# Program 3 – UDP Ping Server and Client (30 points)

**Due Date:** Wednesday, Nov. 20 by 11:59pm

**Program Submission:** submit all the source code (*.java) to Canvas.

## Program Description

You will develop a simple Internet **Ping Server** and a corresponding **Client**. The functionality provided by the Client and Server are similar to the standard ping programs available in modern operating systems. However, your Client and Server will be using **UDP** (instead of ICMP) to communicate with each other. The Server sits in an infinite loop listening for incoming UDP packets. When a packet comes in, the Server simply retrieves the payload, encapsulates the payload in an UDP packet, and sends the packet back to the Client. In other words, the Server is **echoing** the Client's message. In this case, your Client is a **Pinger**. It must send 10 ping messages (UDP packets) consecutively to the Server and record the **RTT** (round-trip time) when the Server echoes back the UDP packet. You MUST use an UDP packet size of **512 bytes**. The Client must output the **minimum RTT**, **maximum RTT,** and the **average RTT** at the end of program execution. I have a server running at **constance.cs.rutgers.edu, 5530** for you to test your client (the pinger.)

## The Ping Server

You must have a **PingServer class** providing UDP service. You can use any port number between 5520 and 5540 to receive the UDP packets. The Server MUST inject artificial loss to simulate the effects of network packet loss. That is, you must define a constant **LOSS_RATE = 0.3** that determines the percentage of packets should be lost, and a constant **AVERAGE_DELAY = 100** (milliseconds) to simulate transmission delay. Set **AVERAGE_DELAY = 0** to find out the true RTT of your packets. Here are some useful methods:

```
//generate a random number between 0 and 1; it's a packet loss if the random number is
//less than LOSS_RATE
Random random = new Random(new Date().getTime());
//create the socket for receiving UDP packets
DatagramSocket udpSocket = new DatagramSocket(PORT_NUMBER);
//allocate the memory space for an UDP packet
byte[] buff = new byte[PACKET_SIZE];
//make an empty UDP packet
DatagramPacket inpacket = new DatagramPacket(buff, PACKET_SIZE);
//receive the next UDP packet
udpSocket.receive(inpacket);
//simulate transmission delay; DOUBLE = 2
Thread.sleep((int)(random.nextDouble() * DOUBLE * AVERAGE_DELAY));
//make an outgoing UDP packet
DatagramPacket outpacket =
      new DatagramPacket(payload, payload.length, clientAddr, clientPort);
//send an UDP packet
udpSocket.send(outpacket);
```

**The Client**

Your Client should send **10 ping messages** to the Server, with an **one second** (1000 milliseconds) **interval**. 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 second 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. And the Client prints an error message identifying packet loss to the system output. You will need **setSoTimeout()** method in **DatagramSocket** class to set the timeout value on a datagram socket. Set **5 second timeout** (5000 milliseconds) for collecting pings after 10 pings were sent, just to wait for possible delayed replies from the Server. You may include a **PingClient class**, a **PingMessage class**, and an **UDPPinger class**.

❖ **PingMessage class**

Use this class to create a ping message to be sent to the Server. The format of a ping message is shown below. You must provide the following public methods.

```
public PingMessage(InetAddress addr, int port, String payload)   //constructor
public InetAddress getIP()  //get the destination IP address
public int getPort()  //get the destination port number
public String getPayload()  //get the content of the payload
```

| Destination IP address | Destination Port Number | Payload |
|---|---|---|

The **payload** part of each PING message is a **String** with the following format. You can use the **getTime()** method in **Date** class to get the current time as the timestamp.

| PING | SPACE | Sequence Number | SPACE | timestamp |
|---|---|---|---|---|

❖ **UDPPinger class**

Use an instance of this class to send a ping message and receive the echo message from the Server. You must provide 2 public methods. You may need **.getAddress()**, .**getData()** and **.getPort()** in **DatagramPacket** class.

```
/**
   This method sends an UDP packet with an instance of PingMessage.
   Use the constructor
       DatagramPacket(byte[] payload, int length, InetAddress address, int port)
   to construct an UDP packet and send the UDP packet.
*/
public void sendPing(PingMessage ping)

//This method receives the UDP packet from the Server. This method may throw
// SocketTimeoutException.
public PingMessage receivePing()
```

❖ **PingClient class**

This is the Client class.This class extends **UDPPinger class**, and implements the Java **Runnable interface**. That is, you MUST implement the **run()** method. In the **run()** method, you MUST

(1) Create an instance of **DatagramSocket**.
(2) Set the time out for the socket to be 1 second (1000 milliseconds)
(3) Set up a for loop to send 10 PING messages, record the replies from the Server, and compute the RTT.
(4) Wait additional 5 seconds (set time out to 5 seconds) after 10 PINGs are sent (if less than 10 replies are received), just in case replies are on the way.
(5) Compute and output the average RTT, minimum and maximum RTTs. Use the following formula. Set the RTT to 1000 milliseconds in case of a packet loss.

```
RTT = (current timestamp) – (previous timestamp when sending the packet)
```

**Program Requirement**

1. Your programs MUST NOT crash under any situation.
2. You MUST try-catch everything and display appropriate error messages identifying the specific exception.
3. You MUST display information about ping messages on Server's system output, and on Client's system output.

**Sample output for the Server**

```
Ping Server running....
Waiting for UDP packet....
Received from: /127.0.0.1 PING 0 1572707989954
Packet loss...., reply not sent.
Waiting for UDP packet....
Received from: /127.0.0.1 PING 1 1572707990988
Reply sent.
Waiting for UDP packet....
Received from: /127.0.0.1 PING 2 1572707991636
Packet loss...., reply not sent.
Waiting for UDP packet....
Received from: /127.0.0.1 PING 3 1572707992637
Reply sent.
Waiting for UDP packet....
Received from: /127.0.0.1 PING 4 1572707992660
Reply sent.
Waiting for UDP packet....
Received from: /127.0.0.1 PING 5 1572707992981
Packet loss...., reply not sent.
Waiting for UDP packet....
Received from: /127.0.0.1 PING 6 1572707993983
Reply sent.
Waiting for UDP packet....
Received from: /127.0.0.1 PING 7 1572707994937
Packet loss...., reply not sent.
Waiting for UDP packet....
Received from: /127.0.0.1 PING 8 1572707995938
Reply sent.
Waiting for UDP packet....
Received from: /127.0.0.1 PING 9 1572707996880
Reply sent.
Waiting for UDP packet....
```

## Sample output for the Client

```
Contacting host: localhost at port 5530
receivePing...java.net.SocketTimeoutException: Receive timed out
Received packet from /127.0.0.1 5530 Sat Nov 02 11:19:51 EDT 2019
receivePing...java.net.SocketTimeoutException: Receive timed out
Received packet from /127.0.0.1 5530 Sat Nov 02 11:19:52 EDT 2019
Received packet from /127.0.0.1 5530 Sat Nov 02 11:19:52 EDT 2019
receivePing...java.net.SocketTimeoutException: Receive timed out
Received packet from /127.0.0.1 5530 Sat Nov 02 11:19:54 EDT 2019
receivePing...java.net.SocketTimeoutException: Receive timed out
Received packet from /127.0.0.1 5530 Sat Nov 02 11:19:56 EDT 2019
Received packet from /127.0.0.1 5530 Sat Nov 02 11:19:57 EDT 2019
PING 0: false RTT: 1000
PING 1: true RTT: 648
PING 2: false RTT: 1000
PING 3: true RTT: 23
PING 4: true RTT: 321
PING 5: false RTT: 1000
PING 6: true RTT: 954
PING 7: false RTT: 1000
PING 8: true RTT: 942
PING 9: true RTT: 854
Minimum = 23ms, Maximum = 1000ms, Average = 774.2ms.
```

## Program Grading

| Exceptions/Violations | Each Offense | Max Off |
|---|---|---|
| Did not implement the server, or the server doesn't run | 15 | 15 |
| Did not implement the client, or the client doesn't run | 15 | 15 |
| Client malfunction, or not using 512-byte chunks | 2 | 6 |
| Server malfunction, or not simulating packet loss | 2 | 6 |
| Incorrect minimum, maximum, average RTT | 2 | 6 |
| Incorrect ping message format | 1 | 3 |
| Improper exception handling | 1 | 3 |
| Improper messages on Sever console output | 1 | 3 |
| Improper messages on Client console output | 1 | 3 |