Norges teknisk-naturvitenskapelige universitet

Institutt for informasjonssikkerhet og kommunikasjonsteknologi

TTM4100 – Kommunikasjon – Tjenester og nett

Praksisøving 4

UDP Pinger

Denne praksisøvingen er én av åtte praksisøvinger i emnet. Du må levere og få godkjent minst seks av disse praksisøvingene, og tilsvarende for teoriøvingene, for å kunne gå opp til eksamen. Hvis du lurer på noe angående øvingen kan du stille spørsmål på Piazza eller få veiledning av læringsassistent tirsdager i F1 kl. 16:15–19:00.

Oppgavene er ment å fokusere på læring og forståelse av pensum, og ikke debugging av Python-kode. Nøl derfor ikke med å spørre i øvingstime eller på Piazza om du står fast på noe med koden! Du kan velge om du vil svare på norsk eller engelsk.

Last opp svarfilene dine på Blackboard innen fristen. Filene som skal lastes opp er en Python-fil med klientkode og screenshots av den fungerende klienten din, som beskrevet i oppgaveteksten.

Innleveringsfrist: **søndag 8. mars 2020, kl. 23:59.**

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 Python, and then implement the 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.

N.B. It is strongly recommended to read **Sect. 2.7.1** in the textbook before (and while) doing this programming lab.

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.

The code can also be found in the file "UDPPingServer.py".

```
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))
print ("Ready to receive Pings!!");
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.decode().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.encode(), address)
	print(message)
```

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. Since 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 randomized integer variable which determines whether a particular incoming packet is lost or not.

Client Code

You are to complete the skeleton code in the file "Skeleton UDPPingerClient.py" using Python 3. The places where you need to fill in code are marked with `#FILL IN START` and `#FILL IN END`. Each place may require one or more lines of code.

The client should send 10 pings to the server. Because UDP is an unreliable protocol, a packet sent from the client to the server, or vice versa, may be lost in the network. 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.

N.B. You will need to import the module *time* in order to use time-related functions like *time()* and/or *asctime()*. Some return time as a number, while others return the time as a string or other formats. Check the Python documentation and use the one(s) that suit you better.

Message Format

The ping messages in this lab are formatted in a simple way. The client message is one line, consisting of ASCII characters (becoming a byte array of UTF-8 characters after calling `encode()`) 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.