CMSC 440 Programming Assignment
Simple PING Application

Due Date: Sunday December 4th, 2022 - 11:59pm

Description:

The goal of this assignment is to allow you to demonstrate your knowledge of socket programming for UDP connections. You will learn how to send and receive datagram packets using UDP sockets and, how to set a proper socket timeout. Throughout this assignment, you will gain familiarity with a Ping application and its usefulness in computing statistics such as packet loss rate. In this assignment, you will develop a simple Internet ping server written, and implement a corresponding client. The functionality provided by these programs is similar to the functionality provided by standard ping programs available in modern operating systems. However, these programs use a simpler protocol, UDP, rather than the standard Internet Control Message Protocol (ICMP) to communicate with each other. 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.

PINGClient

- Your client should be named PINGClient.java, PINGClient.py, PINGClient.c, etc.
- Your ping client should be able to send ping and receive responses from your ping server.
- Your client should accept a two command-line argument: the first one is either the hostname or the IP of your ping server, and the second is the port number your server is running at.
    - Example: PINGClient egr-v-cmsc440-2.rams.adp.vcu.edu 10500
    - Or: PINGClient 10.0.0.2 10500
    - If any of the arguments are incorrect, exit after printing an error message of the form “ERR - arg x”, where x is the argument number.
- Your ping client should send ping packets to the ping server given in the hostname/IP and the port thru a UDP connection.
    - Handle exceptions as you wish -- for example, if the host doesn't exist or the port given is not open
- The client should construct and send a valid PING packet consisting of PING header and PING payload.
- The PING header includes the following fields:
    - “**Version**” field of a byte length that includes the version number of your ping system. Let the value of this field to be “1” corresponding to Ver 1.0,
    - “**SequenceNo**” field of type integer. The value starts at 1 and progresses to 10 for each successive ping message sent by the client,
    - “**Timestamp**” field of type float. The client should set this value to the current time when this packet is constructed,

    - "**Size**" field of type integer that is set to the number of bytes (size) of the payload portion of the ping packet.
- The PING payload includes the following lines
    - "**Host: <hostname>**" where <hostname> is the hostname of the client,
    - "**Class-name: VCU-CMSC440-FALL-2022**",
    - "**User-name: <your last name>, <your first name>**".
- Print the ping request before you send it to the server. The ping header should be printed as:
    - ---------- Ping Packet Header ----------
    - Version: <Version field>
    - Sequence No.: <SequenceNo field>
    - Time: <Timestep field>
    - Payload Size: <Size field>
    - --------- Ping Packet Payload ----------
    - Host: <hostname>
    - Class-name: VCU-CMSC440-FALL-2022
    - User-name: <your last name>, <your first name>
    - ----------
- The client should send 10 pings to the server. The client should first try to receive a response from the server for the ping packet he just sent before sending the next packet. Because UDP is an unreliable protocol, a packet sent from the client to the server may be lost in the network, or vice versa. For this reason, the client cannot wait indefinitely for a reply to a ping message. You should get the client wait up to one second for a reply; if no reply is received within one second, your client program should assume that the packet was lost during transmission across the network. And transmit the next packet.
- When the client receives a reply from the server for the current ping packet the client sent, the client needs to parse the received reply packet and do the following:
    - Print the reply packet (header and payload) using the same format described earlier for transmitted ping packets with the exception of the divider lines that should be "---------- Received Ping Reply Header ----------" and "---------- Received Ping Reply Payload ----------".
    - Calculate and print the round trip time (RTT), in seconds. RTT is calculated by calculating the difference between the time the reply packet is received by the client, and the timestamp field in the packet, which was set when the packet was created.
- If the client times out and did not receive a reply for the packet that was just sent, the client needs to print: "---------- Ping Reply Timed out ----------"
- After the client is done with the transmitted 10 ping packets, it needs to print a summary of the ping process. In particular the client needs to print the minimum, maximum, and average RTTs at the end of all ping packets. In addition, your client needs to calculate and print the packet loss rate (in percentage).This should be print in the format:
    - Summary: <min RTT> :: <max RTT> :: <avg RTT> :: <packet loss rate>
- Once you have this part of the client working, you should test it with your server

PINGServer

- Here, you will develop your own version of a ping server.
- Your server should be named PINGServer.java, PINGServer.py, PINGServer.c, etc.
- The program must accept a single command-line argument: port, which is the port that it will listen on for incoming pings from clients.
    - If any of the arguments are incorrect, exit after printing an error message of the form "ERR - arg x", where x is the argument number.
    - The only error-checking that needs to be done on the port is to ensure it is a positive integer less than 65536.
    - Remember that only ports 10000-11000 are open for use.
- If the program is successful in creating the server socket using the input port number argument, your program should print this out in the form of: "PINGServer's socket is created using port number <port> with IP address <ip>…", where <port> is the input argument, and <ip> is the IP address of the server machine.
- If your program as unsuccessful in creating the socket using the input port number argument, it is because this port number is already being taken by another active socket.
    - In this case, the program should exit after printing an error message "ERR - cannot create PINGServer socket using port number <port>", where port is the input argument.
- If the socket is created successfully, the server should sit in an infinite loop listening for incoming UDP packets.
- When a packet arrives, the server simply capitalizes the encapsulated data in the received packet and then sends it back to the client.
- UDP provides applications with an unreliable transport service. Messages may get lost in the network due to router queue overflows, faulty hardware or some other reasons. Because packet loss is rare or even non-existent in typical campus networks, the server in this project injects artificial loss to simulate the effects of network packet loss. The server creates a variable randomized integer which determines whether a particular incoming packet is lost or not.
- More specifically, in this project, we assume that 30% of the client's packets will be lost. We will simulate this in the server code in which when a packet arrives to the server, the server will generate a random integer number in the range of [1, 10]. If the randomized integer is less than or equal to 3, the server ignores the received packet (simulating a packet drop). If the randomized integer is greater than or equal to 4, the server resumes normally and handle the received packet as describe earlier.
- For each packet arrives to the server,
    - You need to print the client's IP address, port, packet's sequence number, and whether the packet will be dropped or not in the format IP:port:Seq#:DROPPED in case if the packet will be ignored or IP:port:Seq#:RECEIVED
      Example: 172.18.233.83:63307:56:DROPPED
    - Print the header and the payload of the received ping packet in the following format:
        - ----------Received Ping Packet Header----------

        - Version: <Version field>
        - Sequence No.: <SequenceNo field>
        - Time: <Timestep field>
        - Payload Size: <Size field>
        - ---------Received Ping Packet Payload----------
        - Host: <hostname>
        - Class-name: VCU-CMSC440-FALL-2022
        - User-name: <your last name>, <your first name>
        - ----------
    - Construct a valid response including: 1) the ping response header fields (**Version**, **SequenceNo**, **Timestamp**, **Size**) that should copy the corresponding values in the received ping packet, and 2) the ping response payload that includes capitalization of each line of the payload of the received ping packet.
    - Print the reply packet (header and payload) using the same format described earlier for transmitted ping packets with the exception of the divider lines that should be "----------Ping Reply Header ----------" and "---------- Ping Reply Payload ----------".
    - The ping server program should remain running until the user closes it with Ctrl-C.
    - Once you have your server working, you could test with your PINGClient.

VM Linux Machines:

- Open "terminal" window either on Windows, Mac, or Unix machine.
- On terminal, type "ssh –l <eID> 172.18.233.74" then your VCU password to authenticate. Note that <eID> is your VCU user login. If success, you will be logged to machine "egr-v-cmsc440-1"
- You can also log to other machines "egr-v-cmsc440-2" and "egr-v-cmsc440-3". To do so:
    - Open a new terminal
    - Log first to machine "egr-v-cmsc440-1" as described above using the **ssh** command.
    - Then, once you logged to "egr-v-cmsc440-1", use "*ssh 10.0.0.2*" or "*ssh 10.0.0.3*" to log to "egr-v-cmsc440-2" or "egr-v-cmsc440-3" respectively.

Rules:

- ***The only programming networking classes allowed are the basic socket classes that we've used with the examples.*** For example, java.net.URL is not allowed and urllib2 in Python is not allowed.
- Your code should run on the VM Linux machines (e.g., 172.18.233.74). Note that only ports 10000-11000 are open for use
- You are not permitted to work with anyone else (even students not in the class) - all of the coding and documentation must be your own.
- Your program must compile (if Java/C++) and run on the VM Linux machines.
- You must write neat code and document it well. You will lose points for sloppy programs that contain little or no comments.

Hints:

- Look back in your notes to recall how UDP client-server network applications are structured.
- If you are using Java, note that readLine() in Java strips off newline characters before returning a String.
- If you are using Java, use the equals() method in Java to compare two Strings.

Testing:

A large part of your program's grade will be determined by how well it handles a set of inputs. You should test your program rigorously before submitting. Because your programs will be run and tested using a script, you must format your output exactly as I have described, or you will lose points.

Submission Materials:

- Make sure your program compiles and executes on Dept's VM Linux machines.
- Create a "Readme.txt" file for your program that lists how to compile and execute the program. Include your name and your V# as the first line in the Readme.txt.
- You must name your source programs PINGClient.java/PINGServer.java, PINGClient.py/PINGServer.py, or PINGClient.cpp/PINGServer.cpp.
- Submit all files necessary to compile your program.
- Zip all files of your program files in a single zip file and name it prog_assign.zip
- Submit through Canvas.

Submitting Assignments using Canvas

- Instructions from the official Canvas help pages

- https://community.canvaslms.com/t5/Student-Guide/How-do-I-submit-an-online-assignment/ta-p/503

- Step-by-step instructions:

1. Log in to the course Canvas page
2. Click the **Assignments** link on the left sidebar
3. Click the name of the intended assignment under the "**Programming Assignment**" group
4. To submit an assignment, click the **Start Assignment** button.
5. To upload a file from your computer as your assignment, select the **File Upload** tab.
6. You may upload as many files as needed.
7. When you've finished adding all files needed for the assignment, click **Submit Assignment**.

- After you have submitted your work, you will see information on the Sidebar about your submission with a link to your submission to download if necessary

Important: *If you submit your assignment and later realize you have made a mistake, you can resubmit another version of your assignment using the New Attempt button. The time of your last attempt will be counted as your submission date. You will only be able to view the details of your most recent submission on the Sidebar.*