

Programming Assignment_1: My Pingers

UDP:

The ping tool simulated using UDP in the following code and the client side of the code is the task at hand here. However, the additional tasks here are to calculate the RTT statistics like the maximum ping time, minimum and the average ping time, alongside the packet loss percentage. The step flow is as follows:

Server:

1. UDP Socket:
 - We are creating UDP socket by AF_INET which is the IPv4. Bind() function used to bind the socket to all available networks and the port number 11000.
2. While loop for receive and response:
 - Recvfrom() function used to receive message from the udp client up to 1024 bytes.
 - Message.upper() converts the message to uppercase.
3. Packet loss:
 - Here, we generate random number between 1 and 10 using the random.randint(1,10) for simulate the packet loss. We put a condition if rand>8 means if the random number is greater than 8 then server simulates the packet loss and print simulating packet loss.
4. Response:
 - If packets are not lost server responds to client and print responding to client.

Client:

1. Variables initialisation and Socket creation:
 - It initializes variables to track the minimum RTT (min_rtt), maximum RTT (max_rtt), total RTT, and the number of lost packets. Now a UDP socket is created to communicate with the server, along with a 1-second timeout is set to avoid waiting indefinitely for a response.
2. Ping Loop: For each ping attempt (1 to N):
 - Sends a message containing time stamp, sequence number and also notes the sent time for RTT calculation. If no response is confirmed in a second time, There is a packet loss detected as the timeout is triggered.
3. Close Socket and Stats calculation:
 - Close the socket after all the pings are apparently received.
 - If the response is received, then it begins calculating the stats of RTT as discussed. And it only calculates the average RTT for the successful packets whilst keeping track of lost packet numbers so as to calculate the packet loss percentage.

The client code is as shown below:

```
1 import socket      #importing socket for network communication
2 import time       #importing time module to measure RTT time
3 from math import inf   #importing math module for mathematics function
4
5 # Input from the user
6 server_address = input("Enter server IP address: ")    #write the ip address of server
7 server_port = int(input("Enter server port: "))        #write the server port number
8 N = int(input("Enter number of pings: "))            #enter the number of pings
9
10
11 min_rtt=inf      #set min_rtt to positive infinity to find min rtt
12 max_rtt=-inf     #set max_rtt to positive infinity to find max rtt
13 total_rtt=0       #initialize total_rtt to 0 to calculate the avg rtt
14 lost_packets=0    #initialize lost packet to 0
15
16
17 client_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)    #creating udp server
18 client_socket.settimeout(1.0)                                         # Set timeout of 1 second for waiting for the response
19
20 for sequence_number in range(1, N + 1):
21     message = f"Ping {sequence_number} {time.time()}"                  # Preparing the message with sequence number
22
23     try:
24         start_time = time.time()                                       # Record the time before sending the message
25
26         client_socket.sendto(message.encode(), (server_address, server_port))  # Sending the ping message to the server
27
28         response, addr = client_socket.recvfrom(1024)                 # receive data (upto 1024 bytes) from server
29
30         end_time = time.time()
31         rtt = (end_time - start_time) * 1000
32         min_rtt = min(min_rtt, rtt)
33         max_rtt = max(max_rtt, rtt)
34         total_rtt += rtt
35
36
37
38         print(f"Response from server: {response.decode()}")           #printing server response
39         print(f"Sequence number: {sequence_number}, RTT: {rtt:.3f} ms") #printing rtt
40
41     except socket.timeout:
42         print(f"Request timed out for the packet #{sequence_number}") # Handle packet loss where server does not respond
43         lost_packets+= 1
44
45
46     client_socket.close()      # Close the socket after the loop
47
48
49     successful_pings = N - lost_packets                         #calculating number of successfull pings
50     packet_loss_percent = (lost_packets/N) * 100                #calculating packet loss percentage
51     if successful_pings > 0:
52         avg_rtt = total_rtt/successful_pings                     #calculating avg rtt for successful pings
53
54     else:
55         avg_rtt = 0
56
57     if successful_pings>0:                                     #print rtt for successful pings
58         print(f"Min RTT: {min_rtt:.2f} ms")
59         print(f"Max RTT: {max_rtt:.2f} ms")
60         print(f"Average RTT: {avg_rtt:.2f} ms")
61
62     print(f"Packet Loss Rate: {packet_loss_percent:.2f}%)"       #print the packet loss rate
```

The Response viewed on client side is as below:

A screenshot of the Visual Studio Code interface. The title bar says "udpclient.py - Documents - Visual Studio Code". The menu bar includes File, Edit, Selection, View, Go, Run, Terminal, Help. The left sidebar has icons for file operations like Open, Save, Find, and others. The main area shows a code editor with several tabs: Server1.py, udpclient.py (which is the active tab), UDPmodifiedserver.py, TCPserver.py, TCPClient.py, TCPmodifiedserver.py, ICMP.py. Below the tabs is a "PROBLEMS" section which is currently empty. The "OUTPUT" tab shows the terminal output. The "TERMINAL" tab is selected and displays the following text:

```
● khushbu@khushbu-VirtualBox:~/Documents$ python3 udpclient.py
Enter server port: 11000
Enter number of pings: 13
Response from server: PING 1 1725953830.3149152
Sequence number: 1, RTT: 164.984 ms
Response from server: PING 2 1725953830.4801238
Sequence number: 2, RTT: 9.501 ms
Response from server: PING 3 1725953830.4902184
Sequence number: 3, RTT: 9.602 ms
Response from server: PING 4 1725953830.5000834
Sequence number: 4, RTT: 13.831 ms
Response from server: PING 5 1725953830.5140245
Sequence number: 5, RTT: 14.020 ms
Response from server: PING 6 1725953830.528209
Sequence number: 6, RTT: 17.584 ms
Request timed out for the packet #7
Response from server: PING 8 1725953832.5492961
Sequence number: 8, RTT: 179.590 ms
Response from server: PING 9 1725953832.729126
Sequence number: 9, RTT: 203.833 ms
Request timed out for the packet #10
Response from server: PING 11 1725953834.939953
Sequence number: 11, RTT: 144.548 ms
Response from server: PING 12 1725953835.0845754
Sequence number: 12, RTT: 104.001 ms
Response from server: PING 13 1725953835.188648
Sequence number: 13, RTT: 9.461 ms
Min RTT: 9.46 ms
Max RTT: 203.83 ms
Average RTT: 79.18 ms
Packet Loss Rate: 15.38%
● khushbu@khushbu-VirtualBox:~/Documents$
```

The status bar at the bottom shows "Ln 26, Col 63" and "Python".

Here, on the client side we can see the RTT for each response and minimum RTT, maximum RTT, average rtt. The packet loss value is 15.38%.

The Response viewed on Server side is as follows:

```
(manikanta@Mani)-[~/Documents/Python Projects/Assignment1]
$ python3 Server1.py
Server is ready to receive pings...
Received message: PING 1 1725953830.3149152 from ('172.21.135.150', 65119), Random value: 1
Responding to the client...
Received message: PING 2 1725953830.4801238 from ('172.21.135.150', 65119), Random value: 4
Responding to the client...
Received message: PING 3 1725953830.4902184 from ('172.21.135.150', 65119), Random value: 7
Responding to the client...
Received message: PING 4 1725953830.5000834 from ('172.21.135.150', 65119), Random value: 4
Responding to the client...
Received message: PING 5 1725953830.5140245 from ('172.21.135.150', 65119), Random value: 1
Responding to the client...
Received message: PING 6 1725953830.528209 from ('172.21.135.150', 65119), Random value: 8
Responding to the client...
Received message: PING 7 1725953830.5459611 from ('172.21.135.150', 65119), Random value: 10
Simulating packet loss...
Received message: PING 8 1725953832.5492961 from ('172.21.135.150', 65119), Random value: 7
Responding to the client...
Received message: PING 9 1725953832.729126 from ('172.21.135.150', 65119), Random value: 5
Responding to the client...
Received message: PING 10 1725953832.933111 from ('172.21.135.150', 65119), Random value: 10
Simulating packet loss...
Received message: PING 11 1725953834.939953 from ('172.21.135.150', 65119), Random value: 6
Responding to the client...
Received message: PING 12 1725953835.0845754 from ('172.21.135.150', 65119), Random value: 6
Responding to the client...
Received message: PING 13 1725953835.188648 from ('172.21.135.150', 65119), Random value: 5
Responding to the client...
```

Here, we can see the pings and client IP address.

Modified Server:

1. UDP Socket:
 - We are creating UDP socket by AF_INET which is the IPv4. Bind() function used to bind the socket to all available networks and the port number 11000.
2. While loop for receive and response:
 - Recvfrom() function used to receive message from the udp client up to 1024 bytes.
 - Message.upper() converts the message to uppercase.
3. Sendto():
 - If there are no packet loss, sendto() sends modified message back to the client.

The server code is as shown below:

A screenshot of a code editor window titled "ACN_assignment -2". The left sidebar shows a file tree with several Python files: tcpclient.py, UdpPingingclient.py, Readme.md, udp_modified_server.py, tcp_modified_server.py, tcp_ping_client.py, tcp_ping_server.py, tcpclient.py, tcpconcurrentserver.py, tcpmodifiedserver.py, tcppingclient, tcppingserver.py, tcpserver.py, tcpservertry.py, udp_modified_server.py, UdpPingingclient.py, UDPpingclient.py, UDPpingserver.py, and udpserver.py. The main pane displays the content of the "udp_modified_server.py" file. The code implements a UDP server that receives messages from clients, capitalizes them, and sends responses back. It includes logic to handle packet loss by skipping responses for messages with sequence numbers greater than 8.

```
1 #import random      # importing random to generate the random numbers
2 from socket import * #importing fucntions and constants from the socket module
3
4 serverSocket = socket(AF_INET, SOCK_DGRAM) #creating UDP socket
5
6 # Assign IP address and port number to socket
7 serverSocket.bind(('', 11000)) # binding socket to the IP address
8 print("Server is ready to receive pings...") #printing message server is ready
9
10 while True:
11     # Generate a random number between 1 to 10 (both inclusive)
12     rand = random.randint(1, 10)
13
14     # Receive the client packet along with the address it is coming from
15     message, address = serverSocket.recvfrom(1024)
16
17     # Capitalize the message from the client
18     message = message.upper()
19
20     print(f"Received message: {message.decode()} from {address}")
21
22     # If rand is greater than 8, we consider the packet lost and do not respond to the client
23     #if rand > 8:
24     #    print("Simulating packet loss...") #print the packet loss message
25     #    continue
26
27     # Otherwise, send the response back to the client
28     serverSocket.sendto(message, address)
29     print("Responding to the client...")
```

The Response viewed on internal:

A screenshot of a terminal window titled "(manikanta@Mani) [~/Documents/Python Projects/Assignment1]". The user runs the command "python3 UDPmodifiedserver.py". The server starts listening and receives 10 ping messages from the client. The server responds to the first 8 messages and ignores the last 2 due to simulated packet loss (sequence numbers 9 and 10). The output shows the host IP address (127.0.0.1), port (11000), number of pings (10), and detailed timing information for each response.

```
KeyboardInterrupt
--(manikanta@Mani) [~/Documents/Python Projects/Assignment1]
└$ sudo tc qdisc add dev lo root netem loss 20%
--(manikanta@Mani) [~/Documents/Python Projects/Assignment1]
└$ python3 UDPmodifiedserver.py
Server is ready to receive pings...
Received message: PING 1 1725952463.4850192 from ('127.0.0.1', 41408)
Responding to the client...
Received message: PING 2 1725952463.4852443 from ('127.0.0.1', 41408)
Responding to the client...
Received message: PING 3 1725952463.4853597 from ('127.0.0.1', 41408)
Responding to the client...
Received message: PING 4 1725952464.4868836 from ('127.0.0.1', 41408)
Responding to the client...
Received message: PING 5 1725952464.4868836 from ('127.0.0.1', 41408)
Received message: PING 6 1725952465.4880335 from ('127.0.0.1', 41408)
Responding to the client...
Received message: PING 7 1725952465.4882483 from ('127.0.0.1', 41408)
Responding to the client...
Received message: PING 8 1725952466.4893253 from ('127.0.0.1', 41408)
Responding to the client...
Received message: PING 9 1725952466.489485 from ('127.0.0.1', 41408)
Received message: PING 10 1725952466.489606 from ('127.0.0.1', 41408)
Responding to the client...
--(manikanta@Mani) [~/Documents/Python Projects/Assignment1]
└$
```

Here, we run the modified server code on the local host. We can see the host ip address, minimum rtt, , maximum rtt and average rtt. The packet loss rate is 30%.

We use sudo tc qdisc add dev lo root netem loss 20% command in linux terminal to simulate packet loss on specific network interface (i.e lo)

The Response viewed on client side is as below:

```
khushbu@khushbu-VirtualBox:~/Documents$ python3 udpclient.py
Enter server port: 11000
Enter number of pings: 15
Response from server: PING 1 1725953636.2917125
Sequence number: 1, RTT: 36.978 ms
Response from server: PING 2 1725953636.328858
Sequence number: 2, RTT: 9.487 ms
Request timed out for the packet #3
Request timed out for the packet #4
Response from server: PING 5 1725953640.3627703
Sequence number: 5, RTT: 162.234 ms
Response from server: PING 6 1725953640.5251834
Sequence number: 6, RTT: 104.763 ms
Response from server: PING 7 1725953640.6300766
Sequence number: 7, RTT: 13.909 ms
Response from server: PING 8 1725953640.6441288
Sequence number: 8, RTT: 16.325 ms
Response from server: PING 9 1725953640.6606221
Sequence number: 9, RTT: 9.874 ms
Response from server: PING 10 1725953640.670655
Sequence number: 10, RTT: 59.318 ms
Response from server: PING 11 1725953640.730103
Sequence number: 11, RTT: 9.943 ms
Response from server: PING 12 1725953640.7401612
Sequence number: 12, RTT: 43.127 ms
Request timed out for the packet #13
Response from server: PING 14 1725953642.7858193
Sequence number: 14, RTT: 197.695 ms
Response from server: PING 15 1725953642.9836364
Sequence number: 15, RTT: 204.376 ms
Min RTT: 9.49 ms
Max RTT: 204.38 ms
Average RTT: 72.34 ms
Packet Loss Rate: 20.00%
khushbu@khushbu-VirtualBox:~/Documents$
```

Here, we run the modified server code on the different machines. We can see the minimum rtt, , maximum rtt and average rtt. The packet loss rate is 20%.

The Response viewed on Server side is as follows:

```
(manikanta@Mani)-[~/Documents/Python Projects/Assignment1]
$ python3 UDPmodifiedserver.py
Server is ready to receive pings...
Received message: PING 1 1725953636.2917125 from ('172.21.135.150', 62832),
Responding to the client...
Received message: PING 2 1725953636.328858 from ('172.21.135.150', 62832),
Responding to the client...
Received message: PING 3 1725953636.338476 from ('172.21.135.150', 62832),
Responding to the client...
Received message: PING 4 1725953638.3419523 from ('172.21.135.150', 62832),
Responding to the client...
Received message: PING 5 1725953640.3627703 from ('172.21.135.150', 62832),
Responding to the client...
Received message: PING 6 1725953640.5251834 from ('172.21.135.150', 62832),
Responding to the client...
Received message: PING 7 1725953640.6300766 from ('172.21.135.150', 62832),
Responding to the client...
Received message: PING 8 1725953640.6441288 from ('172.21.135.150', 62832),
Responding to the client...
Received message: PING 9 1725953640.6606221 from ('172.21.135.150', 62832),
Responding to the client...
Received message: PING 10 1725953640.670655 from ('172.21.135.150', 62832),
Responding to the client...
Received message: PING 11 1725953640.730103 from ('172.21.135.150', 62832),
Responding to the client...
Received message: PING 12 1725953640.7401612 from ('172.21.135.150', 62832)
Responding to the client...
Received message: PING 13 1725953640.7833645 from ('172.21.135.150', 62832)
Responding to the client...
Received message: PING 14 1725953642.7858193 from ('172.21.135.150', 62832)
Responding to the client...
Received message: PING 15 1725953642.9836364 from ('172.21.135.150', 62832)
Responding to the client...

```

ICMP error message:

We are creating a UDP server which will listen for incoming client connections. Simulate ICMP errors to demonstrate how network errors can be handled.

1. UDP Socket:

- We are creating UDP socket by AF_INET which is the IPv4. Client.connect() function used to connect to the all-available networks and the port number 12000.

2. ICMP error :

- We have also created a raw socket for sending icmp error message.
- create_icmp_error this function is used to generate a icmp error message.
- It is constructed with checksum and if randomly generated is greater than 8 then it will show the icmp error messages.
- If no error is generated then we will get the normal response.

The server code is as shown below:

```

File Edit Selection View Go ... ← → ⌂ ACN_assignment - 2
EXPLORER udp_modified_server.py tcp_modified_server.py TCPping_client.py TCPpingICMPerrorserver.py UDPPingerICMPerrorserver.py ...
icmp_code.py rjtcpclient.py README.md
tcp_modified_server.py TCPping_server.py TCPclient.py
TCPconcurrentserver.py TCPmodifiedserver.py TCPpingICMPerror...
TCPpingICMPerrors...
TCPpingclient.py UDPpingclient.py UDPpingerrorserver.py
UDPPingerICMPerror...
UDPPingerserver.py
udpserver.py
OUTLINE
Finance headline
India reported 4...
In 33, Col 27 Spaces: 4 UTF-8 (F) Python 3.10.2 22:53 12-09-2024

```

```

from socket import *
import socket
import random
import struct
import os

# Constants for ICMP message
ICMP_TYPE_DEST_UNREACHABLE = 3
ICMP_CODE_PORT_UNREACHABLE = 3

# Function to create an ICMP error packet (simulates unreachable port)
def create_icmp_error(dest_addr, code=ICMP_CODE_PORT_UNREACHABLE):
    packet_type = ICMP_TYPE_DEST_UNREACHABLE
    checksum = 0 # Placeholder for now, we'll calculate it next
    packet_id = os.getpid() & 0xFFFF # Use the process ID for identification
    seq_number = 1 # Just a simple sequence number

    # Create the ICMP header (checksum is zero for now)
    header = struct.pack('bbHH', packet_type, code, checksum, packet_id, seq_number)

    # Calculate checksum for ICMP (used for error detection)
    def calculate_checksum(source_string):
        countTo = (len(source_string) // 2) * 2
        sum = 0
        count = 0

        # Add 2 bytes at a time
        while count < countTo:
            this_val = source_string[count + 1] * 256 + source_string[count]
            sum = sum + this_val
            sum = sum & 0xffffffff # Keep within 32 bits

    checksum = calculate_checksum(header) # Calculate correct checksum
    # Pack the header again with the correct checksum
    header = struct.pack('bbHH', packet_type, code, socket.htons(checksum), packet_id, seq_number)

    return header # Return the final ICMP header

```

The screenshot shows a Python code editor with the following details:

- Title Bar:** ACN assignment - 2
- File Explorer:** Shows multiple Python files in the project, including `ACN AS...`, `icmp_code.py`, `rajtpingclient.py`, `README.md`, `tcp_modified_server.py`, `tcp_ping_client.py`, `TCPPingerrICMPerrorserver.py`, `TCPPingerrICMPerrorclient.py`, and `UDPPingerICMPerrorserver.py`.
- Code Editor:** The current file is `UDPPingerICMPerrorserver.py`. The code defines a `udp_server()` function that creates a UDP socket, binds it to port 12000, and then enters a loop to receive data from clients. It checks if a random number is greater than 8 (indicating an ICMP error) and then either sends a simulated ICMP error or a normal UDP response back to the client. Finally, it runs the server.

```
def udp_server():
    udp_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    udp_socket.bind(("0.0.0.0", 12000)) # Bind to port 12000

    # Create raw ICMP socket for sending ICMP errors
    icmp_socket = socket.socket(socket.AF_INET, socket.SOCK_RAW, socket.getprotobynumber("icmp"))

    print("Server is running and waiting for UDP connections...")

    while True:
        # Receive UDP data and client address
        data, addr = udp_socket.recvfrom(1024)
        print(f"Received UDP request from {addr}")

        # Generate a random number (to decide if we'll simulate an ICMP error)
        rand_num = random.randint(1, 10)
        print(f"Random number: ({rand_num})")

        if rand_num > 8:
            # If random number is high, simulate ICMP error
            print(f"Simulating ICMP error for {addr[0]}")
            icmp_error = create_icmp_error(addr[0])
            icmp_socket.sendto(icmp_error, addr)
        else:
            # Otherwise, just echo back the data to the client
            print(f"Sending normal UDP response to {addr}")
            udp_socket.sendto(data, addr)

    # Run the server
    udp_server()
```

- Bottom Status Bar:** Shows file statistics (Ln 33, Col 27), code analysis (Spaces: 4, LF: 1), and system information (Python 3.10.2, ENG IN, 2254, 12-09-2024).

The client code is as shown below:

The screenshot shows a code editor interface with multiple tabs open, all related to network communication. The tabs include:

- EXPLORER
- ACN AS...
- udp_modified_server.py
- tcp_modified_server.py
- tcp_ping_client.py
- TCPPingICMPerrorserver.py
- TCPpingICMPerrorclient.py
- UDPPingICMPerrorclient.py

The main pane displays the source code for `UDPPingICMPerrorclient.py`. The code is as follows:

```
import socket # We'll need the socket module for network communication
import os # Used for accessing OS-level functions like process ID
import time # We'll use the time module to handle timeouts and delays
import struct # Needed for packing and unpacking binary data (like ICMP headers)
import select # For monitoring socket readiness with a timeout

# ICMP constants: type 8 is for Echo Request (i.e., a ping request)
ICMP_ECHO_REQUEST = 8

# Mapping ICMP "Destination Unreachable" codes to human-readable messages
ICMP_CODE_UNREACHABLE = {
    0: "Destination Network Unreachable",
    1: "Destination Host Unreachable",
    3: "Port Unreachable",
}

# Function to calculate the checksum for the ICMP packet (essentially a way to ensure data integrity)
def checksum(source_string):
    sum = 0 # This will accumulate the sum of the data
    # We'll process two bytes at a time, so we calculate how many full 2-byte chunks there are
    countIO = (len(source_string) // 2) * 2
    count = 0

    # Loop through the string two bytes at a time
    while count < countIO:
        # Convert the two bytes into a single 16-bit value (big-endian format)
        thisVal = source_string[count + 1] * 256 + source_string[count]
        sum = sum + thisVal # Add the value to the running sum
        sum = sum & 0xffffffff # Make sure we don't overflow beyond 32 bits
        count = count + 2 # Move to the next two bytes

    # If the length of the string is odd, we have one last byte to process
    if countIO < len(source_string):
        sum = sum + source_string[len(source_string) - 1]
        sum = sum & 0xffffffff # Again, ensure we don't exceed 32 bits

    return sum
```

```

File Edit Selection View Go ... < > ACN_assignment -2
EXPLORER ... udp_modified_server.py tcp_modified_server.py tcp_ping_client.py TCPPingerICMPerrorserver.py TCPpingrICMPerrorclient.py UDPPingerICMPerrorclient.py
icmp_code.py README.md
tcpclient.py
top_modified_server.py
tcp_ping_client.py
tcpclient.py
topconcurrentserver.py
topmodifiedserver.py
TCPpingrICMPerror...
tcpserver.py
tcpseverver.py
tcpseververing.py
udp_modified_server.py
udp_sample_client.py
udp_sample.py
UsPingingclient.py
UDPPingerclient.py
UDPPingerICMPerror...
UDPPingerICMPerror...
UDPpingerserver.py
udpserver.py
115 # Run the UDP client to send pings to the specified server
116 udp_client("127.0.0.1", num_pings=10)

81 def udp_client(server_ip, num_pings=10, timeout=5):
82     udp_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
83     # Create a raw ICMP socket to listen for ICMP error messages (like "Destination Unreachable")
84     icmp_socket = socket.socket(socket.AF_INET, socket.SOCK_RAW, socket.getprotobymode("icmp"))
85
86     # Loop through the number of pings we want to send
87     for i in range(num_pings):
88         try:
89             print(f"Sending ping {i+1}/{num_pings} to {server_ip}")
90             send_udp_ping(udp_socket, server_ip) # Send a ping via UDP
91
92             # After sending the ping, listen for any ICMP errors
93             icmp_error = receive_icmp_error(icmp_socket, timeout)
94             if "Error" in icmp_error:
95                 # If we receive an ICMP error, print it
96                 print(icmp_error)
97             else:
98                 # If no error, assume we received a valid response
99                 print(f"Received from {server_ip}")
100
101             # Handle cases where the socket times out waiting for a response
102             except socket.timeout:
103                 print("Request timed out.")
104
105             # Catch any other errors that might occur during the process
106             except Exception as e:
107                 print(f"An error occurred: {e}")
108
109             time.sleep(1) # Wait for 1 second before sending the next ping
110
111             # Close both the UDP and ICMP sockets when we're done
112             udp_socket.close()
113             icmp_socket.close()
114
115 # Run the UDP client to send pings to the specified server
116 udp_client("127.0.0.1", num_pings=10)

```

Ln 55, Col 40 Spaces: 4 UTF-8 LF Python 3.10.2

Finance headline
India reported 4...

ENG IN 22:53 12-09-2024

The Response viewed on client side is as follows:

```

● kasare@kasare-IdeaPad-Gaming-3-15IMH05:~/Desktop$ sudo python3 UDPICMP.py
Sending ping 1/20 to 192.168.173.217
Received from 192.168.173.217
Sending ping 2/20 to 192.168.173.217
Received from 192.168.173.217
Sending ping 3/20 to 192.168.173.217
Received from 192.168.173.217
Sending ping 4/20 to 192.168.173.217
Received from 192.168.173.217
Sending ping 5/20 to 192.168.173.217
Error: Port Unreachable
Sending ping 6/20 to 192.168.173.217
Received from 192.168.173.217
Sending ping 7/20 to 192.168.173.217
Received from 192.168.173.217
Sending ping 8/20 to 192.168.173.217
Received from 192.168.173.217
Sending ping 9/20 to 192.168.173.217
Received from 192.168.173.217
Sending ping 10/20 to 192.168.173.217
Received from 192.168.173.217
Sending ping 11/20 to 192.168.173.217
Received from 192.168.173.217
Sending ping 12/20 to 192.168.173.217
Received from 192.168.173.217
Sending ping 13/20 to 192.168.173.217
Received from 192.168.173.217
Sending ping 14/20 to 192.168.173.217
Received from 192.168.173.217
Sending ping 15/20 to 192.168.173.217
Received from 192.168.173.217
Sending ping 16/20 to 192.168.173.217
Received from 192.168.173.217
Sending ping 17/20 to 192.168.173.217
Error: Port Unreachable
Sending ping 18/20 to 192.168.173.217

```

The Response viewed on client side is as follows:

```
(manikanta@Mani)-[~/Documents/Python Projects/Assignment1]
$ sudo python3 UDPPingerICMPerrorserver.py
[sudo] password for manikanta:
Sorry, try again.
[sudo] password for manikanta:
Server is running and waiting for UDP connections...
Received UDP request from ('192.168.173.187', 48302)
Random number: 2
Sending normal UDP response to ('192.168.173.187', 48302)
Received UDP request from ('192.168.173.187', 48302)
Random number: 4
Sending normal UDP response to ('192.168.173.187', 48302)
Received UDP request from ('192.168.173.187', 48302)
Random number: 7
Sending normal UDP response to ('192.168.173.187', 48302)
Received UDP request from ('192.168.173.187', 48302)
Random number: 8
Sending normal UDP response to ('192.168.173.187', 48302)
Received UDP request from ('192.168.173.187', 48302)
Random number: 10
Simulating ICMP error for ('192.168.173.187', 48302)
Received UDP request from ('192.168.173.187', 48302)
Random number: 1
Sending normal UDP response to ('192.168.173.187', 48302)
Received UDP request from ('192.168.173.187', 48302)
Random number: 8
Sending normal UDP response to ('192.168.173.187', 48302)
Received UDP request from ('192.168.173.187', 48302)
Random number: 5
Sending normal UDP response to ('192.168.173.187', 48302)
Received UDP request from ('192.168.173.187', 48302)
Random number: 10
Simulating ICMP error for ('192.168.173.187', 48302)
Received UDP request from ('192.168.173.187', 48302)
Random number: 1
Sending normal UDP response to ('192.168.173.187', 48302)
Received UDP request from ('192.168.173.187', 48302)
Random number: 9
Simulating ICMP error for ('192.168.173.187', 48302)
Received UDP request from ('192.168.173.187', 48302)
Random number: 2
Sending normal UDP response to ('192.168.173.187', 48302)
Received UDP request from ('192.168.173.187', 48302)
Random number: 9
Simulating ICMP error for ('192.168.173.187', 48302)
Received UDP request from ('192.168.173.187', 48302)
Random number: 5
Sending normal UDP response to ('192.168.173.187', 48302)
Received UDP request from ('192.168.173.187', 48302)
Random number: 2
Sending normal UDP response to ('192.168.173.187', 48302)
Received UDP request from ('192.168.173.187', 48302)
Random number: 4
Sending normal UDP response to ('192.168.173.187', 48302)
```

TCP:

The ping tool simulated using TCP in the following code and the server, modified server and client side of the code is the task at hand here. However, the additional tasks here are to calculate the RTT statistics like the maximum ping time, minimum and the average ping time, alongside the packet loss percentage. The step flow is as follows:

Server:

1. TCP Socket:
 - We are creating TCP socket by AF_INET which is the IPv4. Bind() function used to bind the socket to all available networks and the port number 12000.
2. Server.listen():
 - It is used to accepting incoming connection request from client.
3. Server.accept():
 - Server.accept() is used to accept the connections and return the client address.
4. While loop:
 - Server wait for client to connect and when connection is established server enters in nested loop and handle the communication.
 - Recvfrom() function used to receive message from the tcp client up to 1024 bytes.
 - If no data is received then loop will break.
5. Packet loss:
 - Here, we generate random number between 1 and 10 using the random.randint(1,10) for simulate the packet loss. We put a condition if rand>8 means if the random number is greater than 8 then server simulates the packet loss and print simulating packet loss.
6. Connection close :
 - client_socket.close() is used to close the client connection.

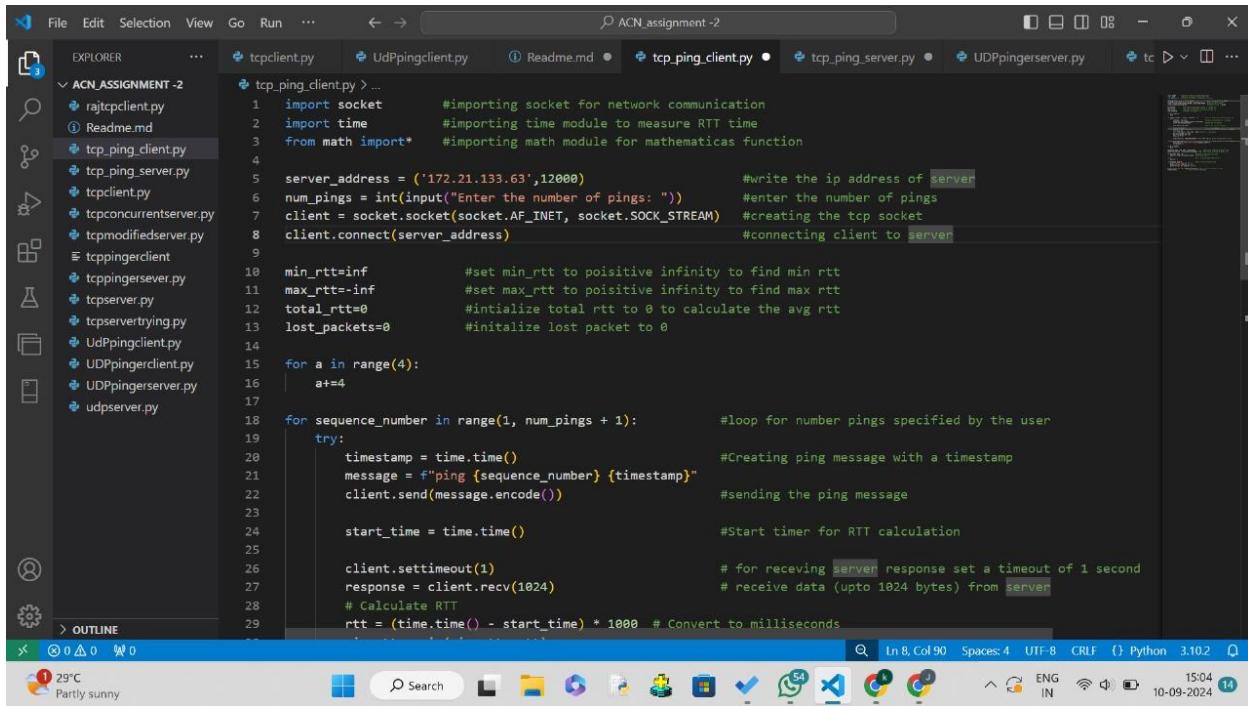
The server code is as shown below:

```
File Edit Selection View Go ... ACN_assignment -2
EXPLORER
tcpclient.py UDPpingclient.py Readme.md tcp_ping_server.py UDPpingserver.py tcpservertrying.py tcpconcurrentserver.py rajtcpclient.py
tcp_ping_server.py
1 import socket      #importing socket for network communication
2 import random      #importing random to generate the random numbers
3
4 server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)    #creating tcp socket
5 server.bind(('', 12000))                                         #binding socket to the IP address
6
7 server.listen(4)                                                 #listens for incoming connection with queue upto 4
8
9 print('TCP_Pingserver is listening now ... ')
10
11 while True:
12     client_socket, client_address = server.accept()           #Accepting the incoming connection
13
14     print(f'Connected to {client_address}')                      #printing the client address
15
16     while True:
17         data = client_socket.recv(1024)                         #handle connection with clients
18
19         if not data:                                           #if no data received from the client side loop will break
20             break
21
22         random_integer = random.randint(1, 10)
23         print(f'Received packet: {data.decode()}')
24         if random_integer >= 8:
25             upper_data = data.upper()
26             client_socket.send(upper_data)
27
28         print(f'Sent response: {upper_data.decode()}')
29     else:
30         print(f'Packet lost (no response)')
31
32     # Close the client socket
33     client_socket.close()                                     #close the client socket
```

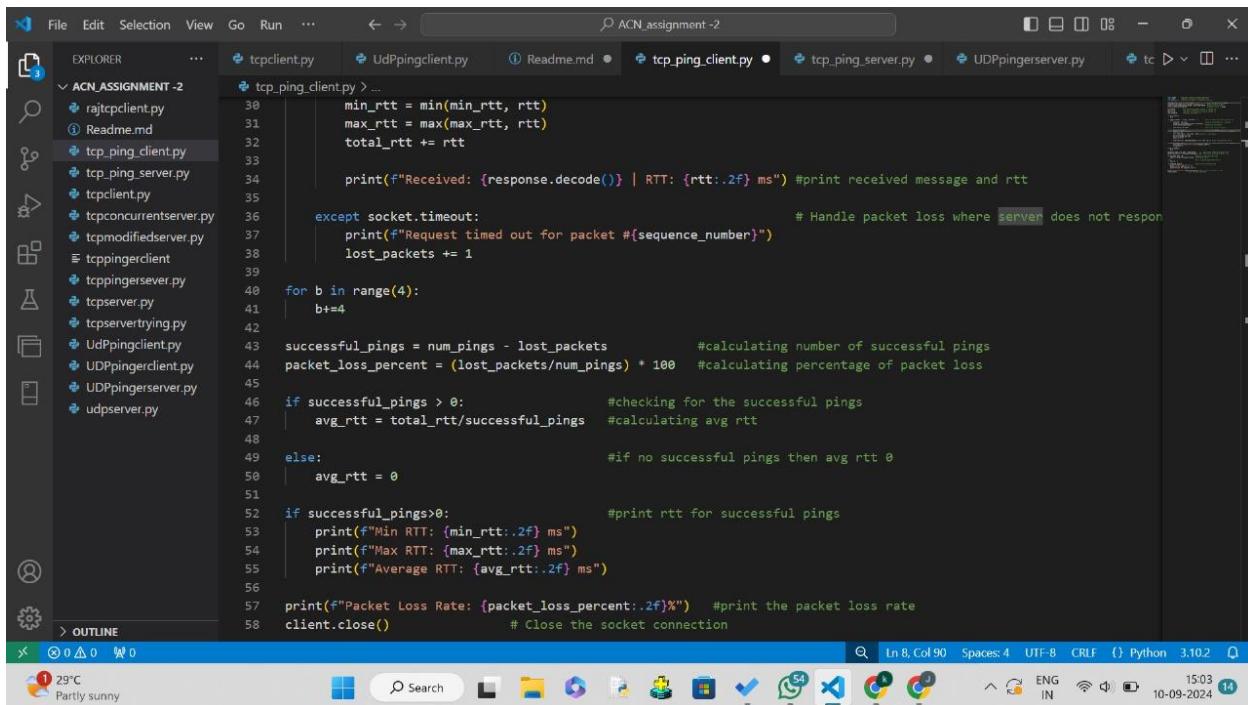
Client:

1. TCP Socket:
 - We are creating TCP socket by AF_INET which is the IPv4. Client.connect() function used to connect to the server and the port number 12000.
2. Ping:
 - Sends a message containing time stamp, sequence number and also notes the sent time for RTT calculation. If no response is confirmed in a second time, There is a packet loss detected as the timeout is triggered.
3. Stats calculation:
 - If the response is received, then It begins calculating the stats of RTT as discussed. And it only calculates the average RTT for the successful packets whilst keeping track of lost packet numbers so as to calculate the packet loss percentage.
4. Client.close():
 - It is used to close the socket after all the pings are apparently received.

The client code is as shown below:



```
tcp_ping_client.py > ...
1 import socket      #importing socket for network communication
2 import time        #importing time module to measure RTT time
3 from math import*  #importing math module for mathematics function
4
5 server_address = ('172.21.133.63',12000)           #write the ip address of server
6 num_pings = int(input("Enter the number of pings: ")) #enter the number of pings
7 client = socket.socket(socket.AF_INET, socket.SOCK_STREAM) #creating the tcp socket
8 client.connect(server_address)                      #connecting client to server
9
10 min_rtt=inf          #set min_rtt to positive infinity to find min rtt
11 max_rtt=-inf         #set max_rtt to negative infinity to find max rtt
12 total_rtt=0           #initialize total rtt to 0 to calculate the avg rtt
13 lost_packets=0        #initialize lost packet to 0
14
15 for a in range(4):
16     a+=4
17
18 for sequence_number in range(1, num_pings + 1):      #loop for number pings specified by the user
19     try:
20         timestamp = time.time()                      #Creating ping message with a timestamp
21         message = f"ping {sequence_number} {timestamp}"
22         client.send(message.encode())
23
24         start_time = time.time()                     #Start timer for RTT calculation
25
26         client.settimeout(1)                         # for receiving server response set a timeout of 1 second
27         response = client.recv(1024)                 # receive data (upto 1024 bytes) from server
28
29         # Calculate RTT
30         rtt = (time.time() - start_time) * 1000    # Convert to milliseconds
31         min_rtt = min(min_rtt, rtt)
32         max_rtt = max(max_rtt, rtt)
33         total_rtt += rtt
34
35         print(f"Received: {response.decode()} | RTT: {rtt:.2f} ms") #print received message and rtt
36
37     except socket.timeout:
38         print(f"Request timed out for packet #{sequence_number}") # Handle packet loss where server does not respond
39         lost_packets += 1
40
41     for b in range(4):
42         b+=4
43
44     successful_pings = num_pings - lost_packets       #calculating number of successful pings
45     packet_loss_percent = (lost_packets/num_pings) * 100 #calculating percentage of packet loss
46
47     if successful_pings > 0:                          #checking for the successful pings
48         avg_rtt = total_rtt/successful_pings          #calculating avg rtt
49
50     else:
51         avg_rtt = 0
52
53     if successful_pings>0:                          #print rtt for successful pings
54         print(f"Min RTT: {min_rtt:.2f} ms")
55         print(f"Max RTT: {max_rtt:.2f} ms")
56         print(f"Average RTT: {avg_rtt:.2f} ms")
57
58 print(f"Packet Loss Rate: {packet_loss_percent:.2f}%") #print the packet loss rate
59 client.close()                                     # Close the socket connection
```



```
tcp_ping_client.py > ...
30     min_rtt = min(min_rtt, rtt)
31     max_rtt = max(max_rtt, rtt)
32     total_rtt += rtt
33
34     print(f"Received: {response.decode()} | RTT: {rtt:.2f} ms") #print received message and rtt
35
36 except socket.timeout:
37     print(f"Request timed out for packet #{sequence_number}") # Handle packet loss where server does not respond
38     lost_packets += 1
39
40 for b in range(4):
41     b+=4
42
43 successful_pings = num_pings - lost_packets       #calculating number of successful pings
44 packet_loss_percent = (lost_packets/num_pings) * 100 #calculating percentage of packet loss
45
46 if successful_pings > 0:                          #checking for the successful pings
47     avg_rtt = total_rtt/successful_pings          #calculating avg rtt
48
49 else:
50     avg_rtt = 0
51
52 if successful_pings>0:                          #print rtt for successful pings
53     print(f"Min RTT: {min_rtt:.2f} ms")
54     print(f"Max RTT: {max_rtt:.2f} ms")
55     print(f"Average RTT: {avg_rtt:.2f} ms")
56
57 print(f"Packet Loss Rate: {packet_loss_percent:.2f}%") #print the packet loss rate
58 client.close()                                     # Close the socket connection
```

The Response viewed on client side is as below:

The screenshot shows a Visual Studio Code interface. The left sidebar contains icons for file operations like copy, search, and refresh. The top menu bar includes File, Edit, Selection, View, Go, Run, Terminal, Help, and a tab for TCPClient.py. Below the menu is a toolbar with PROBLEMS, OUTPUT, DEBUG CONSOLE, TERMINAL (which is selected), and PORTS. The main editor area displays a Python script named TCPClient.py:

```
1 import socket      #importing socket for network communication
2 import time        #importing time module to measure RTT time
3 from math import* #importing math module for mathematicalas function
4
5 server_address = ('172.21.133.63',12000)          #write the ip address of server
6 num_pings = int(input("Enter the number of pings: ")) #enter the number of pings
7 client = socket.socket(socket.AF_INET, socket.SOCK_STREAM) #creating the tcp socket
8 client.connect(server_address)                      #connecting client to server
9
10
```

The terminal pane below shows the execution of the script:

```
● khushbu@khushbu-VirtualBox:~/Documents$ python3 TCPClient.py
Enter the number of pings: 13
Received: PING 1 1725953948.9555027 | RTT: 205.19 ms
Received: PING 2 1725953949.1611853 | RTT: 199.95 ms
Request timed out for packet #3
Received: PING 4 1725953950.3632216 | RTT: 24.42 ms
Request timed out for packet #5
Received: PING 6 1725953951.3888993 | RTT: 225.90 ms
Received: PING 7 1725953951.615089 | RTT: 108.17 ms
Received: PING 8 1725953951.7238193 | RTT: 8.80 ms
Received: PING 9 1725953951.735971 | RTT: 11.36 ms
Received: PING 10 1725953951.7476087 | RTT: 8.55 ms
Received: PING 11 1725953951.7566798 | RTT: 14.72 ms
Received: PING 12 1725953951.7716372 | RTT: 10.72 ms
Received: PING 13 1725953951.7829857 | RTT: 8.83 ms
Min RTT: 8.55 ms
Max RTT: 225.90 ms
Average RTT: 75.15 ms
Packet Loss Rate: 15.38%
○ khushbu@khushbu-VirtualBox:~/Documents$
```

At the bottom, status indicators show Ln 6, Col 33, Spaces: 4, UTF-8, LF, Python, and a file icon.

Here, on the client side we can see the RTT for each response and minimum RTT, maximum RTT, average rtt. The packet loss value is 15.38%.

The Response viewed on Server side is as follows:

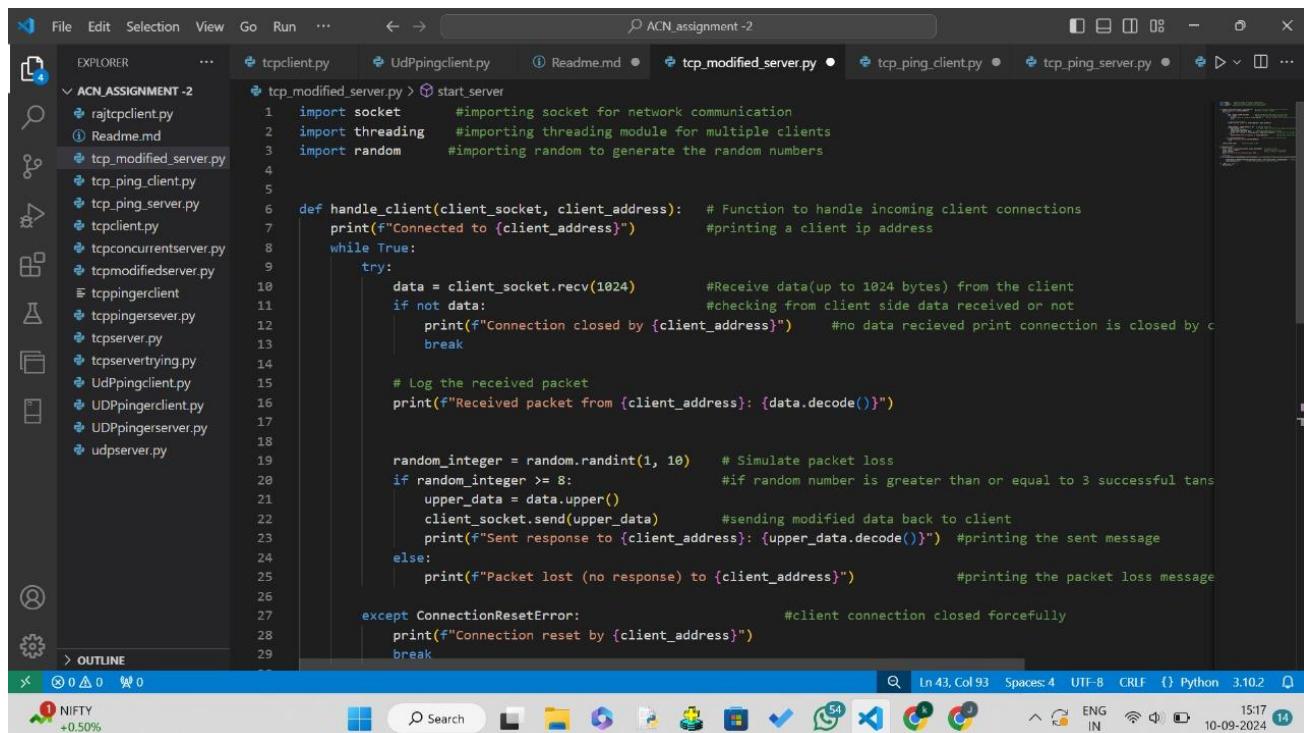
```
(manikanta@Mani)-[~/Documents/Python Projects/Assignment1]
$ python3 TCPserver.py
TCP_Pingserver is listening now ...
Connected to ('172.21.135.150', 58683)
Received packet: ping 1 1725953948.9555027
Sent response: PING 1 1725953948.9555027
Received packet: ping 2 1725953949.1611853
Sent response: PING 2 1725953949.1611853
Received packet: ping 3 1725953949.361377
Packet lost (no response)
Received packet: ping 4 1725953950.3632216
Sent response: PING 4 1725953950.3632216
Received packet: ping 5 1725953950.387832
Packet lost (no response)
Received packet: ping 6 1725953951.3888993
Sent response: PING 6 1725953951.3888993
Received packet: ping 7 1725953951.615089
Sent response: PING 7 1725953951.615089
Received packet: ping 8 1725953951.7238193
Sent response: PING 8 1725953951.7238193
Received packet: ping 9 1725953951.735971
Sent response: PING 9 1725953951.735971
Received packet: ping 10 1725953951.7476087
Sent response: PING 10 1725953951.7476087
Received packet: ping 11 1725953951.7566798
Sent response: PING 11 1725953951.7566798
Received packet: ping 12 1725953951.7716372
Sent response: PING 12 1725953951.7716372
Received packet: ping 13 1725953951.7829857
Sent response: PING 13 1725953951.7829857
```

We can see the client ip address and response message on the server side terminal.

Modified Server:

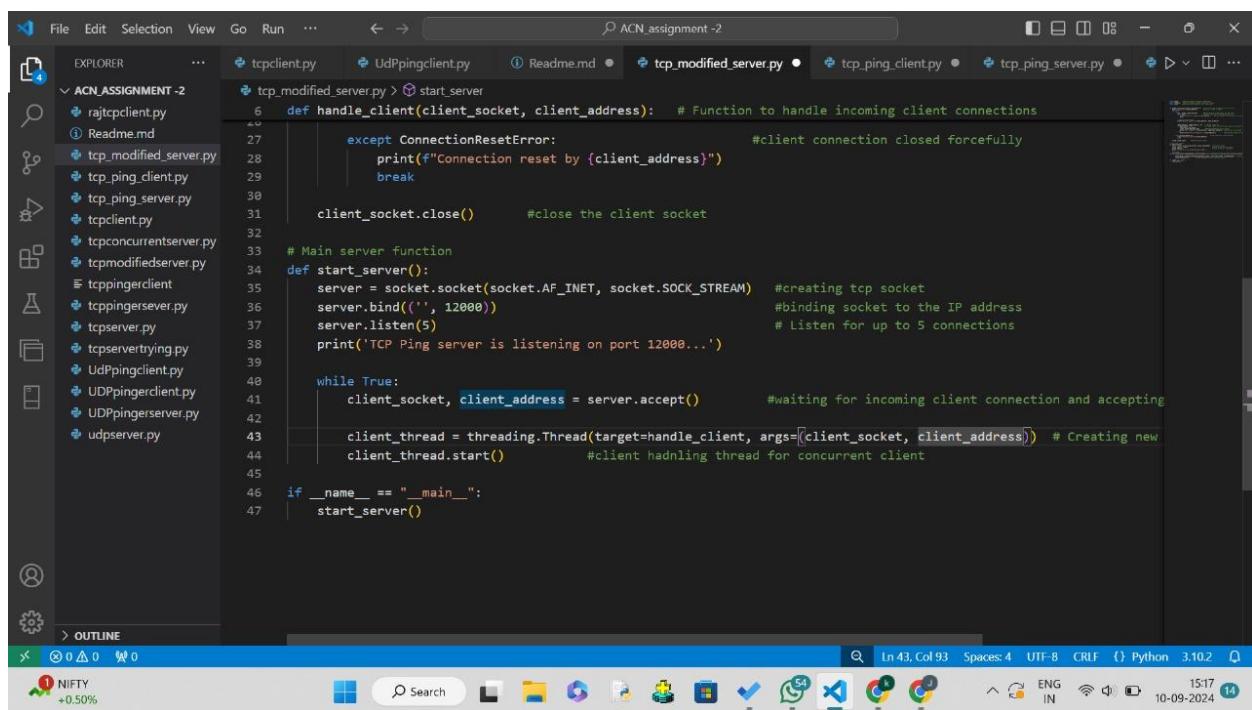
1. TCP Socket:
 - We are creating TCP socket by AF_INET which is the IPv4. Client.connect() function used to connect to the all-available networks and the port number 12000.
2. Server.listen():
 - It is used to accepting incoming connection request from client.
3. Server.accept():
 - Server.accept() is used to accept the connections and return the client address.
4. Start_server():
 - It is used to accepting the incoming connections.
 - Client_thread is used to execute new thread and execute the new connection.

The server code is as shown below:



```
import socket
import threading
import random

def handle_client(client_socket, client_address):
    print(f"Connected to {client_address}")
    while True:
        try:
            data = client_socket.recv(1024)
            if not data:
                print(f"Connection closed by {client_address}")
                break
            print(f"Received packet from {client_address}: {data.decode()}")
            random_integer = random.randint(1, 10)
            if random_integer >= 8:
                upper_data = data.upper()
                client_socket.send(upper_data)
                print(f"Sent response to {client_address}: {upper_data.decode()}")
            else:
                print(f"Packet lost (no response) to {client_address}")
        except ConnectionResetError:
            print(f"Connection reset by {client_address}")
            break
```



```
def handle_client(client_socket, client_address):
    print(f"Connected to {client_address}")
    except ConnectionResetError:
        print(f"Connection reset by {client_address}")
        break
    client_socket.close()

# Main server function
def start_server():
    server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    server.bind(' ', 12000)
    server.listen(5)
    print('TCP Ping server is listening on port 12000...')

    while True:
        client_socket, client_address = server.accept()
        client_thread = threading.Thread(target=handle_client, args=[client_socket, client_address])
        client_thread.start()

if __name__ == "__main__":
    start_server()
```

The Response viewed on multi-client side is as below:

```

TCPClient.py - Documents - Visual Studio Code
File Edit Selection View Go Run Terminal Help
Server1.py udpclient.py UDPmodifiedserver.py TCPserver.py TCPClient.py X TCPmodifiedserver.py ICMP.py ...
TCPClient.py
1
2 import socket      #importing socket for network communication
3 import time        #importing time module to measure RTT time
4 from math import* #importing math module for mathematical functions
5
6 server_address = ('172.21.133.63',12000)           #write the ip address of server
7 num_pings = int(input("Enter the number of pings: ")) | #enter the number of pings
8 client = socket.socket(socket.AF_INET, socket.SOCK_STREAM) #creating the tcp socket
9 client.connect(server_address)                      #connecting client to server
10
11 min_rtt=inf          #set min_rtt to positive infinity to find min_rtt
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
• khushbu@khushbu-VirtualBox:~/Documents$ python3 TCPClient.py
Enter the number of pings: 15
Received: PING 1 1725954210.0295997 | RTT: 9.26 ms
Request timed out for packet #2
Received: PING 3 1725954211.0424895 | RTT: 156.48 ms
Received: PING 4 1725954211.199245 | RTT: 9.06 ms
Received: PING 5 1725954211.208524 | RTT: 20.63 ms
Received: PING 6 1725954211.229967 | RTT: 16.86 ms
Received: PING 7 1725954211.2472491 | RTT: 54.46 ms
Received: PING 8 1725954211.3019722 | RTT: 10.29 ms
Request timed out for packet #9
Request timed out for packet #10
Received: PING 11 1725954213.3261194 | RTT: 639.54 ms
Received: PING 12 1725954213.9659104 | RTT: 9.19 ms
Request timed out for packet #13
Request timed out for packet #14
Request timed out for packet #15
Min RTT: 9.06 ms
Max RTT: 639.54 ms
Average RTT: 102.86 ms
Packet Loss Rate: 40.00%
khushbu@khushbu-VirtualBox:~/Documents$ 

```

For one client number of pings were 15, we can see the minimum rtt, maximum rtt and average rtt value and also the packet loss rate is 40%.

```

File Edit Selection View Go ... ACN_assignment -2
tcpclient.py > UdpPingingClient.py Readme.md UDPpingerserver.py tcpservertrying.py tcpconcurrentserver.py ...
tcpclient.py ...
1 import socket
2 import time
3 from math import*
4
5 server_address = ['172.21.133.63',12000]
6 num_pings = int(input("Enter the number of pings: "))
7 client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE PORTS
PS C:\Users\Naveen\Desktop\ACN_assignment -2> & C:/Users/Naveen/AppData/Local/Programs/Python/Python310/python.exe "c:/Users/Naveen/Desktop/ACN_assignment -2/tcpclient.py"
Enter the number of pings: 6
Request timed out for packet #1
Request timed out for packet #2
Received: PING 3 1725954193.039287 | RTT: 20.98 ms
Received: PING 4 1725954193.0605466 | RTT: 6.35 ms
Received: PING 5 1725954193.0679007 | RTT: 5.67 ms
Received: PING 6 1725954193.074087 | RTT: 6.21 ms
Min RTT: 5.67 ms
Max RTT: 20.98 ms
Average RTT: 9.80 ms
Packet Loss Rate: 33.33%
PS C:\Users\Naveen\Desktop\ACN_assignment -2>

```

At the same time we run the client side on other machine also for number pings were 6 we are getting the minimum rtt, max rtt and avg rtt for all 6 ping and the packet loss rate is 33.33%.

The Response viewed on Server side is as follows:

```
(manikanta@Mani)-[~/Documents/Python Projects/Assignment1]
$ python3 TCPmodifiedserver.py
TCP Ping server is listening on port 12000...
Connected to ('172.21.133.69', 53691)
Received packet from ('172.21.133.69', 53691): ping 1 1725954191.0097036
Packet lost (no response) to ('172.21.133.69', 53691)
Connected to ('172.21.135.150', 58819)
Received packet from ('172.21.135.150', 58819): ping 1 1725954210.0295997
Sent response to ('172.21.135.150', 58819): PING 1 1725954210.0295997
Received packet from ('172.21.135.150', 58819): ping 2 1725954210.0392838
Packet lost (no response) to ('172.21.135.150', 58819)
Received packet from ('172.21.133.69', 53691): ping 2 1725954192.0219042
Packet lost (no response) to ('172.21.133.69', 53691)
Received packet from ('172.21.133.69', 53691): ping 3 1725954193.039287
Received packet from ('172.21.135.150', 58819): ping 3 1725954211.0424895
Sent response to ('172.21.133.69', 53691): PING 3 1725954193.039287
Sent response to ('172.21.135.150', 58819): PING 3 1725954211.0424895
Received packet from ('172.21.133.69', 53691): ping 4 1725954193.0605466
Sent response to ('172.21.133.69', 53691): PING 4 1725954193.0605466
Received packet from ('172.21.133.69', 53691): ping 5 1725954193.0679007
Sent response to ('172.21.133.69', 53691): PING 5 1725954193.0679007
Received packet from ('172.21.133.69', 53691): ping 6 1725954193.074087
Sent response to ('172.21.133.69', 53691): PING 6 1725954193.074087
Connection closed by ('172.21.133.69', 53691)
Received packet from ('172.21.135.150', 58819): ping 4 1725954211.199245
Sent response to ('172.21.135.150', 58819): PING 4 1725954211.199245
Received packet from ('172.21.135.150', 58819): ping 5 1725954211.208524
Sent response to ('172.21.135.150', 58819): PING 5 1725954211.208524
Received packet from ('172.21.135.150', 58819): ping 6 1725954211.229967
Sent response to ('172.21.135.150', 58819): PING 6 1725954211.229967
Received packet from ('172.21.135.150', 58819): ping 7 1725954211.2472491
Sent response to ('172.21.135.150', 58819): PING 7 1725954211.2472491
Received packet from ('172.21.135.150', 58819): ping 8 1725954211.3019722
Sent response to ('172.21.135.150', 58819): PING 8 1725954211.3019722
Received packet from ('172.21.135.150', 58819): ping 9 1725954211.3127265
Packet lost (no response) to ('172.21.135.150', 58819)
Received packet from ('172.21.135.150', 58819): ping 10 1725954212.3214242
Packet lost (no response) to ('172.21.135.150', 58819)
Received packet from ('172.21.135.150', 58819): ping 11 1725954213.3261194
Sent response to ('172.21.135.150', 58819): PING 11 1725954213.3261194
Received packet from ('172.21.135.150', 58819): ping 12 1725954213.9659104
Sent response to ('172.21.135.150', 58819): PING 12 1725954213.9659104
Received packet from ('172.21.135.150', 58819): ping 13 1725954213.9752972
Packet lost (no response) to ('172.21.135.150', 58819)
Received packet from ('172.21.135.150', 58819): ping 14 1725954214.976838
Packet lost (no response) to ('172.21.135.150', 58819)
Received packet from ('172.21.135.150', 58819): ping 15 1725954215.983246
Packet lost (no response) to ('172.21.135.150', 58819)
Connection closed by ('172.21.135.150', 58819)
```

ICMP error message:

We are creating a TCP server which will listen for incoming client connections. Simulate ICMP errors to demonstrate how network errors can be handled.

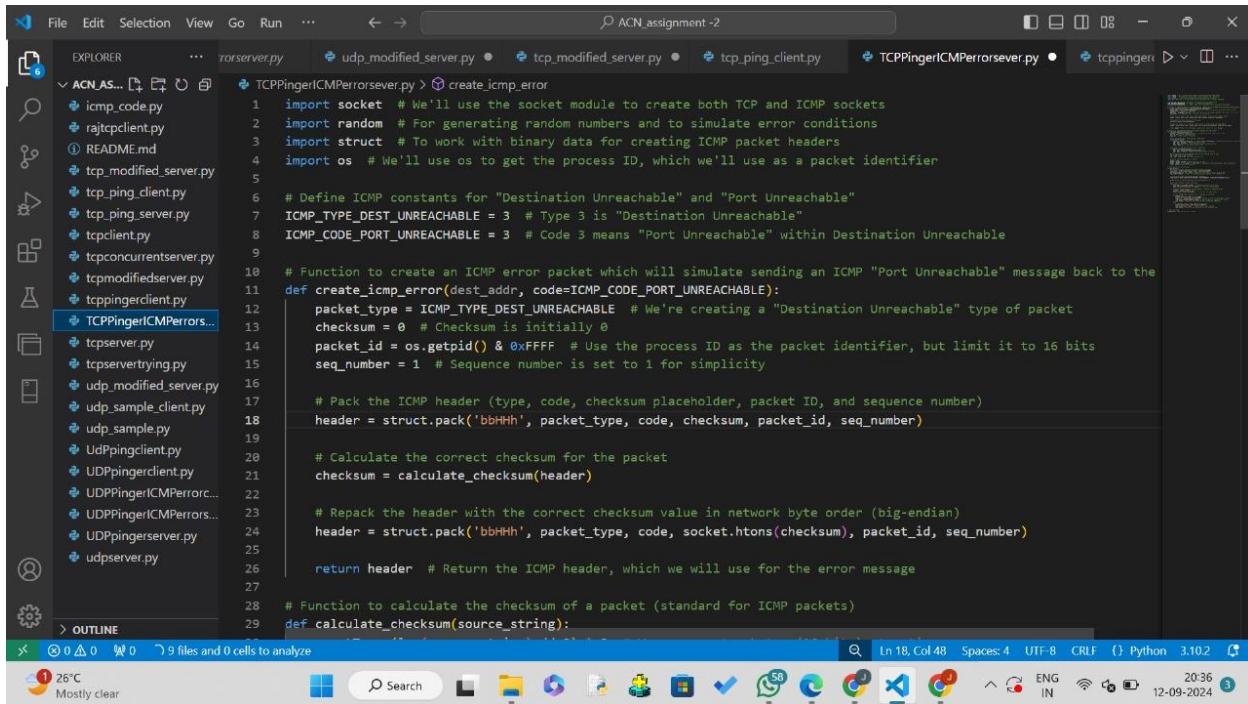
3. TCP Socket:

- We are creating TCP socket by AF_INET which is the IPv4. Client.connect() function used to connect to the all-available networks and the port number 12000.

4. ICMP error :

- We have also created a raw socket for sending icmp error message.
- create_icmp_error this function is used to generate a icmp error message.
- It is constructed with checksum and if randomly generated is greater than 8 then it will show the icmp error messages.
- If no error is generated then we will get the normal response.

The server code is as shown below:



The screenshot shows the PyCharm IDE interface with the following details:

- File Structure (EXPLORER):** The project tree shows several files under the "ACN_ASSIGNMENT-2" folder, including:
 - icmp_code.py
 - rajtcpclient.py
 - README.md
 - tcp_modified_server.py
 - tcp_ping_client.py
 - tcp_ping_server.py
 - tcpclient.py
 - tcpconcurrentserver.py
 - tcpmodifiedserver.py
 - TCPPingICMPerrorserver.py (selected)
 - tcpserver.py
 - tcpservertry.py
 - udp_modified_server.py
 - udp_sample_client.py
 - udp_sample.py
 - UDPPingclient.py
 - UDPPingerclient.py
 - UDPPingerICMPerror...
 - UDPPingerserver.py
 - udpserver.py
- Code Editor (TCPPingICMPerrorserver.py):** The code implements ICMP error simulation. Key parts include:
 - Imports: socket, random, struct, os.
 - ICMP constants: ICMP_TYPE_DEST_UNREACHABLE (3), ICMP_CODE_PORT_UNREACHABLE (3).
 - A function `create_icmp_error(dest_addr, code=ICMP_CODE_PORT_UNREACHABLE):` which constructs an ICMP error packet with type 3, code 3, and a process ID-based packet ID.
 - A helper function `calculate_checksum(header)` to calculate the checksum.
 - The function returns the ICMP header.
- Bottom Status Bar:** Shows the current file (TCPPingICMPerrorserver.py), line (Line 18, Col 48), spaces (Spaces 4), encoding (UTF-8), Python version (3.10.2), and date/time (12-09-2024 20:36).

The screenshot shows a dual-monitor setup. The left monitor displays a code editor with Python code for a UDP modified server. The right monitor displays another code editor with Python code for a TCP ping client. Both editors have toolbars, status bars, and code completion features.

Left Monitor (ACN_assignment - 2)

```
File Edit Selection View Go Run ... ACN_assignment - 2
EXPLORER rorserver.py udp_modified_server.py tcp.modified_server.py TCPingerICMPerrorserver.py tcppingclient.py
ACN_AS... icmp_code.py raftcpclient.py README.md TCPingerICMPerrorserver.py
tcp.modified_server.py tcping_client.py tcping_server.py tcpingclient.py TCPingerICMPerrorserver.py
tcpingclient.py tcpcurrentserver.py tcpmmodifiedserver.py tcppingerclient.py
tcppingerclient.py TCPingerICMPerrorserver.py
tcpserver.py tcpserving.py udp.modified_server.py udp.sample_client.py UDPingerICMPerrorserver.py
udp.sample.py UDPingerclient.py UDPingerclient.py UDPingerICMPerrorserver.py
udpserver.py

TCPingerICMPerrorserver.py
29 def calculate_checksum(source_string):
30     countTo = (len(source_string) // 2) * 2 # We process two bytes (16 bits) at a time
31     sum = 0 # Initialize the checksum sum
32     count = 0 # Start from the first byte
33
34     # Loop through the data, summing up all 16-bit words
35     while count < countTo:
36         this_val = source_string[count + 1] * 256 + source_string[count] # Combine two bytes into a 16-bit word
37         sum = sum + this_val # Add the value to the sum
38         sum = sum & 0xffffffff # Keep it within 32 bits to avoid overflow
39         count = count + 2 # Move to the next two bytes
40
41     # If there's an odd byte left over, add it separately
42     if countTo < len(source_string):
43         sum = sum + source_string[len(source_string) - 1]
44         sum = sum & 0xffffffff # Again, ensure the sum doesn't overflow
45
46     # Fold the carry bits (from the upper 16 bits) back into the lower 16 bits
47     sum = (sum >> 16) + (sum & 0xffff)
48     sum = sum + (sum >> 16) # Add any remaining carry
49
50     # Perform the one's complement of the sum
51     answer = ~sum
52     answer = answer & 0xffff # Ensure it fits in 16 bits
53     answer = answer >> 8 | (answer << 8 & 0xff00) # Swap bytes to match network byte order (big-endian)
54
55     return answer # Return the checksum
56
57 # Main function for the TCP server
58 def tcp_server():
59
```

Right Monitor (ACN_assignment - 2)

```
File Edit Selection View Go Run ... ACN_assignment - 2
EXPLORER udp.modified_server.py tcp.modified_server.py TCPingerICMPerrorserver.py TCPpingICMPerrorclient.py tcpping_server.py icmp_ci
ACN_ASSIGNMENT - 2 TCPpingICMPerrorclient.py
icmp_code.py raftcpclient.py README.md TCPpingICMPerrorclient.py
tcp.modified_server.py tcping_client.py tcping_server.py tcpingclient.py TCPpingICMPerrorclient.py
tcpingclient.py tcpcurrentserver.py tcpmmodifiedserver.py tcppingerclient.py
tcppingerclient.py TCPpingICMPerrorclient.py
tcpserver.py tcpserving.py udp.modified_server.py udp.sample_client.py UDPpingICMPerrorclient.py
udp.sample.py UDPpingclient.py UDPpingclient.py UDPpingICMPerrorclient.py
udpserver.py

TCPpingICMPerrorclient.py
72 def tcp_client(server_ip, num_requests=5, timeout=5):
73     # Create a raw socket for ICMP (to capture error messages)
74     icmp_socket = socket.socket(socket.AF_INET, socket.SOCK_RAW, socket.getprotobynumber("icmp"))
75     # Loop to send multiple TCP requests
76     for i in range(num_requests):
77         try:
78             print(f"Sending TCP request {i+1}/{num_requests} to {server_ip}")
79             # Create a TCP socket
80             tcp_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
81             tcp_socket.settimeout(timeout) # Set a timeout for the TCP connection
82             try:
83                 tcp_socket.connect((server_ip, 12001)) # Try to connect to the server on port 12001
84                 response = tcp_socket.recv(1024) # Receive response data from the server
85                 print(f"Received response from server: {response.decode()}" # Print the server response
86
87             # Handle timeout if the server doesn't respond within the timeout period
88             except socket.timeout:
89                 print("TCP connection timed out")
90             # Handle connection refusal (if the server is not listening on the port)
91             except ConnectionRefusedError:
92                 print("TCP connection refused.")
93             finally:
94                 tcp_socket.close() # Close the TCP socket after the request
95
96             # Check for ICMP error messages (like "Destination Unreachable")
97             icmp_error = receive_icmp_error(icmp_socket, timeout)
98             if "Error" in icmp_error:
99                 print(icmp_error) # Print any ICMP error messages if detected
100            # Catch any other unexpected errors and print the exception
101            except Exception as e:
102                print(f"An error occurred: {e}")
103                time.sleep(1) # Wait for 1 second before sending the next request
104
105            icmp_socket.close() # Close the ICMP socket after all requests are done
106        # Run the TCP client
107        tcp_client("127.0.0.1", num_requests=30) # Call the client function with 30 requests to the localhost
108
```

The client code is as shown below:

File Edit Selection View Go ... ← → ⌂ ACN_assignment -2

EXPLORER modified_server.py • icmp_client.py • TCPPingerICMPerrorserver.py • TCPpingrICMPerrorclient.py • UDPpingrICMPerrorclient.py • tcp_ping_server.py • icmp_cc

```

ACN_AS_...:
    + UDPpingrICMPerrorclient.py > checksum
        1 import socket # We'll need the socket module for network communication
        2 import os # used for accessing OS-level functions like process ID
        3 import time # We'll use the time module to handle timeouts and delays
        4 import struct # Needed for packing and unpacking binary data (like ICMP headers)
        5 import select # For monitoring socket readiness with a timeout
        6
        7 # ICMP constants: type 8 is for Echo Request (i.e., a ping request)
        8 ICMP_ECHO_REQUEST = 8
        9
        10 # Mapping ICMP "Destination Unreachable" codes to human-readable messages
        11 ICMP_CODE_UNREACHABLE = {
        12     0: "Destination Network Unreachable",
        13     1: "Destination Host Unreachable",
        14     3: "Port Unreachable",
        15 }
        16
        17 # Function to calculate the checksum for the ICMP packet (essentially a way to ensure data integrity)
        18 def checksum(source_string):
        19     sum = 0 # This will accumulate the sum of the data
        20     # We'll process two bytes at a time, so we calculate how many full 2-byte chunks there are
        21     countTo = (len(source_string) // 2) * 2
        22     count = 0
        23
        24     # Loop through the string two bytes at a time
        25     while count < countTo:
        26         # Convert the two bytes into a single 16-bit value (big-endian format)
        27         thisVal = source_string[count + 1] * 256 + source_string[count]
        28         sum = sum + thisVal # Add the value to the running sum
        29         sum = sum & 0xffffffff # Make sure we don't overflow beyond 32 bits
        30         count = count + 2 # Move to the next two bytes
        31
        32     # If the length of the string is odd, we have one last byte to process
        33     if countTo < len(source_string):
        34         sum = sum + source_string[len(source_string) - 1]
        35         sum = sum & 0xffffffff # Again, ensure we don't exceed 32 bits
        36
        37     # Fold the sum into 16 bits by adding the upper 16 bits to the lower 16 bits
        38     sum = (sum >> 16) + (sum & 0xffff)
        39     sum = sum + (sum >> 16) # Add any remaining carry
        40     answer = ~sum # Perform a bitwise NOT to get the checksum
        41     answer = answer & 0xffff # Mask to 16 bits
        42     # Swap bytes (because network order is big-endian)
        43     answer = answer >> 8 | (answer << 8 & 0xff00)
        44     return answer # Return the calculated checksum
        45
        46 # Function to receive ICMP error messages (e.g., Destination Unreachable)
        47 def receive_icmp_error(my_socket, timeout):
        48     time_left = timeout # Start with the full timeout value
        49     while True:
        50         start_time = time.time() # Record the start time of this cycle
        51         ready = select.select([my_socket], [], [], time_left) # Wait for the socket to be ready or timeout
        52         time_in_select = (time.time() - start_time) # Measure how long select() took
        53
        54         if ready[0] == []: # If no socket is ready (timeout expired)
        55             return "Request timed out."
        56
        57         time_received = time.time() # Time when the packet is received
        58         rec_packet, addr = my_socket.recvfrom(1024) # Read the incoming packet (max 1024 bytes)
        59         icmp_header = rec_packet[20:28] # Extract the ICMP header from the packet
        60         icmp_type, code, checksum, packet_id, sequence = struct.unpack("BHBB", icmp_header) # Unpack it
        61
        62         # If the ICMP message is of type 3 (Destination Unreachable)
        63         if icmp_type == 3:
        64             if code in ICMP_CODE_UNREACHABLE:
        65                 # Return the appropriate human-readable error message
        66                 return f"Error: {ICMP_CODE_UNREACHABLE[code]}"
        67             else:
        68                 # If we don't know the code, return a generic error message
        69                 return f"Error: Unknown ICMP error code {code}"
        70
        71     # Subtract the time spent in select() from the remaining time

```

File Edit Selection View Go ... ← → ⌂ ACN_assignment -2

EXPLORER modified_server.py • icmp_client.py • TCPPingerICMPerrorserver.py • TCPpingrICMPerrorclient.py • UDPpingrICMPerrorclient.py • tcp_ping_server.py • icmp_cc

```

ACN_AS_...:
    + UDPpingrICMPerrorclient.py > checksum
        18 def checksum(source_string):
        37     sum = (sum >> 16) + (sum & 0xffff)
        38     sum = sum + (sum >> 16) # Add any remaining carry
        39     answer = ~sum # Perform a bitwise NOT to get the checksum
        40     answer = answer & 0xffff # Mask to 16 bits
        41     # Swap bytes (because network order is big-endian)
        42     answer = answer >> 8 | (answer << 8 & 0xff00)
        43     return answer # Return the calculated checksum
        44
        45 # Function to receive ICMP error messages (e.g., Destination Unreachable)
        46 def receive_icmp_error(my_socket, timeout):
        47     time_left = timeout # Start with the full timeout value
        48     while True:
        49         start_time = time.time() # Record the start time of this cycle
        50         ready = select.select([my_socket], [], [], time_left) # Wait for the socket to be ready or timeout
        51         time_in_select = (time.time() - start_time) # Measure how long select() took
        52
        53         if ready[0] == []: # If no socket is ready (timeout expired)
        54             return "Request timed out."
        55
        56         time_received = time.time() # Time when the packet is received
        57         rec_packet, addr = my_socket.recvfrom(1024) # Read the incoming packet (max 1024 bytes)
        58         icmp_header = rec_packet[20:28] # Extract the ICMP header from the packet
        59         icmp_type, code, checksum, packet_id, sequence = struct.unpack("BHBB", icmp_header) # Unpack it
        60
        61         # If the ICMP message is of type 3 (Destination Unreachable)
        62         if icmp_type == 3:
        63             if code in ICMP_CODE_UNREACHABLE:
        64                 # Return the appropriate human-readable error message
        65                 return f"Error: {ICMP_CODE_UNREACHABLE[code]}"
        66             else:
        67                 # If we don't know the code, return a generic error message
        68                 return f"Error: Unknown ICMP error code {code}"
        69
        70     # Subtract the time spent in select() from the remaining time

```

File Edit Selection View Go ... ← → ⌂ ACN_assignment -2

EXPLORER modified_server.py • icmp_client.py • TCPPingerICMPerrorserver.py • TCPpingrICMPerrorclient.py • UDPpingrICMPerrorclient.py • tcp_ping_server.py • icmp_cc

```

ACN_AS_...:
    + UDPpingrICMPerrorclient.py > checksum
        18 def checksum(source_string):
        37     sum = (sum >> 16) + (sum & 0xffff)
        38     sum = sum + (sum >> 16) # Add any remaining carry
        39     answer = ~sum # Perform a bitwise NOT to get the checksum
        40     answer = answer & 0xffff # Mask to 16 bits
        41     # Swap bytes (because network order is big-endian)
        42     answer = answer >> 8 | (answer << 8 & 0xff00)
        43     return answer # Return the calculated checksum
        44
        45 # Function to receive ICMP error messages (e.g., Destination Unreachable)
        46 def receive_icmp_error(my_socket, timeout):
        47     time_left = timeout # Start with the full timeout value
        48     while True:
        49         start_time = time.time() # Record the start time of this cycle
        50         ready = select.select([my_socket], [], [], time_left) # Wait for the socket to be ready or timeout
        51         time_in_select = (time.time() - start_time) # Measure how long select() took
        52
        53         if ready[0] == []: # If no socket is ready (timeout expired)
        54             return "Request timed out."
        55
        56         time_received = time.time() # Time when the packet is received
        57         rec_packet, addr = my_socket.recvfrom(1024) # Read the incoming packet (max 1024 bytes)
        58         icmp_header = rec_packet[20:28] # Extract the ICMP header from the packet
        59         icmp_type, code, checksum, packet_id, sequence = struct.unpack("BHBB", icmp_header) # Unpack it
        60
        61         # If the ICMP message is of type 3 (Destination Unreachable)
        62         if icmp_type == 3:
        63             if code in ICMP_CODE_UNREACHABLE:
        64                 # Return the appropriate human-readable error message
        65                 return f"Error: {ICMP_CODE_UNREACHABLE[code]}"
        66             else:
        67                 # If we don't know the code, return a generic error message
        68                 return f"Error: Unknown ICMP error code {code}"
        69
        70     # Subtract the time spent in select() from the remaining time

```

The screenshot shows the Visual Studio Code interface with the following details:

- Title Bar:** ACN_assignment - 2
- Explorer:** Shows the project structure for 'ACN_ASSIGNMENT - 2' containing files like modified_server.py, icmp_code.py, README.md, and various TCP and UDP ping client/server files.
- Code Editor:** Displays the content of `UDPPingerICMPerrorclient.py`. The code implements a UDP ping client that sends ICMP error messages to a server at port 12000. It includes logic for sending pings, receiving ICMP errors, and handling socket timeouts.
- Bottom Status Bar:** Shows file analysis (104 files, 0 cells), system icons (Windows Start, Search, Task View, File Explorer, etc.), and system status (2045, 12-09-2024).

The Response viewed on client side is as follows:

The screenshot shows the Visual Studio Code interface with the following details:

- Title Bar:** TCPPingerICMPerrorclient.py - Documents - Visual Studio Code
- Terminal:** Shows the command `sudo python3 TCPPingerICMPerrorclient.py` being run and its output. The output indicates multiple TCP requests being sent to 192.168.195.217, with some responses and one error message: "Error: Port Unreachable".

```

khushbu@khushbu-VirtualBox:~/Documents$ sudo python3 TCPPingerICMPerrorclient.py
Sending TCP request 1/30 to 192.168.195.217
Received response from server:
Sending TCP request 2/30 to 192.168.195.217
Received response from server:
Sending TCP request 3/30 to 192.168.195.217
TCP connection timed out.
Sending TCP request 4/30 to 192.168.195.217
Received response from server:
Sending TCP request 5/30 to 192.168.195.217
TCP connection timed out.
Sending TCP request 6/30 to 192.168.195.217
Received response from server:
Sending TCP request 7/30 to 192.168.195.217
Received response from server:
Sending TCP request 8/30 to 192.168.195.217
Received response from server:
Sending TCP request 9/30 to 192.168.195.217
TCP connection timed out.
Error: Port Unreachable
Sending TCP request 10/30 to 192.168.195.217
Received response from server:
Sending TCP request 11/30 to 192.168.195.217
Received response from server:
Sending TCP request 12/30 to 192.168.195.217
TCP connection timed out.
Sending TCP request 13/30 to 192.168.195.217
Received response from server:
Sending TCP request 14/30 to 192.168.195.217
Received response from server:
Sending TCP request 15/30 to 192.168.195.217
Received response from server:
Sending TCP request 16/30 to 192.168.195.217

```

Here, we are getting error message which is port unreachable or the packets which random number is greater than 8.

The Response viewed on server side is as follows:

```
[manikanta@Mani]-(~/Documents/Python Projects/Assignment1]
$ sudo iptables -A INPUT -s 192.168.195.110 -p icmp -j REJECT --reject-with icmp-host-unreachable
[manikanta@Mani]-(~/Documents/Python Projects/Assignment1]
$ sudo iptables -F
```

ICMP Pinger:

Task at hand is to handle the sending ICMP echo requests and getting responses so as to calculate the stats of the RTT (min, max, avg) and handling packet loss and detecting errors.

Skeleton code explanation:

Checksum calculation:

Checksum () Function calculates the checksum of the data ensuring the credibility of the packet and rendering it is a valid packet or not.

SendOnePing () Function:

Uses a raw socket to send a data+header packet to a destination address, it constructs ICMP echo request packet also before sending it.

receiveOnePing () Function:

waits to receive ICMP echo response from destination address and extracts critical information from that packet like TTL, sequence number etc,

doOnePing () Function:

Sending and receiving pings as the key functionality for calculation of RTT stats

Ping () Function:

This is the main function on loop that runs and sends pings with a countable time interval gap like a standard ping utility and prints results.

Completing receiveOnePing() Function:

we extract the ICMP header from the IP packet by string slicing after the IP header and unpacking the required details of the packet like type, code, checksum, packetID, sequence number. And only when the packet sequence is matching the sent request, and type = 0 (typically for echo reply). And then calculate the RTT from the sent and received time difference.

We've handled the error scenarios of ICMP (i.e Type = 3) and returned the respective error messages as required, according to the error codes ranging from 0,1,2,3.

The below are the prompt responses given by the LLM

ChatGPT ▾

- Extract the ICMP Header from the received packet.
- Check for the correct ICMP Type (Echo Reply) and ID to match the Echo Request.
- Calculate RTT (Round Trip Time).

The code provided for `receiveOnePing()` is mostly complete, but ensure that you handle errors and extract ICMP headers correctly:

```
python Copy code  
def receiveOnePing(mySocket, ID, timeout, destAddr):  
    timeLeft = timeout  
    while True:  
        startedSelect = time.time()  
        whatReady = select.select([mySocket], [], [], timeLeft)  
        howLongInSelect = time.time() - startedSelect  
        if whatReady[0] == []:  
            return "Request timed out."  
        timeReceived = time.time()  
        recPacket, addr = mySocket.recvfrom(1024)  
  
        # Extract ICMP header from IP packet  
        icmpHeader = recPacket[20:28] # IP header is 20 bytes  
        type, code, checksum, packetID, sequence = struct.unpack("bbHHh", icmpHeader)  
  
        if type == 0 and packetID == ID: # Type 0 is Echo Reply  
            bytesInDouble = struct.calcsize("d")  
            timeSent = struct.unpack("d", recPacket[28:28 + bytesInDouble])[0]  
            rtt = timeReceived - timeSent  
            return rtt  
        else:  
            return f"ICMP error: Type {type}, Code {code}"
```

4. Add Error Handling for ICMP Responses

Modify the `receiveOnePing()` function to handle common ICMP error types, such as "Destination Unreachable" or "TTL Expired." The basic structure for error detection is already in place:

```
python Copy code  
if type == 3: # Destination Unreachable  
    if code == 0:  
        return "Destination Network Unreachable"  
    elif code == 1:  
        return "Destination Host Unreachable"  
    # Add more codes as needed  
elif type == 11: # TTL Expired  
    return "Time to Live exceeded"
```

Modifying Ping function:

The ping() function is modified to report the Maximum, Minimum and Average times of RTT for the number of selected pings(4 in our case). Also, we calculate the Packet loss rate from the above snippet of code.

5. Complete `ping()` Function for RTT Statistics and Packet Loss

At the end of the pings, you need to print RTT statistics and calculate packet loss rate. The `ping()` function is almost done, just ensure this block of code is correct:

```
python Copy code

def ping(host, timeout=1, count=4):
    dest = gethostbyname(host)
    print(f"Pinging {dest} using Python:")

    rttList = []
    packetsSent = 0
    packetsReceived = 0

    for i in range(count):
        delay = doOnePing(dest, timeout)
        packetsSent += 1
        if isinstance(delay, float): # Valid response
            rttList.append(delay)
            packetsReceived += 1
            print(f"Reply from {dest}: time={delay:.6f}s")
        else:
            print(delay) # Handle error messages

        time.sleep(1)

    # Calculate statistics
    if rttList:
        minRtt = min(rttList)
        maxRtt = max(rttList)
        avgRtt = sum(rttList) / len(rttList)
        packetLoss = ((packetsSent - packetsReceived) / packetsSent) * 100

        print("\n--- Ping statistics ---")
        print(f"Packets: Sent = {packetsSent}, Received = {packetsReceived}, Lost = {packetsReceived - packetsSent}")
        print(f"RTT (Round Trip Times): Min = {minRtt:.6f}s, Max = {maxRtt:.6f}s, Avg = {avgRtt:.6f}s")
    else:
        print("No response received.") ↓
```

Finally:

The ping function has to be called with given the host address, or with it's DNS lookup name inside the main function.

Changes Made in receiveOnePing() function referred from ChatGPT is as follows:

```
35     def receiveOnePing(mySocket, ID, timeout, destAddr):
36         timeLeft = timeout
37
38         while 1:
39             startedSelect = time.time()
40             whatReady = select.select([mySocket], [], [], timeLeft)
41             howLongInSelect = (time.time() - startedSelect)
42             if whatReady[0] == []: # Timeout
43                 return "Request timed out."
44
45             timeReceived = time.time()
46             recPacket, addr = mySocket.recvfrom(1024)
47
48             # Fetch the ICMP header from the IP packet (since, ICMP header starts after 20 bytes and extends 8 bytes)
49             icmpHeader = recPacket[20:28]
50             # unpacking in the format "bbHHh" corresponds to each of the parameter's bit size respectively.
51             type, code, checksum, packetID, sequence = struct.unpack("bbHHh", icmpHeader)
52
53             if type == 0 and packetID == ID: # Echo Reply (type 0)
54                 bytesInDouble = struct.calcsize("d")
55                 timeSent = struct.unpack("d", recPacket[28 : 28+bytesInDouble])[0]
56                 rtt = (timeReceived - timeSent) * 1000 # conversion into milli sec
57                 return f"Response from {destAddr}: bytes-{len(recPacket)} time-{rtt:.2f}ms TTL={recPacket[8]}"
58
59             # type 3 corresponds to the error and the given are corresponding error messages
60             elif type == 3: # Destination can't be reached
61                 if code == 0:
62                     return f"Destination Network Unreachable"
63                 elif code == 1:
64                     return f"Destination Host Unreachable"
65                 elif code == 2:
66                     return f"Destination Protocol Unreachable"
67                 elif code == 3:
68                     return f"Destination Port Unreachable"
69                 else:
70                     return f"Destination Unreachable (code {code})"
71
72
73             timeLeft = timeLeft - howLongInSelect
74             if timeLeft <= 0:
75                 return "Request timed out."
```

Changes made in Ping() function is as follows:

```

110  def ping(host, timeout=1, count=4):
111      dest = gethostbyname(host)
112      print(f"Pinging {dest} using Python:")
113      print("")
114
115      # Statistics variables
116      min_rtt = float('inf')
117      max_rtt = float('-inf')
118      total_rtt = 0
119      packet_loss = 0
120
121      # Send ping requests to a server separated by approximately one second
122      for i in range(count):
123          delay = doOnePing(dest, timeout)
124          print(delay)
125
126          # Calculate minimum, max, and average RTT in ms
127          if "Request timed out" in delay:
128              packet_loss += 1
129          else:
130              time_value = float(delay.split("time=")[-1].split("ms")[0])
131              total_rtt += time_value
132              min_rtt = min(min_rtt, time_value)
133              max_rtt = max(max_rtt, time_value)
134
135          time.sleep(1) # wait for one second between pings for let and calculations
136
137          # Calculate statistics
138          successful_pings = count - packet_loss
139          if successful_pings > 0:
140              avg_rtt = total_rtt / successful_pings
141          else:
142              avg_rtt = 0
143
144          packet_loss_rate = (packet_loss / count) * 100
145
146          print(f"\n-- {host} ping statistics --")
147          print(f"{count} packets transmitted, {successful_pings} received, {packet_loss_rate:.2f}% packet loss \n")
148          if successful_pings > 0:
149              print_("Approximate round trip times in milli-seconds:")
150              print_(f"Minimum : {min_rtt:.2f}ms, Maximum: {max_rtt:.2f}ms, Average {avg_rtt:.2f}ms")
151
152
153 ➤ if __name__ == "__main__":
154     # Calling Ping function with the host name
155     ping("google.com")

```

Final output is as follows:

```
Pinging 192.168.195.217 using Python:
```

```
Response from 192.168.195.217: bytes=36 time=87.47ms TTL=64
Response from 192.168.195.217: bytes=36 time=265.50ms TTL=64
Response from 192.168.195.217: bytes=36 time=249.40ms TTL=64
Response from 192.168.195.217: bytes=36 time=255.12ms TTL=64
Response from 192.168.195.217: bytes=36 time=113.69ms TTL=64
Response from 192.168.195.217: bytes=36 time=53.89ms TTL=64
Response from 192.168.195.217: bytes=36 time=207.12ms TTL=64
Response from 192.168.195.217: bytes=36 time=312.58ms TTL=64
Request timed out.
Response from 192.168.195.217: bytes=36 time=180.89ms TTL=64
Response from 192.168.195.217: bytes=36 time=19.35ms TTL=64
Response from 192.168.195.217: bytes=36 time=40.06ms TTL=64
Response from 192.168.195.217: bytes=36 time=306.38ms TTL=64
Request timed out.
Response from 192.168.195.217: bytes=36 time=276.53ms TTL=64
Response from 192.168.195.217: bytes=36 time=359.62ms TTL=64
Response from 192.168.195.217: bytes=36 time=6.98ms TTL=64
Response from 192.168.195.217: bytes=36 time=45.45ms TTL=64
Response from 192.168.195.217: bytes=36 time=303.05ms TTL=64
Response from 192.168.195.217: bytes=36 time=222.75ms TTL=64

--- 192.168.195.217 ping statistics ---
20 packets transmitted, 18 received, 10.00% packet loss

Approximate round trip times in milli-seconds:
Minimum : 6.98ms, Maximum: 359.62ms, Average 183.66ms
```

YEDLA JAGADISH KUMAR:

Task/Section	Contribution (describe the work done, in brief)	Challenges faced (if any).
Research & Info gathering	Gathered info about handling packet loss, error handling, checksum calculation and timeout mechanisms in UDP communication, and researched about ICMP error codes And about udp client	Understanding how to handle packet loss in connectionless protocol like UDP. Calculation of accurate checksum and RTTs
Code Development	Developed the UDP client to send requests and wait for server responses Measuring RTTs statistics, Checksum calculation, Implemented the ICMP error client and server according to their error codes.	Handling raw sockets and ICMP error code message formatting.
Testing & Debugging	Tested the client by simulating various network conditions of interfaces like “lo” and “enp0s3” for injecting packet loss at NIC level,	Debugging timeout issues like that client no needed to wait for a longer time for response.
Documentation & Report Writing	Documented the UDP client implementation, explaining the role of timeout handling and packet retransmission.	
Final Review Submission	Reviewed the client code to ensure it meets project requirements and functions as intended. Submitted the code and documentation after final testing.	

CHAUDHARY KHUSHBU :

Task/Section	Contribution (describe the work done, in brief)	Challenges faced (if any).
Research & Info gathering	Researched TCP server and client socket , multi-threading and icmp error message code. Learn how to use the multi-client request on tcp socket.	For research I took help from the LLM source like chatgpt.
Code Development	Developed TCP client for sending request to the tcp server and generate the response from tcp server to tcp client, also measure the rtt value for each packet and print the minimum rtt, maximum rtt, average rtt. For multiple client handling in tcp developed a multi-threading socket code to handle multiple requests at same time. Also developed code for icmp error messages like port unreachable, destination unreachable.	For the icmp error message it is working in virtual box for local host but it is not working in the virtual box when we are trying for different machines. When we connected to mobile data pakcet loss is not happening so we used the netem command and iptable command for packet loss.
Testing & Debugging	Tested the code on local host as well as on different machines also.	For icmp error message in virtual box we did not get the error message like port unreachable, destination unreachable.
Documentation & Report Writing	Documented the TCP server, client, multi-threading, icmp error message and put the screenshot of code an output of the tcp code.	
Final Review Submission	Review the code before the final submission and	

Venkata Manikanta Sai Amalakanti :

Task/Section	Contribution (describe the work done, in brief)	Challenges faced (if any).
Research Info gathering	For ICMP code pinger program, I gathered research articles and some help to know how it works in reality and how we are simulating using a Python program. Also researched on how to simulate error scenario using some linux commands. Studied Python libraries for networking and packet manipulation, including socket, struct, and select.	The snippets have to be individually understood in order to make it work for DNS lookup servers and the foreign machines of our team, to make necessary changes in the code for each scenario respectively.
Code Development	ICMP pinger includes the methods to Ping the server whose IP address is given in the parameter of the Ping () method, the packet is prepared by integrating the respective header and data required to construct it. And is sent using the socket that is created for the communication. main tasks at hand include Checksum calculation, packet creation, pinging the host, stats calculation of the time parameters and packet loss in %.	Calculating the time stats and handling error scenarios are a big task when it came to implementation, but with proper debugging and resetting the string slice lengths, it worked as expected.
Testing & Debugging	Checked for multiple network scenarios and multiple hosts that the client can gather the stats and packet loss information as expected from start to the end of the pinging for required many numbers of times.	
Documentation & Report Writing	Documented how the ping function worked and the remaining methods that are integrated for various functionalities like checksum calculation, stats calculation.	
Final Review Submission	Submitted the final version of the code along with the documentation and report, ensuring that all aspects of the project are complete	

ANTI-PLAGIARISM Statement

We certify that this assignment/report is the result of our collaborative work, based on our collective study and research. All sources, including books, articles, software, datasets, reports, and communications, have been properly acknowledged. This work has not been previously submitted for assessment in any other

course unless specific permission was granted by all involved instructors. We also acknowledge the use of AI tools, such as LLMs (e.g., ChatGPT), for assistance in refining this assignment, if used. We have ensured that their usage complies with the academic integrity policies of this course. We pledge to uphold the principles of honesty, integrity, and responsibility at CSE@IITH. Additionally, we understand our duty to report any violations of academic integrity by others if we become aware of them.

Yelda Jagadish Kumar - cs24mtech11022

Chaudhary Khushbu - cs24mtech14012

Venkata Manikanta Sai Amalakanti - sm24mtech14006

Date: 12/09/2024