**Computer Networking Socket Programming Lab 2: UDP Pinger Lab**

**\*\*Identifying Lab Environment\*\***

The below screenshot depicts my lab environment used for the lab work. I am using MacOS with Python version 3.9.11.

A computer screen shot of a computer screen

Description automatically generated

1. UDP Pinger Lab: Client Code Completed with commented lines for explanation of code

client.py

from socket import \*

import time

import sys

def ping(host, port):

resps = []

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

clientSocket.settimeout(1)

for seq in range(1, 11):

startTime = time.time() # Retrieve the current time

message = "Ping {} {}".format(seq, startTime)

try:

# Sending the message and waiting for the answer

clientSocket.sendto(message.encode(), (host, port))

encodedModified, serverAddress = clientSocket.recvfrom(1024)

# Checking the current time and if the server answered

endTime = time.time()

modifiedMessage = encodedModified.decode()

rtt = (endTime - startTime) \* 1000 # Convert to milliseconds

resps.append((seq, modifiedMessage, rtt))

except timeout:

resps.append((seq, 'Request timed out', 0))

clientSocket.close()

return resps

if \_\_name\_\_ == '\_\_main\_\_':

resps = ping('127.0.0.1', 12000)

print(resps)

Output:

Running the UDPPingerServer.py

A screen shot of a computer

Description automatically generated

Running the client.py

A screenshot of a computer

Description automatically generated

1. Optional Exercise: Enhanced UDPPingerClient with RTT Statistics and Packet Loss Rate

EnhancedPingClient.py

from socket import \*

from time import time, ctime

import sys

def ping(host, port):

rtts = [] # to store the RTTs

packet\_loss\_count = 0 # packet lost counter

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

clientSocket.settimeout(1)

for seq in range(1, 11):

startTime = time()

message = f"Ping {seq} {ctime(startTime)[11:19]}"

try:

clientSocket.sendto(message.encode(), (host, port))

encodedModified, serverAddress = clientSocket.recvfrom(1024)

endTime = time()

server\_reply = encodedModified.decode()

rtt = (endTime - startTime) \* 1000 # RTT in ms

rtts.append(rtt) # storing the RTT in list for getting stat report on this

print(f"Reply from {serverAddress}: {server\_reply}")

print(f"RTT: {rtt:.3f} ms\n")

except timeout:

print(f"PING {seq} Request timed out\n")

packet\_loss\_count += 1

# Closing client

clientSocket.close()

# RTT stats

if rtts:

print("UDP Pinger Report:")

print(f"Maximum RTT: {max(rtts):.3f} ms") # max value from the list

print(f"Minimum RTT: {min(rtts):.3f} ms") # min value from the list

print(f"Average RTT: {sum(rtts) / len(rtts):.3f} ms") # avg calculation from the list

else:

print("No successful pings, all requests timed out.")

# Calculating Packet Loss Rate

total\_packets = 10 # since 10 pings were made from the client skeleton code earlier

packet\_loss\_rate = (packet\_loss\_count / total\_packets) \* 100

print(f"Packet loss rate: {packet\_loss\_rate:.2f}%")

if \_\_name\_\_ == '\_\_main\_\_':

ping('127.0.0.1', 12000)

Output:

Running EnhancedPingClient.py

A screenshot of a computer

Description automatically generated

1. Optional Exercise: UDP Heartbeat

HeartbeatServer.py

from socket import \*

import time

import sys

def serve(port, heartbeat\_timeout=5):

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

serverSocket.bind(('', port))

last\_seq = -1 # to keep track of sequence number for detecting lost packets

last\_heartbeat\_time = time.time() # the last heartbeat was received time

print("Server is ready to receive heartbeat signals.")

while True:

try:

message, address = serverSocket.recvfrom(1024)

current\_time = time.time()

heartbeat = message.decode().split() # heartbeat message decode

seq = int(heartbeat[1])

c\_time = float(heartbeat[2])

print(f"Received Heartbeat {seq} from {address} at {current\_time}, sent at {c\_time}")

if seq != last\_seq + 1 and last\_seq != -1: # checking packet loss

print(f"Warning: Packet loss detected! Expected seq {last\_seq + 1}, but got {seq}.")

if current\_time - last\_heartbeat\_time > heartbeat\_timeout: # checking heartbeat timeout

print("Warning: Heartbeat timeout. Client may have stopped.")

# Updating last sequence number and heartbeat time

last\_seq = seq

last\_heartbeat\_time = current\_time

except KeyboardInterrupt:

serverSocket.close()

sys.exit()

if \_\_name\_\_ == '\_\_main\_\_':

serve(12000)

HeartbeatClient.py

from socket import \*

import time

import sys

def heartbeat(host, port, interval=2):

seq = 0 # Sequence number for heartbeat messages

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

while True:

try:

current\_time = time.time() # Current time

message = f"Heartbeat {seq} {current\_time}"

# Sending heartbeat to server

clientSocket.sendto(message.encode(), (host, port))

print(f"Sent Heartbeat {seq} at {current\_time}")

seq += 1

# Wait for the specified interval before sending the next heartbeat

time.sleep(interval)

except KeyboardInterrupt:

print("Stopping the heartbeat client.")

clientSocket.close()

sys.exit()

if \_\_name\_\_ == '\_\_main\_\_':

heartbeat('127.0.0.1', 12000)

Output:

Running HeartbeatServer.py

A screen shot of a computer

Description automatically generated

Running HeartbeatClient.py

A screen shot of a computer

Description automatically generated