Due Sep 29th before 10 am on Gradescope

Programming problem

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, we will not be using this module in this lab, because it hides the details of SMTP and socket programming.

In order to limit spam, some mail servers do not accept TCP connection 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. You may also try making your connection both from your home and from your university campus.

Problem 1. [5 points]

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 e-mail as junk. Make sure you check the junk/spam folder when you look for the e-mail sent from your client.

What to Hand in In your submission, you are to provide the complete code for your SMTP mail client as well as a screenshot showing that you indeed receive the e-mail message.

Skeleton Python Code for the Mail Client

```python
from socket import *

msg = "\r\nI love computer networks!"
endmsg = "\r\n\r\n"

# Choose a mail server (e.g. Google mail server) and call it mailserver
mailserver = #Fill in start #Fill in end

# Create socket called clientSocket and establish a TCP connection with mailserver
#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\n’
cclientSocket.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
Written problems

Problem 2. [3 points] In this problem, we consider the delay introduced by the TCP slow-start phase. Consider a client and a Web server directly connected by one link of rate $R$. Suppose the client wants to retrieve an object whose size is exactly equal to $15S$, where $S$ is the maximum segment size (MSS). Denote the round-trip time between client and server as $RTT$ (assumed here, rather naively, to be a constant). Ignoring protocol headers, determine the time to retrieve the object (including TCP connection establishment) when

a) $4S/R > S/R + RTT > 2S/R$

b) $S/R + RTT > 4S/R$

c) $S/R > RTT$

Problem 3. [3 points]

Briefly answer the following questions:

a) Suppose there are three routers between a source host and a destination host. Ignoring fragmentation, an IP segment sent from the source host to the destination host will travel over how many interfaces? How many forwarding tables will be indexed to move the datagram from the source to the destination?

b) Suppose an application generates chunks of 40 bytes of data every 20 msec, and each chunk gets encapsulated in a TCP segment and then an IP datagram. What percentage of each datagram will be overhead, and what percentage will be application data?

c) In Section 4.2, we noted that there can be no input queuing if the switching fabric is $n$ times faster than the input line rates, assuming $n$ input lines all have the same line rate. Explain why this should be so.

Problem 4. [4 points] Consider the topology shown in Figure 4.20 of the text (equivalent figure 4.17 from 6th edition reproduced below). Denote the three subnets with host (starting clockwise at 12:00) as Networks A, B, and C. Denote the subnets without hosts as Networks D, E, and F.

![Figure 4.17](image-url) Three routers interconnecting six subnets

a) Assign network address to each of these six subnets, with the following constraints: All addresses must be allocated from 214.97.254/23; Subnet A should have enough addresses to support 250 interfaces; Subnet
B should have enough addresses to support 120 interfaces; and subnet C should have enough addresses to support 120 interfaces. Of course, subnets D, E, and F should each be able to support two interfaces. For each subnet, the assignment should take the form a.b.c.d/x - e.f.g.h/y.

b) Using your answer to part (a), provide the forwarding tables (using longest prefix matching) for each of the three routers.

**Problem 5. [2.5 points]** Consider the following network. With the indicated link costs, use Dijkstra’s shortest-path algorithm to compute the shortest path from x to all network nodes. Show how the algorithm works by computing a table similar to Table 4.3 in the text.

![Network Diagram](network_diagram1.png)

**Problem 6. [2.5 points]** Consider the network shown below, and assume that each node initially knows the costs to each of its neighbors. Consider the distance vector algorithm and show the distance table entries at node z.

![Network Diagram](network_diagram2.png)