Programming assignment 2 (UDP Ping)

Problem Statement

In this lab, you will learn the basics of socket programming for **UDP** in Python. You will learn how to send and receive datagram packets using UDP sockets and also, how to handle UDP packet **retransmissions**. Throughout the lab, you will gain familiarity with a Ping application and its usefulness in computing statistics such as packet loss rate.

You need to be familiar with the standard ping protocol **Internet Control Message Protocol** `ICMP` available in modern operating systems first. 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.

Then you are required to implement these functionalities using a simpler protocol, UDP, rather than the standard `ICMP` to communicate between a client and a server. You are given a skeleton code for the Ping server `pingserver.py` . Your task is to complete the Ping server code first and then design and implement your Ping client (`pingclient.py`).

Server Code

Server sample code

You are provided a skeleton code for the Ping server. You are required to complete the skeleton code `pingserver.py` first. The places where you need to fill in code are marked with `??????????` . Each place may require one or more lines of code. (**Note:** You may modify `pingserver.py` any way you like, as long as it meets the following requirements. You may even ignore this sample code and start all over from scratch.)

Functionalities of your server

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 lab 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.

In the sample code, 30% of the client's packets are simulated to be lost. You should study this sample code carefully, as it will help you write your Ping client. The server sits in an infinite loop listening for incoming UDP packets. When a packet comes in and if a randomized integer is greater than or equal to 4, the server simply **capitalizes the encapsulated data and sends it back to the client**; if the randomized integer is smaller than 3, the server needs to **inform to the client which packet has been dropped** and then **client needs to retransmit the dropped packet**.

Client Code

You need to develop your client code from scratch.

Ping Message Format

The ping messages in this lab are formatted in a simple way. The client message is one line, consisting of ASCII characters in the following format:

```
ping <ping_index> time_stamp
```

where `ping_index` starts at 1 and progresses to 100 for each successive ping message sent by the client, and `time_stamp` is the time when the client sends the message.

Functionalities of your client

You client program should support the following:

The client should send 100 pings to the server. Because UDP is an unreliable protocol, a packet sent from the client to the server may be lost in the network, or vice versa. Therefore, your client should **support retransmitting the dropped packets**. **(Note that you client is not allowed to use socket timeout)**

Specifically, your client program should

1. send the **ping message** using UDP to the Ping server ;
2. print the **ping response message** from the Ping server if the ping message has not been dropped ;

3. retransmit the same ping message if the previous ping message(s) have been dropped by the Ping server ; print which ping message needs to be retransmitted ;
4. calculate and print the round trip time (RTT) in seconds of each ping message/response pair. (Note that for the retransmitted ping request/response pairs, the starting time-stamp can be either the time-stamp of the **initial** ping request message or the **most recent** ping request message)

Run the code

1. Start `CSC361-VM` , open a terminal, and type `sudo mn -x` to generate a simple network topology ;
2. Flush all arp caches (more details can be found in the provided **Wireshark*Ethernet*ARP.pdf**) ;
3. Run your Ping server over one of the hosts created (e.g., h1)

   ```
   python pingserver.py <IP addr of Ping server> <Port number of Ping server>
   ```

4. Run your Ping client over one of the hosts created (e.g. h2)

   ```
   python pingclient.py <IP addr of Ping server> <Port number of Ping server>
   ```

Evaluation

You must demonstrate your working code inside the `CSC361-VM` using Wireshark. Using `CSC361-VM` , the server and the client must use different IP addresses and port numbers. Your Wireshark demo must capture all related packets, including `ARP` and `UDP` .

You are required to create **one single document** before your demo that has

- The screen shot of all your ARP packets and at least 20 UDP packets
- The screen shot of your client

You are **also** required to answer a few questions about your programming assignment during the demo in ECS 360.

The evaluation scheme is:

- (50%) Correctness of Python code
- (20%) The created document before your demo. You should clearly write up how to handle

the retransmissions at both server and client sides and how to calculate the RTT for the retransmitted ping request/response pairs

- (30%) Questions and Answers during the demo (at least 5 questions)

What to Hand in

You will hand in the **complete client/server code** along with the **single document created** to Connex before 19:00, 6th March 2020.