# Lab 2: UDP Pinger Lab

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. Throughout the lab, you will gain familiarity with a Ping application and its usefulness in computing statistics such as packet loss rate.

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 below. Your task is to write the Ping client.

## Server Code

The following code fully implements a ping server. You need to compile and run this code before running your client program. *You do not need to modify this code.*

In this 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.

# UDPPingerServer.py

# We will need the following module to generate randomized lost packets

import random

from socket import \*

# Create a UDP socket

# Notice the use of SOCK\_DGRAM for UDP packets

serverSocket = socket(AF\_INET, SOCK\_DGRAM)

# Assign IP address and port number to socket

serverSocket.bind(('', 12000))

while True:

# Generate random number in the range of 0 to 10

rand = random.randint(0, 10)

# Receive the client packet along with the address it is coming from

message, address = serverSocket.recvfrom(1024)

# Capitalize the message from the client

message = message.upper()

# If rand is less is than 4, we consider the packet lost and do not respond

if rand < 4:

continue

# Otherwise, the server responds

serverSocket.sendto(message, address)

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.

## 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 which determines whether a particular incoming packet is lost or not.

## Client Code

You need to implement the following client program.

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 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 server, if any

(3) calculate and print the round trip time (RTT), in seconds, of each packet, if server responses

(4) otherwise, print “Request timed out”

During development, you should run the UDPPingerServer.py on your machine, and test your client by sending packets to *localhost* (or, 127.0.0.1). 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.

## 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 and screenshots at the client verifying that your ping program works as required.

## Optional Exercises

1. Currently, the program calculates the round-trip time for each packet and prints it out individually. Modify this to correspond to the way the standard ping program works. You will need to report the minimum, maximum, and average RTTs at the end of all pings from the client. In addition, calculate the packet loss rate (in percentage).
2. Another similar application to the UDP Ping would be the UDP Heartbeat. The Heartbeat can be used to check if an application is up and running and to report one-way packet loss. The client sends a sequence number and current timestamp in the UDP packet to the server, which is listening for the Heartbeat (i.e., the UDP packets) of the client. Upon receiving the packets, the server calculates the time difference and reports any lost packets. If the Heartbeat packets are missing for some specified period of time, we can assume that the client application has stopped.

Implement the UDP Heartbeat (both client and server). You will need to modify the given UDPPingerServer.py, and your UDP ping client.

|  |
| --- |
| UDPPingerClient.py 소스코드 |
| **import** time **from** socket **import** \*  serverAddr = (**'127.0.0.1'**, 12000) timemax = 1 **for** i **in** range(10):   **with** socket(AF\_INET, SOCK\_DGRAM) **as** clientSocket:  clientSocket.settimeout(timemax)   msg = (**'Ping'**).encode()  start = time.time()  clientSocket.sendto(msg, serverAddr)   **try**:  msg, address = clientSocket.recvfrom(1024)  end = time.time()  rtt = end-start  print(msg.decode() + **' '** +str(i+1) + **' '** + str(rtt)+**'s'**)  **except** timeout:  print(**'Ping'**+ **' '** + str(i + 1) + **' Request timed out'**) |
| serverAddr로 서버의 주소와 포트를 지정했다. 주소는 127.0.0.1, 포트는 12000으로 설정했다. 그리고 최대 회신 대기 시간을 1초로 하기 위해 timemax변수를 만들었다. 10개의 PING을 생성하기 위해 for문을 이용해 10번의 메시지 발신과 수신을 반복되로록 하였다. 다음으로 clientSocket을 생성하고 settimeout을 이용해 clientSocket의 최대 회신 대기 시간을 1로 만들었다. 그리고 UDPPingerServer로 보낼 메시지를 msg = Ping으로 만들었다. 그리고 메시지를 보내는 시간을 측정하기 위해 start 변수로 만들고, clientSocket.sendto(msg, serverAddr)를 통해 serverAddr(127.0.0.1, 12000)로 메시지(msg)를 보낸다.  try구문 안에서 clientSocket.recvfrom(1024) 함수를 통해 UDPPingerServer로부터 회신을 받는다. 그리고 회신을 받고 난 후의 시간을 측정한다. 메시지송신시간-메시지발신시간을 하여 rtt시간을 구한다. 그 후, print 문을 이용하여 msg내용과 PING번호, rtt 시간을 출력한다. 그리고 만약 최대 회신 대기 시간인 1초가 지나게 된다면 except timeout: 구문에서 Ping번호와 Request timed out을 출력하게 된다. |
| 결과화면 |
| UDPPingerServer 결과화면    UDPPingerClient 결과화면 |
| 결과 화면은 다음과 같다. 먼저 UDPPingerServer 에서 UDPPingerClient의 메시지를 기다린다. 그리고 UDPPingerClient에서 UDPPingerServer로 메시지를 보낸다. 출력 결과는 ping sequence\_number time과 같고 결과 화면의 위와 같다.  UDPPingerClient에서 보낸 연속적인 10개의 PING에 대한 결과다. 결과를 살펴보면 PING 4와 PING 8에서 총 2번의 Request timed out이 발생했다. Request timed out이 발생한 이유는 UDPPingerServer에서 패킷 손실이 발생했기 때문이다. 위의 UDPPingerServer 결과 화면에 패킷이 손실이 발생했을 때 dropped packet이 출력된 것을 확인할 수 있다. 즉, UDPPingerClinet에서 UDPPingerServer로 메시지를 보냈을 때, UDPPingerServer의 랜덤함수에서 생성된 값이 4보다 작은 값이 생성되었고, dropped packet 메시지를 출력한다. 그리고 이 때, 우리가 가정한 패킷 손실이 발생했다고 보고 continue문으로 인해 serverSocket.sendto(message, address)가 실행되지 않기 때문에 UDPPingerClient에서 회신 대기시간이 1초가 지나게되고 Request timed out이 출력된다.  PING 1~10중에서 PING 4와 PING 8을 제외한 PING에서는 메시지 Ping에 대해 UDPPingerServer에서 message.upper() 함수가 실행되어 PING으로 메시지가 바뀌게 되고 RTT 시간이 출력된다. |