Lab Exercise 6: Monitor with UDP Socket[1]

(Total: 30 points)

OBJECTIVE:

The objective of this lab is to further explore client/server programming with sockets. A simple UDP-based application will be developed to monitor the round-trip time (RTT) between a client and a server. Classic algorithms will be used to estimate the average RTT and deviation of RTT.

LAB ACTIVITIES:

1. Read carefully the source code MonitorServer.java, which echoes requests from a client using user datagram protocol (UDP). In the server, a received message is only responded at a probability. This is to simulate the effect that a message from the client or the reply from the server can be lost during transmission. Recall UDP does not provide automatic retransmissions. It is not guaranteed that the server receives every request from the client, while the client is not guaranteed to receive every reply from the server. In addition, before the server sends back an echo response, a random delay is introduced to simulate the delay over the network. The reply from the server is just a capitalized version of the received message from the client. You should run the server without any input argument.
2. Next, you need to write a client class MonitorClient.java, which can communicate with the above server (see the following figure) to implement the following required functionality.

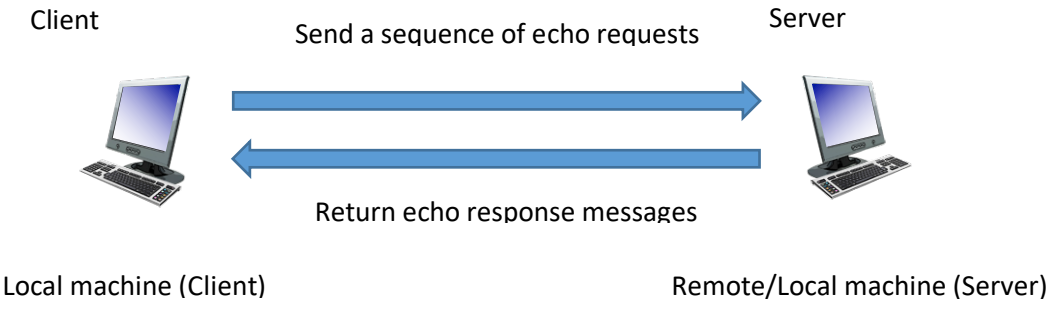

Fig. 1. Architecture of the client-server application.

   a. The client needs to send a sequence of echo requests to the target server over UDP, and receive corresponding echo replies back from the server. Each request is "Hello i", where i is an identification number starting from 0. Thus, the identification numbers for n requests should be 0, 2, …, n-1. From the sending time and the receiving time for each pair of successful request and response, the client measures the delay between the client and the server **in milliseconds**, which is the round-trip time (RTT).

      Note: If you use regex to parse the response messages, you may also append a blank space to the end of each request if that is easier.

   b. As UDP does not invoke automatic retransmission, a message from the client or the server is not guaranteed to be received successfully. The client should wait up to 1 second for a reply from the

[1] The lab materials are adapted from the textbooks "Computer Networking – A Top-Down Approach" by James Kurose and Keith Ross and the materials provided with them. They can only be used by students who registered for this course. Reproduction outside of this course use is prohibited.

server; if no reply is received after 1 second, the client should continue to send the next request.

Note that there can be zero, one, or more than one received replies that are waiting to be fetched. A received reply is not necessary for the most recent request but can be a delayed reply for a former request. You need to decode each reply and pair it correctly with the corresponding request.

Hint: Think about using the `setSoTimeout()` method of `DatagramSocket` class. Then, you need to throw and catch `SocketTimeoutException`.

c. After all echo requests are sent out by the client, the client should continue to monitor incoming messages for a reasonable time period since some replies from the server may be substantially delayed. If no more message comes in for 5 seconds, the client should terminate the current test session. Again, refer to the above hint.

d. At the end of a test session, the client should print the result for each echo request, which is either "no reply" or an RTT measurement if the reply from the server is received. For example:

```
Request 0: RTT = 682
Request 1: RTT = 1989
Request 2: RTT = 2792
Request 3: RTT = 3474
Request 4: RTT = 4354
Request 5: no reply
Request 6: no reply
Request 7: RTT = 2266
```

e. With each RTT measurement, the client updates the EstimatedRTT and DevRTT using the exponentially weighted moving average algorithm in TCP with weights $\alpha = 0.125, \beta = 0.25$, respectively. For the first RTT measurement, denoted by SampleRTT, initialize EstimatedRTT and DevRTT as follows: $EstimatedRTT = SampleRTT, DevRTT = SampleRTT/2$.

f. The `main()` method of class `MonitorClient` should take 2 input arguments: 1) hostname or IP address of server; and 2) port number of server. If you run the test with both the client and the server running on the same computer, then the first input argument should be 127.0.0.1, which represents the localhost.

g. The client needs to work correctly even if not a single reply is received from the server.

3. Now, run a test session with 40 echo requests sent from your client to the sever. Obtain the RTT samples, which are measured from the echo replies successfully received by the client.

4. Although your client should work correctly if not any reply is received, at a high probability you should be able to obtain a number of RTT samples, EstimatedRTT, and DevRTT from the above test session. Then, generate two plots: 1) the first plot includes two curves, which show how RTT samples and EstimatedRTT vary with the update numbers; and 2) the other also includes two curves, which are RTT samples and DevRTT vs. the update numbers. In the first plot, the Y-axis shows both SampleRTT and EstimatedRTT with two curves, while the X-axis is the number of updates (i.e., 0, 2, …, n-1, where n is the total number of RTT samples). In the second plot, the X-axis is the same as the first plot, but the Y-axis shows SampleRTT and DevRTT with two curves. **In the plots, simply skip the requests without replies, for which the client cannot get RTT samples.**

   You can copy the results obtained above and then use an external software such as MATLAB, Excel, Gnuplot, and so on to generate the required plots. You can also use a Java plotting library such as JFreeChart. You do NOT need to include the plotting code in the class MonitorClient.java.

LAB SUBMISSION:

Go through the above lab activities and write your own **client program in Java** according to the requirements. Please add comments in your code and employ good programming practices.

- For this lab, **write a lab report using the posted Word template.** Convert the lab report to a PDF file and submit it to the dropbox on D2L by the due time.
- Also, **zip the Java source code (the original ".java" files) of your client and the given server** in a zip file and submit it to the dropbox on D2L by the due time.