

Network Programming

Labs 3- Ping UDP

Labs individual
Write a pdf document and transmit your code

Exercise 1 - UDP

Overview

the basics of socket programming for UDP . 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 in c . The functionality provided by these programs is similar to the functionality provided by standard ping programs available in 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.

Task 1.1 : You need to compile and run this Server codecode python

The following code 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.

```
# UDPPingServer.py
# We will need the following module to generate randomizedlost packets

import random
from socket import *
import time

# What's your IP address and witch port should we use?
```

```

recieve_host = '127.0.0.1'
recieve_port = 18000

# What's the remote host's IP address and witch port should we use?
remote_host = '127.0.0.1'
remote_port = 1024

# 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((recieve_host, recieve_port))

simulate_packet_loss = True
sleep_for_rand_response_times = True

def get_time():
    return int(round(time.time() * 1000))

def send_message(message, wait=False):
    serverSocket.sendto(message, (remote_host, remote_port))
    if wait == False:
        return
    else:
        return wait_for_response()

# Just respond to ping requests
while True:
    # Receive the client packet along with the address it is coming from
    message, address = serverSocket.recvfrom(remote_port)
    # Capitalize the message from the client
    message = message.upper()
    print 'Recieve: ' + message
    # If rand is less is than 4, we consider the packet lost and do not respond
    if sleep_for_rand_response_times:
        min_sleep = 0.2
        max_sleep = 1.0
        time.sleep(random.uniform(min_sleep, max_sleep))
    if simulate_packet_loss:
        if random.randint(0, 10) < 2:
            print 'Dropped, lol'

```

```

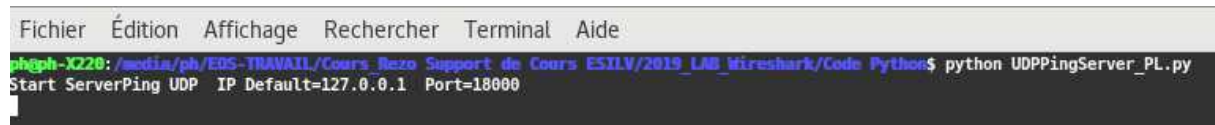
        continue
    elif simulate_packet_loss:
        if random.randint(0, 10) < 4:
            print 'Dropped, lol'
            continue
        serverSocket.sendto(message, address)
        print 'Send: ' + message

```

For more information : The server runs 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 ESILV 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.



```

Fichier Édition Affichage Rechercher Terminal Aide
phgph-X220: /media/ph/EDS-TRAVAIL/Cours_Rezo_Support_de_Cours_ESILV/2019_LAB_Wireshark/Code_Python$ python UDPPingServer_PL.py
Start ServerPing UDP IP Default=127.0.0.1 Port=18000

```

Exercise 2 - Creating a socket

Task 2.1: Implemented the ping client in c

Client Code

You need to implement the following client program, based on the starter code provided. The client should send **n 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 2019S1_C5_DOC_BGNET_Socket Programming_EN documentation to find out how to set the timeout value on a datagram socket.

Specifically, your client program should:

- send the ping message using UDP (Note: Unlike TCP, you do not need to establish a connection first, since UDP is a connectionless protocol.)
- Validate and debug your work with WireShark
- print the response message from server, if any

- d) calculate and print the round trip time (RTT), in seconds, of each packet, if server responses
- e) otherwise, print “Request timed out”
- f) Have your client 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). See below for how the full output should look in your terminal.
- g) Print UDP packet sequence number

Note

During development, you should run the ping server 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.

For more information :

Documentation : 2019S1_C5_DOC_BGNET_Socket Programming_EN

Message Format

The ping messages from the 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.

When your server is running and you execute the client, you should see (something like) the following in the server terminal:

```

Fichier  Édition  Affichage  Rechercher  Terminal  Aide
ph@ph-X220: /media/ph/EDS-TRAVAIL/Cours Rezo Support de Cours ESILV/2019
LAB Wireshark/Code Python$ python UDPPingServer_PL.py
Start ServerPing UDP IP Default=127.0.0.1 Port=18000
Ping ok =>9
Detected packet loss 1
Ping ok =>5
Ping ok =>9
Ping ok =>8
Detected packet loss 3
Detected packet loss 4
Detected packet loss 4
Ping ok =>8
Detected packet loss 1
Ping ok =>5
Ping ok =>9
Detected packet loss 3
Ping ok =>9
Detected packet loss 3
Detected packet loss 4
Detected packet loss 2
Ping ok =>5
Detected packet loss 3

```

Your client terminal window should display something like the following:

```

Fichier  Édition  Affichage  Rechercher  Terminal  Aide
ph@ph-X220: /media/ph/EDS-TRAVAIL/Cours Rezo Support de Cours ESILV/2019
LAB Wireshark/Code Python$ python UDPPingServer PL.py
Start ServerPing UDP IP Default=127.0.0.1 Port=18000
Ping ok =>9
Detected packet loss 1
Ping ok =>5
Ping ok =>9
Ping ok =>8
Detected packet loss 3
Detected packet loss 4
Detected packet loss 4
Ping ok =>8
Detected packet loss 1
Ping ok =>5
Ping ok =>9
Detected packet loss 3
Ping ok =>9
Detected packet loss 3
Detected packet loss 4
Detected packet loss 2
Ping ok =>5
Detected packet loss 3

```