CSE 232 Section B, Computer Networks; Programming Assignment 1: UDP Pinger Lab

Harsh Rajput (2022201)
Aryan Singla (2022112)

Part 1: **UDP Ping Client System**

This system simulates the functionality of the common network utility tool *Ping* using the User Datagram Protocol (UDP). The primary purpose is to send a series of "ping" messages to a server, measure the round-trip time (RTT) for each response, and calculate statistics such as minimum RTT, maximum RTT, average RTT, and packet loss rate. The client sends 10 ping requests and waits for a response from the server.

Server code

UDPPingerServer.py

```python
# We will need the following module to generate randomized lost packets
import time
import random
from socket import *

serverName = "localhost"
serverPort = 12000
serverAddress = (serverName, serverPort)
# 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(('', serverPort))
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, clientAddress = serverSocket.recvfrom(1024)
    # Capitalize the message from the client
    modifiedMessage = message.decode('utf-8').upper().encode('utf-8')
    # 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(modifiedMessage, clientAddress)
```

Client code

UDPPingerClient.py

```python
import time
from socket import *

serverName = "localhost"
serverPort = 12000
serverAddress = (serverName, serverPort)

# Define the timeout value for the client to wait for a response from the server
TIME_OUT = -1
serverResponses = []

# Create a UDP clientSocket
clientSocket = socket(AF_INET, SOCK_DGRAM) # Notice the use of SOCK_DGRAM for UDP packets
# Set the timeout of to 1 second
clientSocket.settimeout(1)
print(f"PING {serverName} 10 times")
total_ping = 10
for sequence_number in range(1,total_ping+1):
    time_sent = time.time()
    message = f"Ping {sequence_number} {time.ctime(time_sent)}".encode('utf-8')
    # Send the UDP packet with the ping message
    clientSocket.sendto(message,serverAddress)
    # Wait for the server to respond
    try:
        modifiedMessage, serverAddress = clientSocket.recvfrom(1024)
        modifiedMessage = modifiedMessage.decode('utf-8')
        time_received = time.time()
        rtt = (time_received - time_sent)*1000
        print(f"udp_seq={sequence_number} message=\"{modifiedMessage}\" rtt={rtt:.3f}ms")
        serverResponses.append(rtt)
    except timeout:
        print(f"udp_seq={sequence_number} Request timed out ")
        serverResponses.append(TIME_OUT)

clientSocket.close()
```

Code to calculate metrics

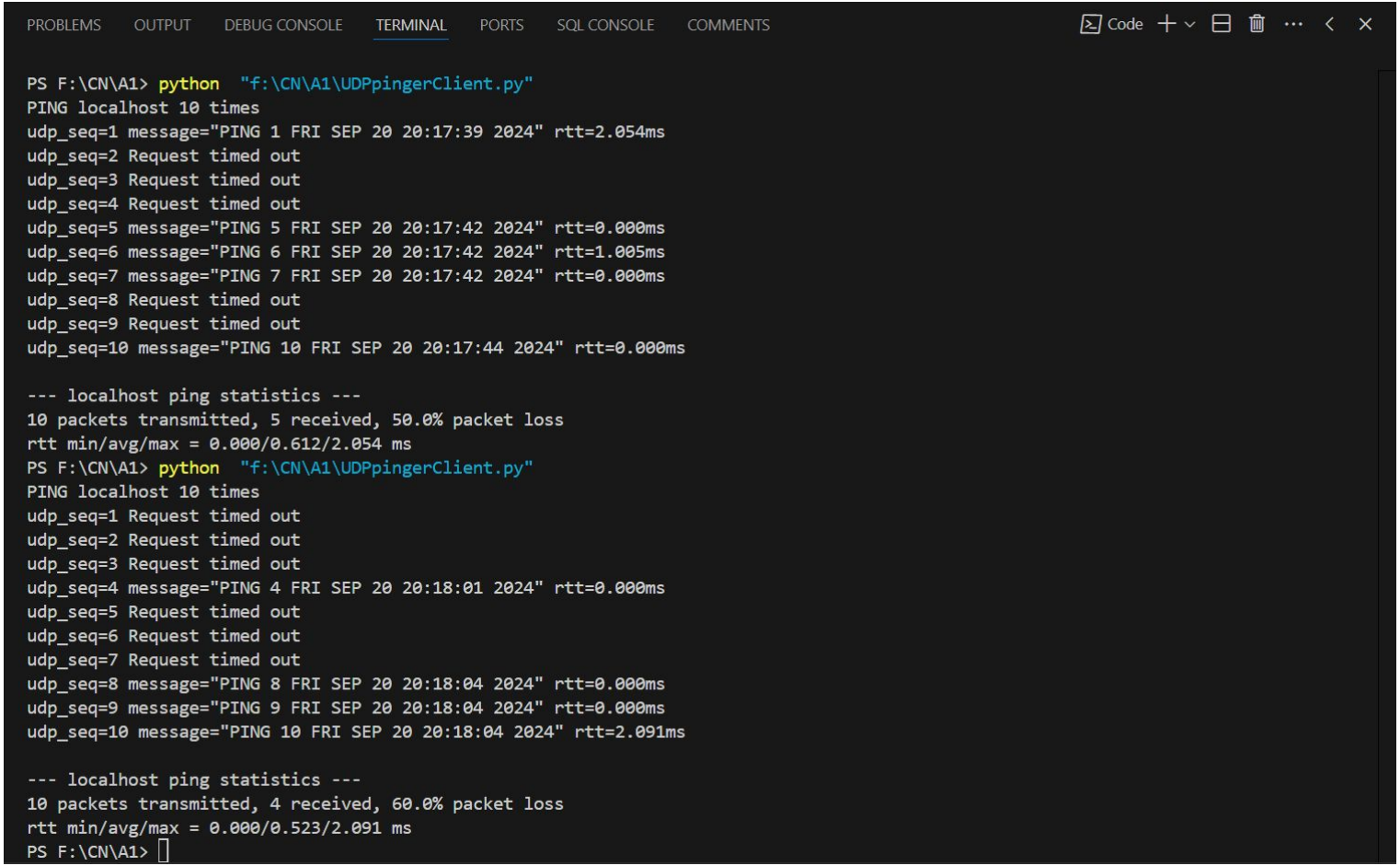
UDPPingerClient.py

```python
# Calculate the statistics
# Report the minimum RTT, maximum RTT, average RTT, and packet loss rate at the end of all pings from the client
total_time = 0
packet_loss = 0
rtt_min = float('inf')
rtt_max = float('-inf')
for rtt in serverResponses:
    if rtt == TIME_OUT:
        packet_loss += 1
    else:
        total_time += rtt
        rtt_min = min(rtt_min,rtt)
        rtt_max = max(rtt_max,rtt)

packet_transmitted = len(serverResponses)
packet_received = packet_transmitted - packet_loss
packet_loss_rate = (packet_loss/packet_transmitted)*100
if packet_received == 0:
    rtt_avg = 0
else:
    rtt_avg = total_time/packet_received

print(f"\n--- {serverName} ping statistics ---")
print(f"{packet_transmitted} packets transmitted, {packet_received} received, {packet_loss_rate:.2f}% packet loss")
print(f"rtt min/avg/max = {rtt_min:.3f}/{rtt_avg:.3f}/{rtt_max:.3f} ms")
```

Client end output



Part 1: **UDP Heartbeat Client and Server System**

Another similar application to the UDP Ping would be the UDP Heartbeat. The Heartbeat can be used to check if an application server is up and running or not. 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) from the client. Upon receiving the packets, the server calculates the time difference and reports it back. If the server response to client Heartbeat packets is missing for some specified number of times (say 3), the client can assume that the server application has stopped. Here again simulate a UDP packet loss at rate 30%.

Server code

UDPHeartbeatServer.py

```python
# We will need the following module to generate randomized lost packets
import time
import random
from socket import *

serverName = "localhost"
serverPort = 12000
serverAddress = (serverName, serverPort)
# 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(('', serverPort))
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, clientAddress = serverSocket.recvfrom(1024)
    # time received
    time_received = time.time()
    # decode the message and get the sequence number and time sent
    message = message.decode('utf-8').split()
    sequence_number = int(message[1])
    # convert the time sent form ctime to float
    time_sent = float(message[2])
    # delay in ms
    time_difference = (time_received - time_sent)*1000
    modifiedMessage = f"Ping {sequence_number} time difference={time_difference:.3f}ms".encode('utf-8')
    # 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(modifiedMessage, clientAddress)
```

Client code

UDPHeartbeatClient.py

```python
import time
from socket import *

serverName = "localhost"
serverPort = 12000
serverAddress = (serverName, serverPort)

# Define the timeout value for the client to wait for a response from the server
TIME_OUT = -1
serverResponses = []

# Create a UDP clientSocket
clientSocket = socket(AF_INET, SOCK_DGRAM) # Notice the use of SOCK_DGRAM for UDP packets
# Set the timeout of to 1 second
clientSocket.settimeout(1)
Heartbeat = 3
print(f"PING {serverName} 10 times")
total_ping = 10
for sequence_number in range(1,total_ping+1):
    time_sent = time.time()
    message = f"Ping {sequence_number} {time_sent}".encode('utf-8')
    # Send the UDP packet with the ping message
    clientSocket.sendto(message,serverAddress)
    # Wait for the server to respond
    try:
        modifiedMessage, serverAddress = clientSocket.recvfrom(1024)
        modifiedMessage = modifiedMessage.decode('utf-8')
        time_received = time.time()
        rtt = (time_received - time_sent)*1000
        print(f"udp_seq={sequence_number} message=\"{modifiedMessage}\" rtt={rtt:.3f}ms")
        serverResponses.append(rtt)
        # reset the heartbeat
        Heartbeat = 3
    except timeout:
        print(f"udp_seq={sequence_number} Request timed out ")
        serverResponses.append(TIME_OUT)
        # decrement the heartbeat and check if the server applica0on has stopped
        Heartbeat -= 1
        if (Heartbeat == 0):
            print("Server is down")
            break

clientSocket.close()
```

Client end Output