Socket Programming Lab #2: UDP Pinger

Full program for *client.py* provided below:

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

def ping(host, port):
    resps = []
    clientSocket = socket(AF_INET, SOCK_DGRAM)

    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
        clientSocket.settimeout(1)

        begin = time.time()
        message = "Ping {0} {1}".format(str(seq), str(begin))

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

        try:
            modifiedMessage, serverAddress = clientSocket.recvfrom(1024)
            end = time.time()

            server_reply = modifiedMessage.decode()
            rtt = end - begin

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

        except timeout:
            resps.append((seq, 'Request timed out', 0))
        #Fill in end

    clientSocket.close()

    return resps

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

In line 18, client socket operations are set to time out after 1 second. In order to calculate RTT, we measure the time it takes for a client to send a message to a server and for the server's response to reach the client. Line 20 records the starting time the client program will send a message to the server. Line 21 prepares the message that the client will send to the server. In line 23, the message is passed through the client socket to be sent to the server socket specified by the server's IP address and port number.

```
19          clientSocket.settimeout(1)
20
21          begin = time.time()
22          message = "Ping {0} {1}".format(str(seq), str(begin))
23
24          clientSocket.sendto(message.encode(), (host, port))
```

Lines 26 - 33 define the try block if a server response is received before the 1 second timeout. Line 27 receives the server response through the client socket. Line 28 records the time the client received the server response, which will later be used to calculate RTT. Line 30 stores the decoded server response into a separate variable. Line 31 calculates RTT by subtracting the time the client sent a message from the time the client received the server response. Line 33 appends a 3-tuple to the list of responses, which include the sequence number, the decoded server response, and RTT.

```
26          try:
27              modifiedMessage, serverAddress = clientSocket.recvfrom(1024)
28              end = time.time()
29
30              server_reply = modifiedMessage.decode()
31              rtt = end - begin
32
33              resps.append((seq, server_reply, rtt))
```

Full program for *clientStats.py* provided below:

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

def ping(host, port):

    resps = []
    clientSocket = socket(AF_INET, SOCK_DGRAM)

    for seq in range(1,11):
        clientSocket.settimeout(1)

        begin = time.time()
        message = "Ping {0} {1}".format(str(seq), str(begin))

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

        try:
            modifiedMessage, serverAddress = clientSocket.recvfrom(1024)
            end = time.time()

            server_reply = modifiedMessage.decode()
            rtt = end - begin

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

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

    clientSocket.close()

    return resps

```

```
34  if __name__ == '__main__':
35      resps = ping('127.0.0.1', 12000)
36      print(resps, "\n")
37
38      maxRTT = averageRTT = packetLoss = packetReceived = 0
39      minRTT = sys.maxsize
40
41      for response in resps:
42          rtt = response[2]
43          message = response[1]
44
45          maxRTT = max(maxRTT, rtt)
46
47          if message != "Request timed out":
48              minRTT = min(minRTT, rtt)
49              averageRTT += rtt
50              packetReceived += 1
51          else:
52              packetLoss += 1
53
54      if packetLoss == len(resps):
55          minRTT = 0
56      else:
57          packetLoss = (packetLoss / len(resps)) * 100
58          averageRTT = (averageRTT / packetReceived) * 1000
59          maxRTT *= 1000
60          minRTT *= 1000
61
62      print("{} packets transmitted, {} packets received, {:.1f}% packet loss".format(len(resps), packetReceived, packetLoss))
63      print("round-trip min/avg/max = {:.3f}/{:.3f}/{:.3f} ms".format(minRTT, averageRTT, maxRTT))
```

The *ping* function remains the same as in the original *client.py* program. In *clientStats.py*, the *__main__* environment is altered to include the ping statistics. In line 38 - 39, the statistics that will be displayed are initialized to 0, with the exception of *minRTT* which is initialized to *sys.maxsize*.

```
38      maxRTT = averageRTT = packetLoss = packetReceived = 0
39      minRTT = sys.maxsize
```

Lines 41 - 52 iterate through each response from the list of responses as compiled in the *ping* function. Line 42 extracts the RTT data from the third entry in the 3-tuple and line 43 extracts the message. Line 45 calculates the maximum RTT. In lines 47 - 50, the conditional is set up if the response does not describe a packet loss then the program continues to calculate minimum RTT, the average RTT, and the amount of packets received. Lines 50 - 51, If the response does describe a packet loss then only the amount of packet losses is incremented.

```
41      for response in resps:
42          rtt = response[2]
43          message = response[1]
44
45          maxRTT = max(maxRTT, rtt)
46
47          if message != "Request timed out":
48              minRTT = min(minRTT, rtt)
49              averageRTT += rtt
50              packetReceived += 1
51          else:
52              packetLoss += 1
```

Line 54 calculates the percentage of packet losses. In the event that the percentage of packet loss is 100%, then we expect all other statistics to return 0. Since all statistics were initially set to 0, except for minimum RTT, only minimum RTT needs to be set to 0. This is described in lines 56 - 57. Otherwise, we calculate average RTT and convert the average, minimum and maximum RTTs to milliseconds. This is described in lines 58 - 61. Lines 63 - 64 print out the statistics to the user interface.

```
54  packetLoss = (packetLoss / len(resps)) * 100
55
56  if packetLoss == 100:
57      minRTT = 0
58  else:
59      averageRTT = (averageRTT / packetReceived) * 1000
60      maxRTT *= 1000
61      minRTT *= 1000
62
63  print("{} packets transmitted, {} packets received, {:.1f}% packet loss".format(len(resps), packetReceived, packetLoss))
64  print("round-trip min/avg/max = {:.3f}/{:.3f}/{:.3f} ms".format(minRTT, averageRTT, maxRTT))
```