Department of Electrical Engineering
& Computer Science
York University, Fall 2016

CSE 3214: Socket Programming Project

Due Date: 12:00 Noon of Thursday, April 14, 2016.

Late submissions will not be accepted.

1. Java Ping Client (UDP Socket Programming) [50 points]

In this question, you will be given the complete code for a simple Internet-Ping Server (at http://www.cse.yorku.ca/course/3214/PingServer.zip). Your responsibility will be to implement a corresponding Internet-Ping Client. The functionalities of the provided Ping Server, and your Ping Client, should be similar to the standard Ping programs available in modern operating systems, except that your programs should use UDP rather than ICMP protocol to communicate with each other.

By closely studying the code for Ping Server, you will be able to make the following observations:

- *Packet loss functionality*: Because packet loss occurs rarely in typical campus networks, the server injects artificial loss to simulate the effect of network packet loss. The server's LOSS_RATE parameter determines the percentage of lost packets.
- *Average delay functionality*: The server' parameter AVERAGE_DELAY is used to simulate transmission delay from sending a packet across the Internet. The parameter should be set to a positive value when testing your client and server on the same machine, or when machines are close by on the network.

To compile and run the server, do the following:

```
>   javac PingServer.java
>   java PingServer <port>
```

where port is the port number the server listens on. Remember – you have to pick a port number greater than 1024.

Your Job:

You should write the client so that it sends 10 ping requests to the server, separated by approximately one second. Each message will contain a payload of data that includes the keyword PING, the message sequence number, and the message timestamp. After sending each packet, the client will wait up to one second to receive a reply. If one second goes by without a reply from the server, then the client will assume that its packet or the server's reply packet has been lost in the network. (For the last part, you will need to research the API for DatagramSocket to find out how to set the timeout value on a datagram socket.)

You should write the client so that it can be evoked with the following command:

```
>   java PingClient <port>
```

where 'port' is the number of the port through which the client sends messages to the server. Within the client program, the following user input should initiate the packet transmission procedure:

```
>   ping <host> <port>
```

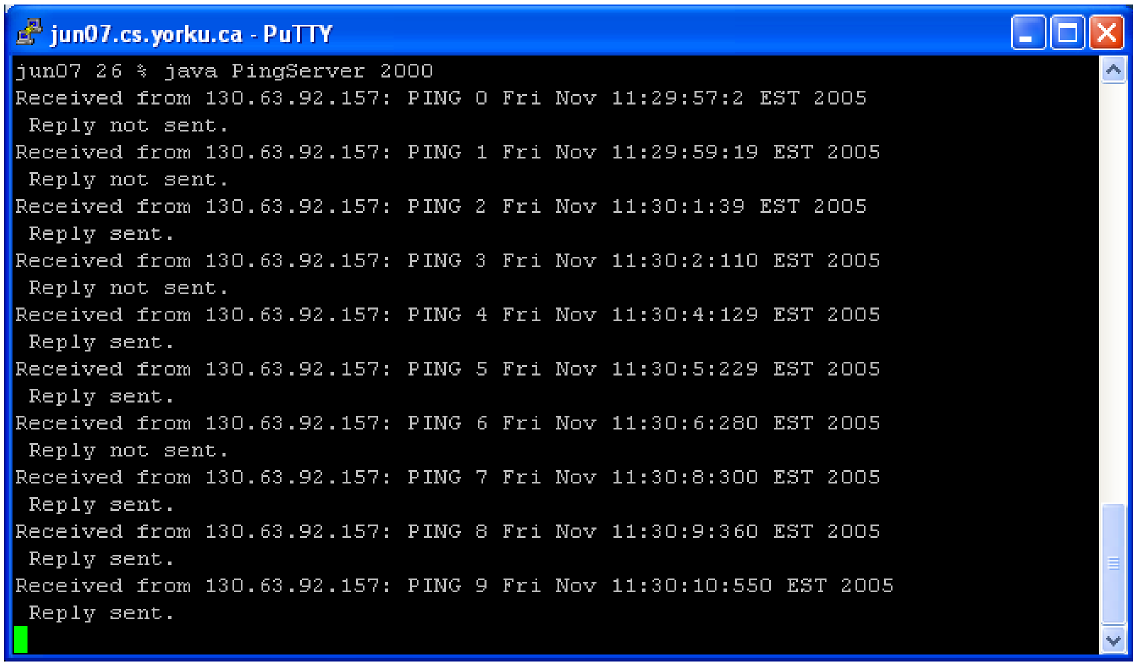
where ‘host’ is the name of the computer the server is running on, and ‘port’ is the port number it is listening to. Note – if you run the client and server on the same machine, then host = localhost, or 127.0.0.1.
The payload of your ping messages should be formatted as follows:

```
      PING sequence_number timestamp CRLF
```

sequence_number - starts at 0 and progresses to 9 for each successive ping message;
time – time when the client sent the message;
CRLF – carriage return and line feed character.

The client should prompt the user about the reception of each ping-response from the server, by printing out its payload.

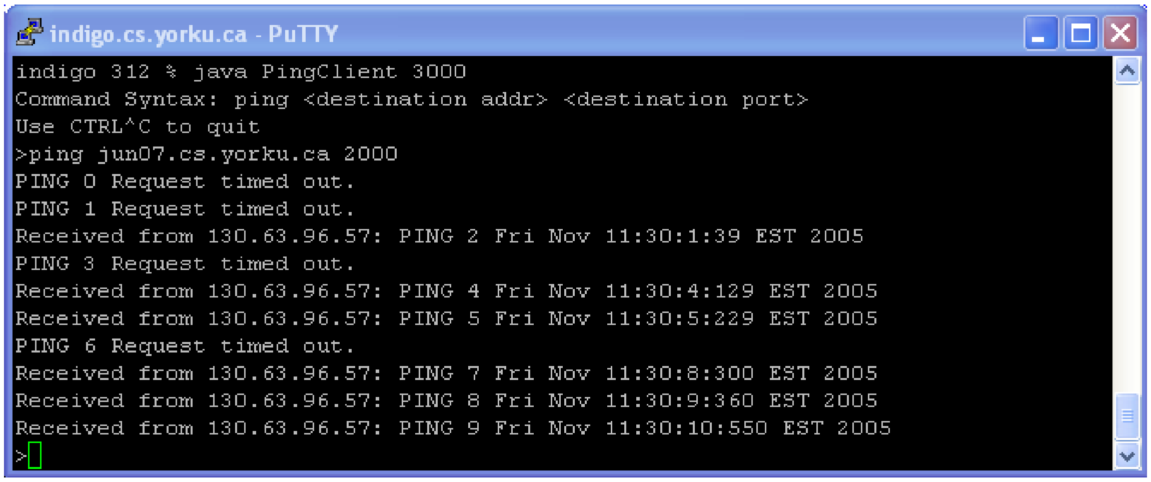
The snapshot of running PingServer and PingClient program are provided in the following pictures.



jun07.cs.yorku.ca - PuTTY

```
jun07 26 % java PingServer 2000
Received from 130.63.92.157: PING 0 Fri Nov 11:29:57:2 EST 2005
 Reply not sent.
Received from 130.63.92.157: PING 1 Fri Nov 11:29:59:19 EST 2005
 Reply not sent.
Received from 130.63.92.157: PING 2 Fri Nov 11:30:1:39 EST 2005
 Reply sent.
Received from 130.63.92.157: PING 3 Fri Nov 11:30:2:110 EST 2005
 Reply not sent.
Received from 130.63.92.157: PING 4 Fri Nov 11:30:4:129 EST 2005
 Reply sent.
Received from 130.63.92.157: PING 5 Fri Nov 11:30:5:229 EST 2005
 Reply sent.
Received from 130.63.92.157: PING 6 Fri Nov 11:30:6:280 EST 2005
 Reply not sent.
Received from 130.63.92.157: PING 7 Fri Nov 11:30:8:300 EST 2005
 Reply sent.
Received from 130.63.92.157: PING 8 Fri Nov 11:30:9:360 EST 2005
 Reply sent.
Received from 130.63.92.157: PING 9 Fri Nov 11:30:10:550 EST 2005
 Reply sent.
```

Figure 2.1 PingServer on: machine = jun07.cs.yorku.ca, port = 2000



indigo.cs.yorku.ca - PuTTY

```
indigo 312 % java PingClient 3000
Command Syntax: ping <destination addr> <destination port>
Use CTRL^C to quit
>ping jun07.cs.yorku.ca 2000
PING 0 Request timed out.
PING 1 Request timed out.
Received from 130.63.96.57: PING 2 Fri Nov 11:30:1:39 EST 2005
PING 3 Request timed out.
Received from 130.63.96.57: PING 4 Fri Nov 11:30:4:129 EST 2005
Received from 130.63.96.57: PING 5 Fri Nov 11:30:5:229 EST 2005
PING 6 Request timed out.
Received from 130.63.96.57: PING 7 Fri Nov 11:30:8:300 EST 2005
Received from 130.63.96.57: PING 8 Fri Nov 11:30:9:360 EST 2005
Received from 130.63.96.57: PING 9 Fri Nov 11:30:10:550 EST 2005
>
```

Figure 2.2 PingClient on: machine = indigo.cs.yorku.ca, port = 3000

2. ”Backdoor Sniffer” (TCP Socket Programming) [50 points]

In this question, you are asked to write a TCP application[1] which would, by means of a client program, allow you to obtain the following important information from a remote (server) machine: the current time on the server’s machine, the server’s time zone, the server’s OS name, the server’s OS version number, as well as the sever-administrator’s name.

In particular, the application should operate in the following manner:
Once the client connects to the server, the server should display the following menu on the client:

```
= = = = = = = = = = = = = = = = Menu = = = = = = = = = = = = = = = =
   date - print the date and time of server’s system
   timezone - print the time zone of server’s system
   OSname - print the name of server’s operating system (OS)
   OSversion - print the of version number of server’s OS
   user - print the name of the user logged onto (i.e. running) the server
   exit - exit the program
= = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
Enter command >
```

The client should then send a command to the server, and the sever should respond by sending the appropriate message back to the client. After typing “exit”, or any other command specified in your program, the client should close the talking session.

You should write the server so that it can be evoked with the following command:

```
     >   java NosyServer <port>
```

where port is the port number the server listens on.

The client should be evoked with the following command:

```
     >   java NosyClient <host> <port>
```

where ‘host’ is the name of the computer that NosyServer is running on, and ‘port’ is the port number of that server.

[1] You have to write both – the client and the server program.