Task 1 - SMTP Mail Client

Your task is to develop a simple mail client that sends email to any recipient. Your client will need to connect to a mail server, dialogue with the mail server using the SMTP protocol, and send an email message to the mail server. Python provides a module, called smtplib, which has built-in methods to send mail using SMTP protocol. However, you will not be using this module in this assignment, because it hides the details of SMTP and socket programming.

In order to limit spam, some mail servers do not accept TCP connections from arbitrary sources. For the experiment described below, you may want to try connecting both to your university mail server and to a popular Webmail server, such as an AOL mail server.

Code

Below you will find the skeleton code for the client. You are to complete the skeleton code. The places where you need to fill in code are marked with #Fill in start and #Fill in end. Each place may require one or more lines of code.

Additional Notes

In some cases, the receiving mail server might classify your email as junk. Make sure that you check your junk/spam folder when you look for the email sent from your client.

What to Hand In

Upload the complete code for the SMTP mail client. Make sure you follow the SMTP protocol for non-encrypted communication. Please make sure to use port 1025 and mail server address 127.0.0.1 to receive full credit.
Note: Comment out all the print statements in your code, otherwise it will fail to autograde your assignment.

Skeleton Python Code for the Mail Client

```
from socket import *


def smtp_client(port=1025, mailserver='127.0.0.1'):
    msg = "\r\n My message"
    endmsg = "\r\n.\r\n"

    # Choose a mail server (e.g. Google mail server) if you want to verify the
script beyond GradeScope

    # Create socket called clientSocket and establish a TCP connection with
mailserver and port
```

```
    # Fill in start
    # Fill in end

    recv = clientSocket.recv(1024).decode()
    print(recv)
    if recv[:3] != '220':
        #print('220 reply not received from server.')

    # Send HELO command and print server response.
    heloCommand = 'HELO Alice\r\n'
    clientSocket.send(heloCommand.encode())
    recv1 = clientSocket.recv(1024).decode()
    print(recv1)
    if recv1[:3] != '250':
        #print('250 reply not received from server.')

    # Send MAIL FROM command and print server response.
    # Fill in start
    # Fill in end

    # Send RCPT TO command and print server response.
    # Fill in start
    # Fill in end

    # Send DATA command and print server response.
    # Fill in start
    # Fill in end

    # Send message data.
    # Fill in start
    # Fill in end

    # Message ends with a single period.
    # Fill in start
    # Fill in end

    # Send QUIT command and get server response.
    # Fill in start
    # Fill in end


if __name__ == '__main__':
    smtp_client(1025, '127.0.0.1')
```

Most Common issues

1. SMTP commands do not follow requirements documented in the RFC referenced above
    a. SMTP commands are case-inconsistent or all lowercase (reference RFC 2821 for requirements)
2. Not establishing socket connection properly
    a. Reference to read - python docs
3. Not encoding or decoding properly
4. Incorrect port or server address

Task 2 - Pinger

Ping is a computer network application used to test whether a particular host is reachable across an IP network. It is also used to self-test the network interface card of the computer or as a latency test. It works
by sending ICMP “echo reply” packets to the target host and listening for ICMP “echo reply” replies. The "echo reply" is sometimes called a pong. Ping measures the round-trip time, records packet loss, and prints a statistical summary of the echo reply packets received (the minimum, maximum, and the mean of the round-trip times and in some versions the standard deviation of the mean).

Your task is to develop your own Ping application in Python. Your application will use ICMP but, in order to keep it simple, will not exactly follow the official specification in RFC 1739. Note that you will only need to write the client side of the program, as the functionality needed on the server side is built into almost all operating systems.

You should complete the Ping application so that it sends ping requests to a specified host separated by approximately one second. Each message contains a payload of data that includes a timestamp. After sending each packet, the application waits up to one second to receive a reply. If one second goes by without a reply from the server, then the client assumes that either the ping packet or the pong packet was lost in the network (or that the server is down).

Code

Below you will find the skeleton code for the client. You are to complete the skeleton code. The place where you need to fill in code is marked with #Fill in start and #Fill in end. In addition, you will need to add a few lines of code in order to calculate minimum

time, average time, maximum time, and stdev time and print the results like in the operating system.

Additional Notes

1. In the “receiveOnePing” method, you need to receive the structure ICMP_ECHO_REPLY and fetch the information you need, such as checksum, sequence number, time to live (TTL), etc. Study the “sendOnePing” method before trying to complete the “receiveOnePing” method.
2. You do not need to be concerned about the checksum, as it is already given in the code.
3. This lab requires the use of raw sockets. In some operating systems, you may need **administrator/root privileges** to be able to run your Pinger program.
4. See the end of this programming exercise for more information on ICMP.

Testing the Pinger

First, test your client by sending packets to localhost, that is, 127.0.0.1.
Then, you should see how your Pinger application communicates across the network by pinging servers
in different continents.

Example of a correct output:

- “No.no.e” is a non-valid domain, therefore, the return value (vars) of ping is ['0', '0.0', '0', '0.0']. Please make sure you use this exact value.
- “Google.co.il” is a valid domain, therefore, the return value is a list that can be created in the following way:

```
vars = [str(round(packet_min, 2)), str(round(packet_avg,
2)),str(round(packet_max, 2)), str(round(stdev(stdev_var),2))]
```

The method “ping” must return a Python list with the above values for a valid and a non-valid domain. [min,avg,max,stdev]. Return values must be in milliseconds.

SKELETON CODE

```
from socket import *
import os
import sys
import struct
import time
import select
import binascii
# Should use stdev

ICMP_ECHO_REQUEST = 8
```

```
def checksum(string):
    csum = 0
    countTo = (len(string) // 2) * 2
  count = 0

    while count < countTo:
        thisVal = (string[count + 1]) * 256 + (string[count])
        csum += thisVal
        csum &= 0xffffffff
        count += 2

    if countTo < len(string):
        csum += (string[len(string) - 1])
        csum &= 0xffffffff

    csum = (csum >> 16) + (csum & 0xffff)
    csum = csum + (csum >> 16)
    answer = ~csum
    answer = answer & 0xffff
    answer = answer >> 8 | (answer << 8 & 0xff00)
    return answer



def receiveOnePing(mySocket, ID, timeout, destAddr):
    timeLeft = timeout

    while 1:
        startedSelect = time.time()
        whatReady = select.select([mySocket], [], [], timeLeft)
        howLongInSelect = (time.time() - startedSelect)
        if whatReady[0] == []:  # Timeout
            return "Request timed out."

        timeReceived = time.time()
        recPacket, addr = mySocket.recvfrom(1024)

        # Fill in start

        # Fetch the ICMP header from the IP packet

        # Fill in end
        timeLeft = timeLeft - howLongInSelect
        if timeLeft <= 0:
            return "Request timed out."


def sendOnePing(mySocket, destAddr, ID):
    # Header is type (8), code (8), checksum (16), id (16), sequence (16)
```

```
myChecksum = 0
   # Make a dummy header with a 0 checksum
   # struct -- Interpret strings as packed binary data
   header = struct.pack("bbHHh", ICMP_ECHO_REQUEST, 0, myChecksum, ID, 1)
   data = struct.pack("d", time.time())
   # Calculate the checksum on the data and the dummy header.
   myChecksum = checksum(header + data)

   # Get the right checksum, and put in the header

   if sys.platform == 'darwin':
       # Convert 16-bit integers from host to network  byte order
       myChecksum = htons(myChecksum) & 0xffff
   else:
       myChecksum = htons(myChecksum)


   header = struct.pack("bbHHh", ICMP_ECHO_REQUEST, 0, myChecksum, ID, 1)
   packet = header + data

   mySocket.sendto(packet, (destAddr, 1))  # AF_INET address must be tuple, not
str


   # Both LISTS and TUPLES consist of a number of objects
   # which can be referenced by their position number within the object.

def doOnePing(destAddr, timeout):
   icmp = getprotobyname("icmp")


   # SOCK_RAW is a powerful socket type. For more details:
http://sockraw.org/papers/sock_raw
   mySocket = socket(AF_INET, SOCK_RAW, icmp)

   myID = os.getpid() & 0xFFFF  # Return the current process i
   sendOnePing(mySocket, destAddr, myID)
   delay = receiveOnePing(mySocket, myID, timeout, destAddr)
   mySocket.close()
   return delay


def ping(host, timeout=1):
   # timeout=1 means: If one second goes by without a reply from the server,
# the client assumes that either the client's ping or the server's pong is lost
   dest = gethostbyname(host)
  print("Pinging " + dest + " using Python:")
   print("")
```

```
    # Calculate vars values and return them
    #  vars = [str(round(packet_min, 2)), str(round(packet_avg, 2)),
str(round(packet_max, 2)),str(round(stdev(stdev_var), 2))]
    # Send ping requests to a server separated by approximately one second
    for i in range(0,4):
        delay = doOnePing(dest, timeout)
        print(delay)
        time.sleep(1)  # one second

    return vars

if __name__ == '__main__':
    ping("google.co.il")
```

Most Common issues

1. var values are not strings
2. var values are not calculated correctly
    a. Recommend to print out your var values and analyze them to see if they actually make sense. If they don't, revisit your method for calculating.

Task 3 - ICMP Traceroute

Traceroute is a computer networking diagnostic tool which allows a user to trace the route from a host running the traceroute program to any other host in the world. Traceroute is implemented with ICMP messages. It works by sending ICMP echo (ICMP type '8') messages to the same destination with increasing value of the time-to-live (TTL) field. The routers along the traceroute path return ICMP Time Exceeded (ICMP type '11') when the TTL field becomes zero. The final destination sends an ICMP reply (ICMP type '0') messages on receiving the ICMP echo request. The IP addresses of the routers which send replies can be extracted from the received packets. The round-trip time between the sending host and a router is determined by setting a timer at the sending host.

Your task is to develop your own Traceroute application in python using ICMP. Your application will use ICMP but, in order to keep it simple, will not exactly follow the official specification in RFC 1739.

Code

Below you will find the skeleton code for the client. You are to complete the skeleton code. The places where you need to fill in code are marked with #Fill in start and #Fill in end. Each place may require one or more lines of code.

Additional Notes

1. You do not need to be concerned about the checksum, as it is already given in the assignment skeleton code.
2. This assignment requires the use of raw sockets. In some operating systems (e.g. MacOS, Windows), you may need administrator/root privileges to be able to run your Traceroute program.
3. Local testing may require you to turn your firewall or antivirus software off to allow the messages to be sent and received.
4. See the end of Lab 4 'ICMP Pinger' programming exercise for more information on ICMP.

Testing the Pinger

Test your client by running your code to trace google.com or bing.com. Your output should return a list and meet the acceptance criteria format provided below.

Output Requirements (Acceptance Criteria)

Your code must produce the traceroute output in the format provided below for Gradescope to verify your code is working correctly.

1. Your trace must collect hop number, roundtrip time (rtt), host ip, and the hostname. If a hostname is not available for a host, you should provide an explicit hostname as "hostname not returnable". Also, if a host is timing out (not responding), you must record this in your trace list item with the text "Request timed out". Example provided below:

```
Example:    1    12ms  10.10.111.10 hop1.com
            2    30ms  10.10.112.10 hostname not returnable
            3    *     Request timed out
            4    5ms   10.10.110.1   target-host.com
```

2. Your get_route() function must return a nested list with trace output. That is, each trace row must be a list that includes the trace results as individual items in the list, which is also inside an overall traceroute list. Example provided below:

   Example: [['1', '12ms', '10.10.111.10', 'hop1.com'], ['2', '30ms', '10.10.112.10', 'hostname not returnable'], ['3', '*', 'Request timed out'], ['4', '5ms', '10.10.110.1', 'target-host.com']]

   Note: Your output will be parsed to verify that it includes the relevant information, so if you do not provide the output of your function in a nested

list, your solution will not work correctly and you will not receive points. Also, note that the example lists include all data as strings.

SKELETON CODE

```
from socket import *
import os
import sys
import struct
import time
import select
import binascii

ICMP_ECHO_REQUEST = 8
MAX_HOPS = 30
TIMEOUT = 2.0
TRIES = 1
# The packet that we shall send to each router along the path is the ICMP echo
# request packet, which is exactly what we had used in the ICMP ping exercise.
# We shall use the same packet that we built in the Ping exercise

def checksum(string):
    csum = 0
    countTo = (len(string) // 2) * 2
    count = 0

    while count < countTo:
        thisVal = (string[count + 1]) * 256 + (string[count])
        csum += thisVal
        csum &= 0xffffffff
        count += 2

    if countTo < len(string):
        csum += (string[len(string) - 1])
        csum &= 0xffffffff

    csum = (csum >> 16) + (csum & 0xffff)
    csum = csum + (csum >> 16)
    answer = ~csum
    answer = answer & 0xffff
    answer = answer >> 8 | (answer << 8 & 0xff00)
    return answer

def build_packet():
    #Fill in start
    # In the sendOnePing() method of the ICMP Ping exercise ,firstly the header of our
    # packet to be sent was made, secondly the checksum was appended to the header and
    # then finally the complete packet was sent to the destination.

    # Make the header in a similar way to the ping exercise.
    # Append checksum to the header.
# Don’t send the packet yet , just return the final packet in this function.
```

```
    #Fill in end

    # So the function ending should look like this

    packet = header + data
    return packet

def get_route(hostname):
    timeLeft = TIMEOUT
    tracelist1 = [] #This is your list to use when iterating through each trace
    tracelist2 = [] #This is your list to contain all traces

    for ttl in range(1,MAX_HOPS):
        for tries in range(TRIES):
            destAddr = gethostbyname(hostname)

            #Fill in start
            # Make a raw socket named mySocket
            #Fill in end

            mySocket.setsockopt(IPPROTO_IP, IP_TTL, struct.pack('I', ttl))
            mySocket.settimeout(TIMEOUT)
            try:
                d = build_packet()
                mySocket.sendto(d, (hostname, 0))
                t= time.time()
                startedSelect = time.time()
                whatReady = select.select([mySocket], [], [], timeLeft)
                howLongInSelect = (time.time() - startedSelect)
                if whatReady[0] == []: # Timeout
                    tracelist1.append("* * * Request timed out.")
                    #Fill in start
                    #You should add the list above to your all traces list
                    #Fill in end
                recvPacket, addr = mySocket.recvfrom(1024)
                timeReceived = time.time()
                timeLeft = timeLeft - howLongInSelect
                if timeLeft <= 0:
                    tracelist1.append("* * * Request timed out.")
                    #Fill in start
                    #You should add the list above to your all traces list
                    #Fill in end
            except timeout:
                continue

            else:
                #Fill in start
                #Fetch the icmp type from the IP packet
                #Fill in end
                try: #try to fetch the hostname
                    #Fill in start
                    #Fill in end
                except herror:   #if the host does not provide a hostname
                    #Fill in start
```

```
#Fill in end

        if types == 11:
            bytes = struct.calcsize("d")
            timeSent = struct.unpack("d", recvPacket[28:28 +
            bytes])[0]
            #Fill in start
            #You should add your responses to your lists here
            #Fill in end
        elif types == 3:
            bytes = struct.calcsize("d")
            timeSent = struct.unpack("d", recvPacket[28:28 + bytes])[0]
            #Fill in start
            #You should add your responses to your lists here
            #Fill in end
        elif types == 0:
            bytes = struct.calcsize("d")
            timeSent = struct.unpack("d", recvPacket[28:28 + bytes])[0]
            #Fill in start
            #You should add your responses to your lists here and return your list if your destination IP is met
            #Fill in end
        else:
            #Fill in start
            #If there is an exception/error to your if statements, you should append that to your list here
            #Fill in end
        break
    finally:
        mySocket.close()
```

Most Common issues

1. Not returning anything from your function
2. Not returning data in the correct format as laid out in the acceptance criteria example
    a. Recommend to print out your returned values and compare them to the acceptance criteria example formatting