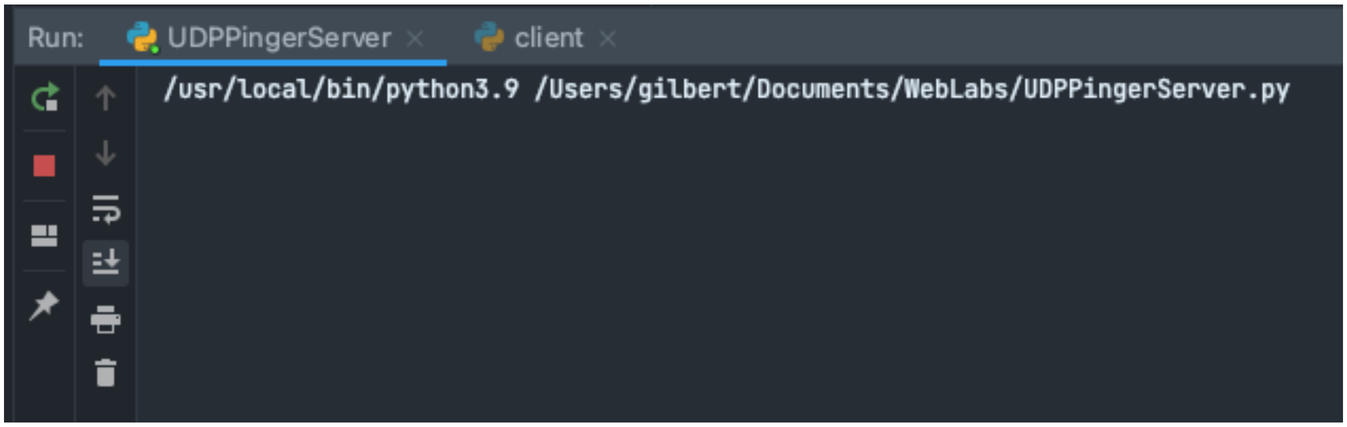
# Lab2: UDP Pinger Lab Report

### Server

The server is `UDPPingerServer.py`, while running the server, we would see:



```
/usr/local/bin/python3.9 /Users/gilbert/Documents/WebLabs/UDPPingerServer.py
```

The server sits in an infinite loop listening for incoming UDP packets. When a packet comes in and if a randomized integer is greater than or equal to 4, the server generates a reply message and sends it back to the client.

### Client

The client is `client.py`, as the below, which includes line-by-line explanation of the code.

The client sends 10 pings to the server. Because UDP is an unreliable protocol, a packet sent from the client to the server may be lost in the network, or vice versa. For this reason, the client cannot wait indefinitely for a reply to a ping message. The client would wait up to one second for a reply; if no reply is received within one second, theclient program should assume that the packet was lost during transmission across the network

```python
from socket import *
import time
import sys


def ping(host, port):
    server_name = host
    server_port = port
    client_socket = socket(AF_INET, SOCK_DGRAM)
    resps = []
    for seq in range(1, 11):
        # Send ping message to server and wait for response back On timeouts, you can use
the following to add to
```

```
        # resps
        # resps.append((seq, ‘Request timed out’, 0))
        # On successful responses, you should instead record the
        # server response and the RTT (must compute server_reply and rtt properly)
        # resps.append((seq, server_reply, rtt))
        #   Fill in start
        # print("seq: ", seq)
        time1 = time.time()
        outputs = 'Ping ' + str(seq) + " " + str(time1)
        # set time out
        client_socket.settimeout(1)
        # client send to server with output ping
        client_socket.sendto(outputs.encode(), (server_name, server_port))
        try:
            # receive response from server
            modified_message, server_address = client_socket.recvfrom(2048)
            # calculate timeout
            time_diff = time.time() - time1
            # decode current response
            cur_res = modified_message.decode()
            # print(cur_res + " RTT: " + str(time_diff))
            resps.append((seq, cur_res, float(time_diff)))
        except:
            # print("lost " + str(seq))
            resps.append((seq, 'Request timed out', 0))
        # Fill in end
    return resps


if __name__ == '__main__':
    resps = ping('127.0.0.1', 12000)
    print(resps)
```

As run the client, we see the `resps` as below:

```
/usr/local/bin/python3.9 /Users/gilbert/Documents/WebLabs/client.py
[(1, 'Reply 1 1648000215.326835 1648000215.326957 cb388ad626a022cb8b0e6356490e15ad\n',
0.00022912025451660156), (2, 'Request timed out', 0), (3, 'Reply 3 1648000216.3281898
1648000216.3284411 76b8f29c5aae64deac1bd54351733416\n', 0.00044608116149902344), (4,
'Reply 4 1648000216.328643 1648000216.328762 c0063ee9bc6400049b6dc6bf3bcb3968\n',
0.0002758502960205078), (5, 'Request timed out', 0), (6, 'Reply 6 1648000217.330049
1648000217.330165 a5623a8b149ee95953e41adb5084f7a3\n', 0.00021600723266601562), (7, 'Reply
7 1648000217.3302689 1648000217.3303308 2c474416ebc1866f2516d3100b5361a0\n',
0.00016617774963378906), (8, 'Reply 8 1648000217.3304389 1648000217.3305118
165849a89908ad0007a02306e3d267c0\n', 0.00015997886657714844), (9, 'Reply 9
1648000217.330602 1648000217.330663 2923855aed599e7b91a08e8fdc2a3465\n',
0.00015997886657714844), (10, 'Reply 10 1648000217.330765 1648000217.330913
8ebc0a91aa4670806248cf28b78ce5eb\n', 0.00026988983154296875)]

Process finished with exit code 0
```

### Optional Exercises

1. The standard ping program works, which reports the minimum, maximum, and average RTTs at the end of all pings from the client. In addition, calculate the packet loss rate (in percentage).

   The modified Stand Client is `ClientStandard.py`

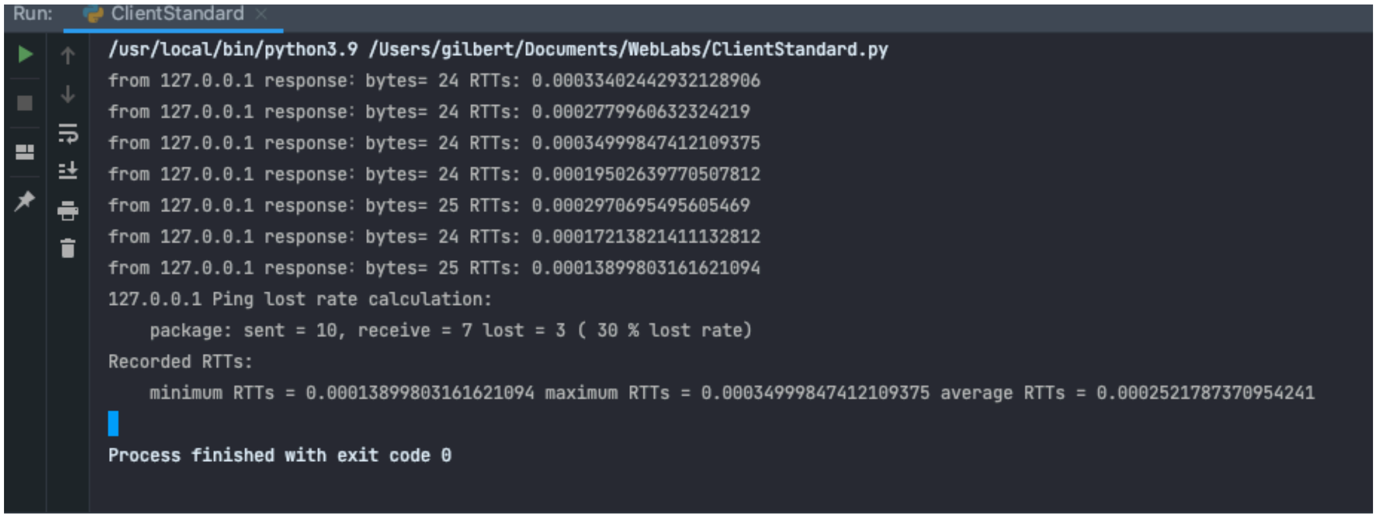
   ```
   from socket import *
   import time
   import sys


   def ping(host, port):
       server_name = host
       server_port = port
       client_socket = socket(AF_INET, SOCK_DGRAM)
       resps = []
       raceNum = 0
       max_diff_time = -1
       min_diff_time = 2
       aver_diff_time = 0
       for seq in range(1, 11):
           # Send ping message to server and wait for response back On timeouts, you can
   use the following to add to
           # resps
           # resps.append((seq, ‘Request timed out’, 0))
           # On successful responses, you should instead record the
           # server response and the RTT (must compute server_reply and rtt properly)
   ```

```
        # resps.append((seq, server_reply, rtt))
        #   Fill in start
        # print("seq: ", seq)
        time1 = time.time()
        outputs = 'Ping ' + str(seq) + " " + str(time1)
        # set time out
        client_socket.settimeout(1)
        # client send to server with output ping
        client_socket.sendto(outputs.encode(), (server_name, server_port))
        try:
            # receive response from server
            modified_message, server_address = client_socket.recvfrom(2048)
            # calculate timeout
            time_diff = time.time() - time1
            print("from", server_name, "response: bytes=", str(len(outputs)), "RTTs:", str(time_diff))
            raceNum += 1
            aver_diff_time += time_diff
            if time_diff > max_diff_time:
                max_diff_time = time_diff
            if time_diff < min_diff_time:
                min_diff_time = time_diff
        except:
            # print("lost " + str(seq))
            resps.append((seq, 'Request timed out', 0))
        # Fill in end
    print(server_name, "Ping lost rate calculation:")
    print("\tpackage: sent = 10, receive =", str(raceNum), "lost =", str(10 - raceNum),
          "(", str(int((10 - raceNum) * 100 / 10)), "% lost rate)")
    if raceNum != 0:
        print("Recorded RTTs:")
        print("\tminimum RTTs =", min_diff_time, "maximum RTTs =", max_diff_time, "average RTTs =",
              aver_diff_time / raceNum)
    return resps


if __name__ == '__main__':
    resps = ping('127.0.0.1', 12000)
    # print(resps)
```

As we run the client, we see the calculations:



```
/usr/local/bin/python3.9 /Users/gilbert/Documents/WebLabs/ClientStandard.py
from 127.0.0.1 response: bytes= 24 RTTs: 0.00033402442932128906
from 127.0.0.1 response: bytes= 24 RTTs: 0.0002779960632324219
from 127.0.0.1 response: bytes= 24 RTTs: 0.00034999847412109375
from 127.0.0.1 response: bytes= 24 RTTs: 0.00019502639770507812
from 127.0.0.1 response: bytes= 25 RTTs: 0.0002970695495605469
from 127.0.0.1 response: bytes= 24 RTTs: 0.00017213821411132812
from 127.0.0.1 response: bytes= 25 RTTs: 0.00013899803161621094
127.0.0.1 Ping lost rate calculation:
    package: sent = 10, receive = 7 lost = 3 ( 30 % lost rate)
Recorded RTTs:
    minimum RTTs = 0.00013899803161621094 maximum RTTs = 0.00034999847412109375 average RTTs = 0.0002521787370954241

Process finished with exit code 0
```

```
/usr/local/bin/python3.9 /Users/gilbert/Documents/WebLabs/ClientStandard.py
from 127.0.0.1 response: bytes= 24 RTTs: 0.00033402442932128906
from 127.0.0.1 response: bytes= 24 RTTs: 0.0002779960632324219
from 127.0.0.1 response: bytes= 24 RTTs: 0.00034999847412109375
from 127.0.0.1 response: bytes= 24 RTTs: 0.00019502639770507812
from 127.0.0.1 response: bytes= 25 RTTs: 0.0002970695495605469
from 127.0.0.1 response: bytes= 24 RTTs: 0.00017213821411132812
from 127.0.0.1 response: bytes= 25 RTTs: 0.00013899803161621094
127.0.0.1 Ping lost rate calculation:
  package: sent = 10, receive = 7 lost = 3 ( 30 % lost rate)
Recorded RTTs:
  minimum RTTs = 0.00013899803161621094 maximum RTTs = 0.00034999847412109375 average
RTTs = 0.0002521787370954241

Process finished with exit code 0
```

2. The Heartbeat can be used to check if an application is up and running and to report one-way packet loss. The client sends a sequence number and current timestamp in the UDP packet to the server, which is listening for the Heartbeat (i.e., the UDP packets) of the client. Upon receiving the packets, the server calculates the time difference and reports any lost packets. If the Heartbeat packets are missing for some specified period of time, we can assume that the client application has stopped.

   We implemented the UDP Heartbeat (both client and server).

   `UDPHeartbeatServer.py`

```
# UDPPingerServer.py
# We will need the following module to generate randomized lost packets
import sys
import time
from socket import *
```

```
def serve(port):
    # Create a UDP socket
    # Notice the use of SOCK_DGRAM for UDP packets
    server_socket = socket(AF_INET, SOCK_DGRAM)
    # Assign IP address and port number to socket
    server_socket.bind(('', port))

    while True:
        try:
            message, address = server_socket.recvfrom(1024)
            message = message.decode()
            message = message.split()[1]
            time_diff = time.time() - float(message)
            print("receive RTT:", time_diff)
        except KeyboardInterrupt:
            server_socket.close()
            sys.exit()
        except:
            continue


if __name__ == '__main__':
    serve(12000)
```

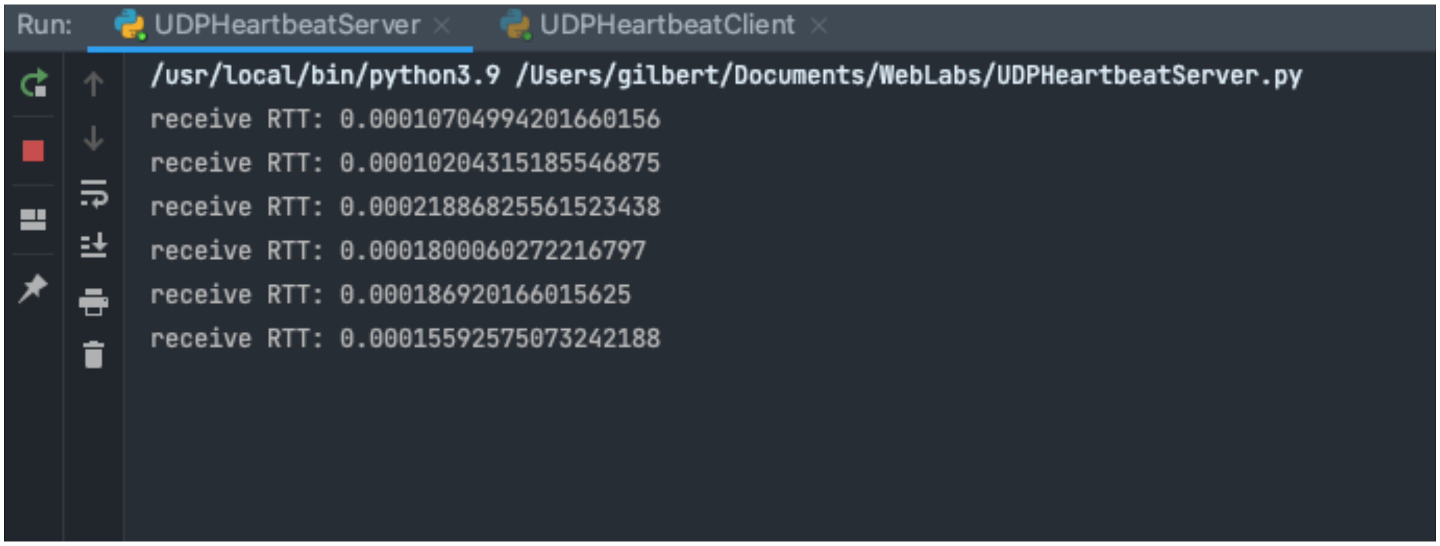
`UDPHeartbeatClient.py`

```
# UDPPingerClient.py
from socket import *
import time


def ping(host, port):
    serverName = host
    serverPort = port
    clientSocket = socket(AF_INET, SOCK_DGRAM)
    while True:
        time1 = time.time()
        outputs = 'Heartbeat ' + str(time1)
        clientSocket.sendto(outputs.encode(), (serverName, serverPort))
        time.sleep(10)
```

```python
if __name__ == '__main__':
    ping('127.0.0.1', 12000)
```

As we run the heart beat server and client, we would see the server is keeping heart beating, with printing RTTs:



```
/usr/local/bin/python3.9 /Users/gilbert/Documents/WebLabs/UDPHeartbeatServer.py
receive RTT: 0.00010704994201660156
receive RTT: 0.00010204315185546875
receive RTT: 0.00021886825561523438
receive RTT: 0.0001800060272216797
receive RTT: 0.000186920166015625
receive RTT: 0.00015592575073242188
```