***NOTE**: Instructions on how to run the code are primarily for the **macOS/Linux** operating systems. Other OSes, such as **Windows**, might slightly differ but will be similar.*

Part 1 – UDP Pinger with No Delay and No Loss

1. My UDP Pinger enables two network entities, the client and the corresponding server, to exchange the same, unmodified message with each other (i.e., point-to-point communication) using the UDP protocol. The client program sends a specified number of pings to the communicating server with the specified message format. Upon receiving a message from the server, the client outputs the server's response message, followed by the respective RTT. Also, the client accumulates a list of RTTs for each successful ping for later statistics. The client will time out if it has not received a response from the server within 1 second, which it will notify with a timed-out message. The server simply receives the client's message and sends the exact same message back to the client. Lastly, my UDP Pinger shows statistics (e.g., central tendency) of the RTTs and the percentage of packet loss.
2. The timeout value is specified for a datagram socket as "**socket.settimeout([num])**" where "**[num]**" is a numerical value in seconds. The **settimeout** function is defined in the **socket** Python standard library. Thus, the socket library must be imported in some manner (e.g., **import socket**).
    a. For example, suppose a server needs to define a timeout value for its application to mitigate starvation; one may (in Python) write "**serverSocket.settimeout(30)**" enabling the server process to resume operation after 30 seconds of delay. In essence, we have some mechanisms to mitigate **blocking communication** between the client and server.
3. How to run the code
    a. To run the code, first, run the server program as a script from the command line as "**python3 udppingserver_no_loss.py**".
    b. Now, open a separate local terminal and run the client program as a script from the command line as "**python3 jd1.py**".

```
duong-2:Desktop duong-jason$ python3 udppingserver_no_loss.py
```

```
duong-2:Desktop duong-jason$ python3 jd1.py
Jason 1: server reply: Jason 1 Sat Feb 25 17:32:17 2023, RTT = 2.54 ms
Jason 2: server reply: Jason 2 Sat Feb 25 17:32:17 2023, RTT = 0.43 ms
Jason 3: server reply: Jason 3 Sat Feb 25 17:32:17 2023, RTT = 0.39 ms
Jason 4: server reply: Jason 4 Sat Feb 25 17:32:17 2023, RTT = 0.43 ms
Jason 5: server reply: Jason 5 Sat Feb 25 17:32:17 2023, RTT = 0.37 ms
Jason 6: server reply: Jason 6 Sat Feb 25 17:32:17 2023, RTT = 0.38 ms
Jason 7: server reply: Jason 7 Sat Feb 25 17:32:17 2023, RTT = 0.36 ms
Jason 8: server reply: Jason 8 Sat Feb 25 17:32:17 2023, RTT = 0.36 ms
Jason 9: server reply: Jason 9 Sat Feb 25 17:32:17 2023, RTT = 0.41 ms
Jason 10: server reply: Jason 10 Sat Feb 25 17:32:17 2023, RTT = 0.37 ms
Min RTT = 0.36 ms
Max RTT = 2.54 ms
Avg RTT = 0.60 ms
Packet lost = 0.00%
duong-2:Desktop duong-jason$
```

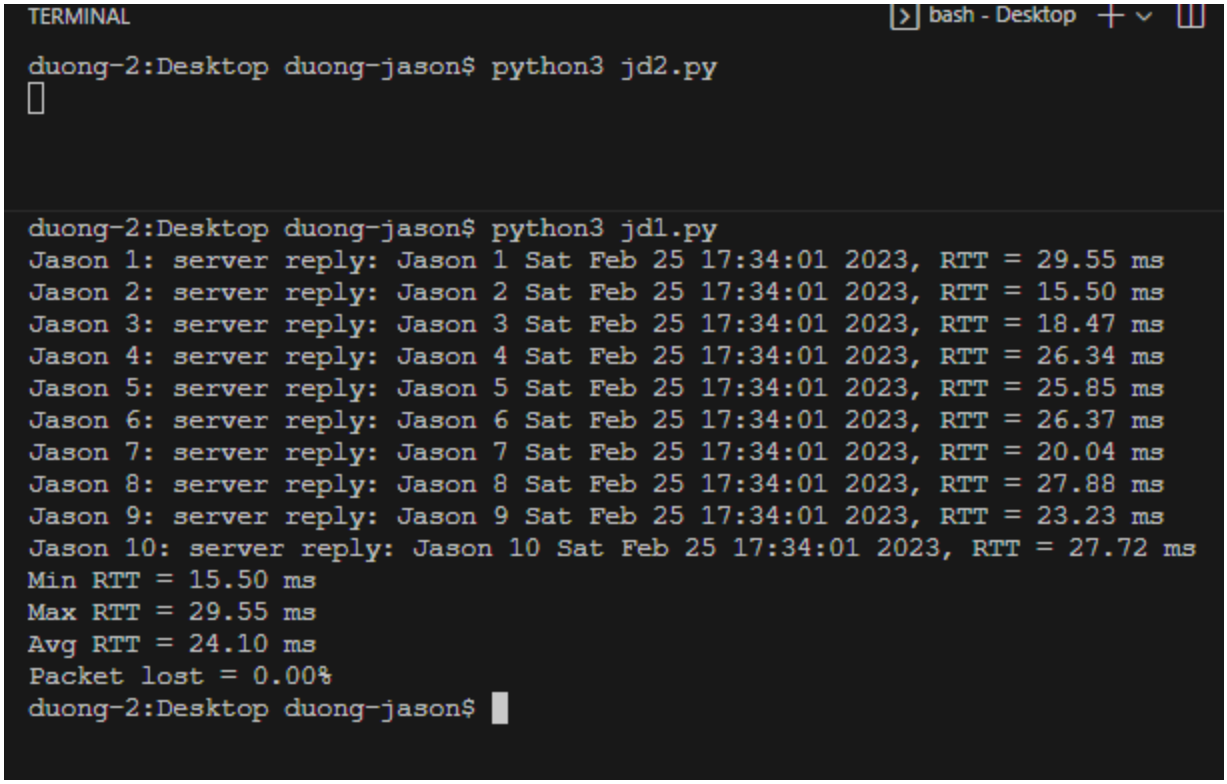
4. Python code listing

```
#!/usr/bin/env python3
# Part 1 - UDP Pinger with No Delay and No Loss

import time
from statistics import mean
from datetime import datetime
from socket import socket, timeout, AF_INET, SOCK_DGRAM

serverName = 'localhost'
serverPort = 12000

TOTAL_PINGS = 10
n_timeout = 0
rtt_list = []

for n_ping in range(1, TOTAL_PINGS+1):
    clientSocket = socket(AF_INET, SOCK_DGRAM)

    # Sets timeout value to 1 second
    clientSocket.settimeout(1)

    # Stores the time when the client sends the message
    start_time = time.time()

    # Send ping message using UDP to server
    message = f'Jason {n_ping} {datetime.now().ctime()}'
```

```
    clientSocket.sendto(message.encode(), (serverName, serverPort))

    print(f'Jason {n_ping}: ', end='')

    try:
        receivedMessage, _ = clientSocket.recvfrom(1024)

        # Calculates and converts the RTT from seconds to milliseconds
        rtt_list.append((time.time() - start_time) * 1000)

        # Prints the response message from the server
        # Retreives the latest ping's RTT (i.e., top of stack)
        print(f'server reply: {receivedMessage.decode()}, RTT = {rtt_list[-1]:.2f}
ms')
    except timeout:
        n_timeout += 1
        print('Request timed out')

    clientSocket.close()

# Summary of all pings
print(f'Min RTT = {min(rtt_list):.2f} ms',
      f'Max RTT = {max(rtt_list):.2f} ms',
      f'Avg RTT = {mean(rtt_list):.2f} ms',
      f'Packet lost = {n_timeout / TOTAL_PINGS:.2%}', sep='\n')
```

Part 2 – UDP Pinger No Loss, with Delays

1. The UDP Ping Server first waits for a request from the client on port 12000. After successfully receiving the client's message, it will abruptly wait/sleep between 10 to 20 ms before sending the same exact message back to its client.
    a. The delay simulation is done using the "**time.sleep([num])**" Python statement, where "**[num]**" is a numerical value in seconds. We select a random number between 10-20 inclusively using "**random.randint(10, 20)**". Since the random value is in seconds, we must divide the value by **1000** to convert the time to milliseconds.
2. How to run the code
    a. To run the code, first, run the server program as a script from the command line as "**python3 jd2.py**"
    b. Now, open a separate local terminal and run the client program as a script from the command line as "**python3 jd1.py**"



3. Python code listing

```
#!/usr/bin/env python3
# UDP Pinger No Loss, with Delays

import time
import random
from socket import socket, AF_INET, SOCK_DGRAM

# Create a UDP socket
serverSocket = socket(AF_INET, SOCK_DGRAM)

# Assign IP address and port number to socket
serverSocket.bind(('', 12000))

while True:
    # Receive the client packet along with the address it is coming from
    message, address = serverSocket.recvfrom(1024)

    # Creates an arbitrary, random RTT delay ranging from 10-20 milliseconds
    time.sleep(random.randint(10, 20) / 1000)

    # The server responds
    serverSocket.sendto(message, address)
```

Part 3 – UDP Pinger with Delays and Packet Losses

1. The UDP Ping Server operates identically to **Part 2** for packet loss injection**,** except before sending the same exact message back to its client, it probabilistically and artificially decides whether the packet is considered lost. In detail, the server does not send the original packet back if the randomly-generated value is within the first 10 integers of the specified 100 integers (i.e., integers 0-9). The probability of dropping a packet is specified under the **THRESHOLD** variable. This is identical to evaluating dropping a packet 10% of the time. If this happens, the control flow goes back to waiting for a response message from its client. This is simulating a packet loss as the server did not execute the send statement and continues to wait for the next ping response from its client.
2. How to run the code
    a. To run the code, first, run the server program as a script from the command line as "**python3 jd3.py**"
    b. Now, open a separate terminal and run the client program as a script from the command line as "**python3 jd1.py**"

```
duong-2:Desktop duong-jason$ python3 jd3.py
```

```
duong-2:Desktop duong-jason$ python3 jd1.py
Jason 1: server reply: Jason 1 Sun Feb 26 19:42:12 2023, RTT = 27.86 ms
Jason 2: server reply: Jason 2 Sun Feb 26 19:42:12 2023, RTT = 17.05 ms
Jason 3: server reply: Jason 3 Sun Feb 26 19:42:12 2023, RTT = 30.50 ms
Jason 4: server reply: Jason 4 Sun Feb 26 19:42:12 2023, RTT = 24.59 ms
Jason 5: server reply: Jason 5 Sun Feb 26 19:42:12 2023, RTT = 14.38 ms
Jason 6: server reply: Jason 6 Sun Feb 26 19:42:12 2023, RTT = 20.16 ms
Jason 7: server reply: Jason 7 Sun Feb 26 19:42:12 2023, RTT = 17.20 ms
Jason 8: server reply: Jason 8 Sun Feb 26 19:42:12 2023, RTT = 14.97 ms
Jason 9: server reply: Jason 9 Sun Feb 26 19:42:12 2023, RTT = 26.60 ms
Jason 10: Request timed out
Jason 11: server reply: Jason 11 Sun Feb 26 19:42:13 2023, RTT = 20.71 ms
Jason 12: server reply: Jason 12 Sun Feb 26 19:42:13 2023, RTT = 17.29 ms
Jason 13: server reply: Jason 13 Sun Feb 26 19:42:13 2023, RTT = 26.26 ms
Jason 14: server reply: Jason 14 Sun Feb 26 19:42:14 2023, RTT = 26.24 ms
Jason 15: server reply: Jason 15 Sun Feb 26 19:42:14 2023, RTT = 15.71 ms
Jason 16: server reply: Jason 16 Sun Feb 26 19:42:14 2023, RTT = 26.28 ms
Jason 17: server reply: Jason 17 Sun Feb 26 19:42:14 2023, RTT = 29.46 ms
Jason 18: server reply: Jason 18 Sun Feb 26 19:42:14 2023, RTT = 15.91 ms
Jason 19: server reply: Jason 19 Sun Feb 26 19:42:14 2023, RTT = 24.94 ms
Jason 20: server reply: Jason 20 Sun Feb 26 19:42:14 2023, RTT = 28.00 ms
Jason 21: server reply: Jason 21 Sun Feb 26 19:42:14 2023, RTT = 17.13 ms
Jason 22: server reply: Jason 22 Sun Feb 26 19:42:14 2023, RTT = 29.30 ms
Jason 23: server reply: Jason 23 Sun Feb 26 19:42:14 2023, RTT = 19.24 ms
Jason 24: server reply: Jason 24 Sun Feb 26 19:42:14 2023, RTT = 19.07 ms
Jason 25: server reply: Jason 25 Sun Feb 26 19:42:14 2023, RTT = 17.29 ms
Jason 26: server reply: Jason 26 Sun Feb 26 19:42:14 2023, RTT = 15.79 ms
Jason 27: server reply: Jason 27 Sun Feb 26 19:42:14 2023, RTT = 15.66 ms
Jason 28: server reply: Jason 28 Sun Feb 26 19:42:14 2023, RTT = 14.27 ms
Jason 29: server reply: Jason 29 Sun Feb 26 19:42:14 2023, RTT = 19.95 ms
Jason 30: server reply: Jason 30 Sun Feb 26 19:42:14 2023, RTT = 27.74 ms
Jason 31: server reply: Jason 31 Sun Feb 26 19:42:14 2023, RTT = 26.22 ms
Jason 32: Request timed out
Jason 33: server reply: Jason 33 Sun Feb 26 19:42:15 2023, RTT = 28.54 ms
Jason 34: Request timed out
Jason 35: server reply: Jason 35 Sun Feb 26 19:42:16 2023, RTT = 19.04 ms
Jason 36: server reply: Jason 36 Sun Feb 26 19:42:16 2023, RTT = 19.71 ms
Jason 37: server reply: Jason 37 Sun Feb 26 19:42:16 2023, RTT = 28.01 ms
Jason 38: server reply: Jason 38 Sun Feb 26 19:42:16 2023, RTT = 25.37 ms
Jason 39: server reply: Jason 39 Sun Feb 26 19:42:16 2023, RTT = 27.78 ms
Jason 40: server reply: Jason 40 Sun Feb 26 19:42:16 2023, RTT = 15.96 ms
Jason 41: Request timed out
Jason 42: server reply: Jason 42 Sun Feb 26 19:42:17 2023, RTT = 16.54 ms
Jason 43: server reply: Jason 43 Sun Feb 26 19:42:17 2023, RTT = 18.78 ms
Jason 44: server reply: Jason 44 Sun Feb 26 19:42:17 2023, RTT = 27.72 ms
Jason 45: server reply: Jason 45 Sun Feb 26 19:42:17 2023, RTT = 23.36 ms
Jason 46: server reply: Jason 46 Sun Feb 26 19:42:17 2023, RTT = 16.13 ms
Jason 47: server reply: Jason 47 Sun Feb 26 19:42:17 2023, RTT = 17.36 ms
Jason 48: server reply: Jason 48 Sun Feb 26 19:42:17 2023, RTT = 17.28 ms
Jason 49: server reply: Jason 49 Sun Feb 26 19:42:17 2023, RTT = 17.27 ms
Jason 50: server reply: Jason 50 Sun Feb 26 19:42:17 2023, RTT = 17.61 ms
Min RTT = 14.27 ms
Max RTT = 30.50 ms
Avg RTT = 21.35 ms
Packet lost = 8.00%
```

3. Python code listing

```
#!/usr/bin/env python3
# Part 3 – UDP Pinger with Delays and Packet Losses

import time
import random
from socket import socket, AF_INET, SOCK_DGRAM

# Artifical Percentage of Packet Loss Occurrence
THRESHOLD = 10

# Create a UDP socket
serverSocket = socket(AF_INET, SOCK_DGRAM)

# Assign IP address and port number to socket
serverSocket.bind(('', 12000))

while True:
    # Receive the client packet along with the address it is coming from
    message, address = serverSocket.recvfrom(1024)

    # Creates an arbitrary, random RTT delay ranging from 10-20 milliseconds
    time.sleep(random.randint(10, 20) / 1000)

    # Does not send original packet back if random value is within the first 10
integers
    if random.randrange(100) < THRESHOLD:
        continue

    # The server responds
    serverSocket.sendto(message, address)
```

Part 4 – UDP Heartbeat Monitor

1. How to run the code
    a. To run the code, first, run the server program as a script from the command line as "**python3 jd4s.py**"
    b. Now, open a separate local terminal and run the client program as a script with the **1** parameter from the command line as "**python3 jd4c.py 1**". The parameter after the python file denotes the <u>first</u> client.
    c. Finally, open another separate local terminal and run the client program as a script with the **2** parameters from the command line as "**python3 jd4c.py 2**". The parameter after the python file denotes the <u>second</u> client.
2. Run-time screen captures
    a. the client sends heartbeat pings to the server.

```
duong-2:Desktop duong-jason$ python3 jd4c.py 1
Jason client1 heartbeat at Sun Feb 26 21:13:15 2023
Jason client1 heartbeat at Sun Feb 26 21:13:17 2023
Jason client1 heartbeat at Sun Feb 26 21:13:32 2023
Jason client1 heartbeat at Sun Feb 26 21:13:56 2023
Jason client1 heartbeat at Sun Feb 26 21:14:09 2023
Jason client1 heartbeat at Sun Feb 26 21:14:31 2023
Jason client1 heartbeat at Sun Feb 26 21:14:56 2023
```

```
duong-2:Desktop duong-jason$ python3 jd4c.py 2
Jason client2 heartbeat at Sun Feb 26 21:13:26 2023
Jason client2 heartbeat at Sun Feb 26 21:13:41 2023
Jason client2 heartbeat at Sun Feb 26 21:13:53 2023
Jason client2 heartbeat at Sun Feb 26 21:14:13 2023
Jason client2 heartbeat at Sun Feb 26 21:14:35 2023
Jason client2 heartbeat at Sun Feb 26 21:14:57 2023
```

b. **and c. server prints the received heartbeat pings from the client, and the time interval, server detects absence of client heartbeat and quits *might take longer than 20 seconds for last heartbeat as server is processing the other client***

```
duong-2:Desktop duong-jason$ python3 jd4s.py
Server received: Jason client1 heartbeat at Sun Feb 26 21:13:15 2023 Last heartbeat received 0 seconds ago
Server received: Jason client1 heartbeat at Sun Feb 26 21:13:17 2023 Last heartbeat received 2 seconds ago
Server received: Jason client2 heartbeat at Sun Feb 26 21:13:26 2023 Last heartbeat received 2 seconds ago
Server received: Jason client1 heartbeat at Sun Feb 26 21:13:32 2023 Last heartbeat received 15 seconds ago
Server received: Jason client2 heartbeat at Sun Feb 26 21:13:41 2023 Last heartbeat received 15 seconds ago
Server received: Jason client2 heartbeat at Sun Feb 26 21:13:53 2023 Last heartbeat received 12 seconds ago
Server received: Jason client1 heartbeat at Sun Feb 26 21:13:56 2023 Last heartbeat received 24 seconds ago
Server received: Jason client1 heartbeat at Sun Feb 26 21:14:09 2023 Last heartbeat received 13 seconds ago
Server received: Jason client2 heartbeat at Sun Feb 26 21:14:13 2023 Last heartbeat received 20 seconds ago
Server received: Jason client1 heartbeat at Sun Feb 26 21:14:31 2023 Last heartbeat received 21 seconds ago
Server received: Jason client2 heartbeat at Sun Feb 26 21:14:35 2023 Last heartbeat received 21 seconds ago
No heartbeat after 20 seconds. Server quits. Server stops.
duong-2:Desktop duong-jason$
```

3. Python code listing

a. **Include as text, the client program listing.**

```
#!/usr/bin/env python3
# Part 4 - UDP Heartbeat Monitor (Client)

import sys
import time
import random
from datetime import datetime
from socket import socket, AF_INET, SOCK_DGRAM

serverName = 'localhost'
serverPort = 12000

n_timeout = 0
```

```
# Used to identify a client process
client_id = sys.argv[1]

# Continuously sends message to the server
while True:
    clientSocket = socket(AF_INET, SOCK_DGRAM)

    # Creates an arbitrary delay ranging between 2-25 seconds
    time.sleep(random.randint(2, 25))

    message = f'Jason client{client_id} heartbeat at {datetime.now().ctime()}'

    print(message)
    clientSocket.sendto(message.encode(), (serverName, serverPort))

    clientSocket.close()
```

b. Include as text, the server program listing.

```
#!/usr/bin/env python3
# Part 4 - UDP Heartbeat Monitor (Server)

import time
from socket import socket, timeout, AF_INET, SOCK_DGRAM

# Create a UDP socket
serverSocket = socket(AF_INET, SOCK_DGRAM)

# Assign IP address and port number to socket
serverSocket.bind(('', 12000))

# Sets the timeout to 20 seconds
serverSocket.settimeout(20)

# Time difference is zero upon the first receive message from a client
delta_time = 0
last_heartbeat_time = {}

try:
    while True:
        # Receive the client packet along with the address it is coming from
        message, address = serverSocket.recvfrom(1024)

        time_recv = time.time()
        client_id = message[12:13]

        # The server calculate the time difference from the last heartbeat
        # (i.e., the time of the previous received message)
```

```
        if last_heartbeat_time.get(client_id):
            delta_time = int(time_recv - last_heartbeat_time[client_id])

        print(f'Server received: {message.decode()} Last heartbeat received
{delta_time} seconds ago')

        # Update time difference from the last heartbeat request
        # If it was the first response from a client, the server initiates the time
receive
        last_heartbeat_time[client_id] = time_recv

        # The server responds
        serverSocket.sendto(message, address)
except timeout:
    print('No heartbeat after 20 seconds. Server quits. Server stops.')
    serverSocket.close()
```