoo.cs.yale.edu) that should be resolved. In what order should a DNS server search its cache before querying other servers?

- **[2c; 5pt]** DNS is a major security target. Please describe, briefly, a scheme, to take down a country's Internet access through an attack on DNS.

[Part 3: Playing with Tools and Observe; 20 points]

- **[3a 12pt]** Please use the dig DNS command and other tools to (1) find 1 gmail server; (2) check whether the IP address 173.194.1.1 is an authorized mail transfer agent for gmail; and (3) give a public key used by Google for DKIM signature. For (3), you will need a gmail message with complete headers. Please show the command outputs to support your answer.

- **[3b 8pt]** DNS allows delegation of the name space hierarchy. Please use the dig tool to find out one sequence of name servers to resolve cicada.cs.yale.edu to its IP address. Please capture the sequence of commands and outputs of using dig. You may start with

```
dig +norecurse @a.root-servers.net cicada.cs.yale.edu A
```

  If multiple name servers are returned, please pick one and continue the process.

Part 4: Submission

Please use Canvas to submit your homework. Your submission should include the source code for Part 1, and a file named Report that answers the discussion questions in Part 2 and result from Part 3. The report file can be either in .txt or .pdf format.

Last Update: Sept. 19, 2018 06:33:30 PM -0400