Carleton University

Department of Systems and Computer Engineering

SYSC 4502 ***Communications Software*** ***Winter 2021***

Lab Handout

Lab 1: UDP Pinger

Due: at the end of your scheduled lab/problem solving session
Late deadline: January 14 at noon

General Information

The labs/exercise sessions this term will practice network programming using Python and the socket API. Each lab will ask you to implement some limited amount of code and submit your solution on cuLearn. The TAs will mark the labs once they are submitted. They will also be available during the scheduled lab sessions to help answering any programming questions you may have:

- Lab Section 1: Mondays 11:30 am to 2:30 pm
- Lab Section 2: Tuesdays 2:30 pm to 5:30 pm

To contact the TAs, you can use the course ZOOM link posted on cuLearn. The labs themselves are due by the end of your scheduled lab/problem solving session. There is also a late deadline some time AFTER the second scheduled lab (posted on each lab handout). No late deadlines beyond that are allowed, if you fail to submit your lab work, you will not get any credit for it. Note that simply running into problems with the late deadline will not be sufficient reason to be granted any deferrals – the original deadline is earlier.

Python

You will need to have Python installed and running on a computer to do the labs. You can download the latest version (Python 3.9.1) from https://www.python.org/downloads/ for your Operating System. On Windows PCs (at least under Windows 10), you can also open a command prompt (type `cmd` in the search bar) and simply type in `python`. On my PC, that brought up the Microsoft Store with the option to GET the current version of Python.

There are a number of free resources that can serve as a refresher on programming in Python. I will go over the basics in class, but you probably will have to make extensive use of online tutorials. Some possible starting points are:

- Python for Beginners: https://www.python.org/about/gettingstarted/
- The Python Tutorial: https://docs.python.org/3/tutorial/
- Python Socket Programming HowTo: https://docs.python.org/3/howto/sockets.html
- Socket Programming in Python: https://realpython.com/python-sockets/

Lab Description

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 set a proper socket timeout.

You will first study a simple Internet ping server written in the Python, 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.

You are given the complete code for the Ping server (can be downloaded from the cuLearn page for the course, called UDP_Pinger_server.py). You can either open the code in a web browser and copy-and-paste the displayed source code into a text editor. Or, preferably, you right-click on the link and choose "Save link as" to download the file to your computer.

Your task is to write the Ping client. As a first step, download and run this code before running your client program. You do not need to modify this code. In the server code, 30% of the client's packets are simulated to be lost. You should study this code carefully, as it will help you write your ping client.

To run any Python program, open a command prompt (Windows terminology) or a terminal/interactive shell (Linux terminology). To run the server, once you entered the correct directory with your code in it, you simply need to type

```
python UDP_Pinger_server.py
```

In some (maybe most) Linux environments, you may need to use the command line:

```
python3 UDP_Pinger_server.py
```

On the machines I tested this, the `python` executable will execute a version of Python 2. All labs and sample code are based on Python 3, and there are differences. So it is important that you develop and test your code with the correct version of Python, as this is the version the TAs will use in testing your submission.

Depending on your OS and security settings, you may have to allow the program to start and to access the network (just accept all such requests as they pop up). No additional command line parameters are required. The code will open a UDP socket bound to the local IP address and port number 12000. The server, being stuck in an infinite loop, will have to be terminated manually by either closing the command prompt/terminal, or by hitting CTRL-C when the focus is on the command prompt/terminal.

Packet Loss

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 in range 0 to 9, which determines whether a particular incoming packet is lost or not. The server sits in an infinite loop listening for incoming UDP packets. When a packet comes in and if the randomized integer is greater than or equal to 3, the server capitalizes the encapsulated data and sends it back to the client.

Lab Handout

Client Code

You need to implement the matching client program. The client takes two command-line parameters, the IP address and port number that the server is listening on. So with the provided code, for example, you should invoke the client as follows (using a separate command prompt or terminal):

```
python UDP_Pinger_client.py 127.0.0.1 12000
```

The client should send 10 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. For this reason, the client cannot wait indefinitely for a reply to a ping message. You should get the client to 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. You will need to look up the Python documentation to find out how to set the timeout value on a datagram socket.

Specifically, your client program should:

(1) send the ping message using UDP (Note: Unlike TCP, you do not need to establish a connection first, since UDP is a connectionless protocol.)
(2) print the response message from the server, if any
(3) calculate and print the round trip time (RTT), in seconds, of each packet, if the server responds
(4) otherwise, print "Request timed out"

During development, you should run `UDP_Pinger_server.py` on your machine, and test your client by sending packets to localhost (or, 127.0.0.1). Since communication between client and server is now local (i.e., within the PC/OS) and not across a real network, measured RTTs will be 0 or a really small value. After you have fully debugged your code, you should see how your application communicates across the network with the ping server and ping client running on different machines. That will require you to determine the IP address used by the server device, using commands such `ipconfig` or `ifconfig`. Alternatively, you can get the server to print out the local IP address and port numbers.

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:

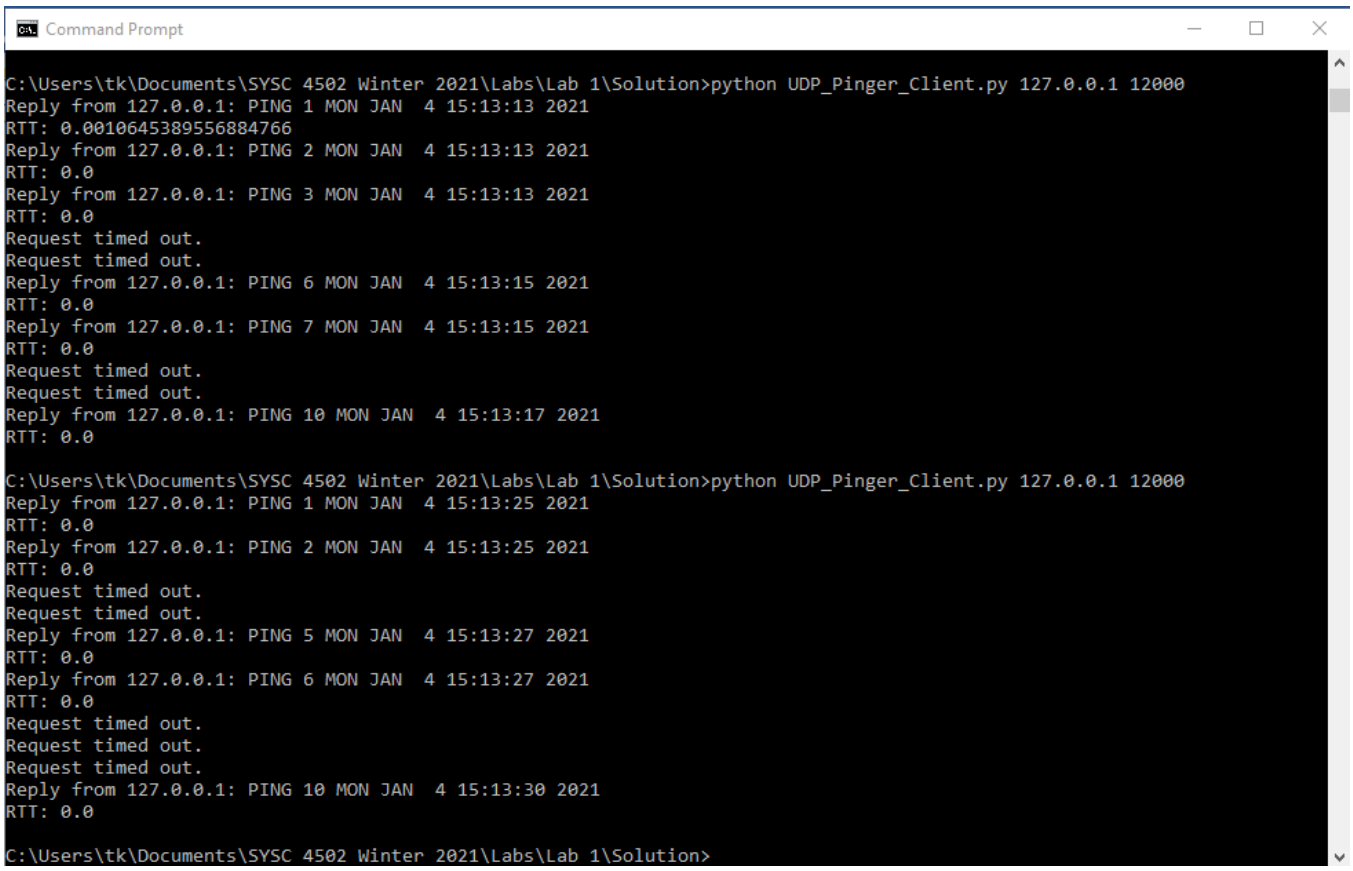
```
Ping sequence_number time
```

where sequence_number starts at 1 and progresses to 10 for each successive ping message sent by the client, and time is the time when the client sends the message.

What to Hand in

You will hand in the complete client code (as a file named `UDP_Pinger_client.py`) and a screenshot at the client verifying that your ping program works as required. A sample screenshot showing two executions of the client is included for your reference below.

Lab Handout


Command Prompt
C:\Users\tk\Documents\SYSC 4502 Winter 2021\Labs\Lab 1\Solution>python UDP_Pinger_Client.py 127.0.0.1 12000
Reply from 127.0.0.1: PING 1 MON JAN 4 15:13:13 2021
RTT: 0.0010645389556884766
Reply from 127.0.0.1: PING 2 MON JAN 4 15:13:13 2021
RTT: 0.0
Reply from 127.0.0.1: PING 3 MON JAN 4 15:13:13 2021
RTT: 0.0
Request timed out.
Request timed out.
Reply from 127.0.0.1: PING 6 MON JAN 4 15:13:15 2021
RTT: 0.0
Reply from 127.0.0.1: PING 7 MON JAN 4 15:13:15 2021
RTT: 0.0
Request timed out.
Request timed out.
Reply from 127.0.0.1: PING 10 MON JAN 4 15:13:17 2021
RTT: 0.0
C:\Users\tk\Documents\SYSC 4502 Winter 2021\Labs\Lab 1\Solution>python UDP_Pinger_Client.py 127.0.0.1 12000
Reply from 127.0.0.1: PING 1 MON JAN 4 15:13:25 2021
RTT: 0.0
Reply from 127.0.0.1: PING 2 MON JAN 4 15:13:25 2021
RTT: 0.0
Request timed out.
Request timed out.
Reply from 127.0.0.1: PING 5 MON JAN 4 15:13:27 2021
RTT: 0.0
Reply from 127.0.0.1: PING 6 MON JAN 4 15:13:27 2021
RTT: 0.0
Request timed out.
Request timed out.
Request timed out.
Reply from 127.0.0.1: PING 10 MON JAN 4 15:13:30 2021
RTT: 0.0
C:\Users\tk\Documents\SYSC 4502 Winter 2021\Labs\Lab 1\Solution>