13. Simple Mail Transfer Protocol

Santhisenan A

March 14, 2018

1 Aim

Implement Simple Mail Transfer Protocol

2 Simple Mail Transfer Protocol

Email is emerging as the one of the most valuable service in internet today. Most of the internet systems use SMTP as a method to transfer mail from one user to another. SMTP is a push protocol and is used to send the mail whereas POP (post office protocol) or IMAP (internet message access protocol) are used to retrieve those mails at the receivers side.

2.1 SMTP Fundamentals

SMTP is an application layer protocol. The client who wants to send the mail opens a TCP connection to the SMTP server and then sends the mail across the connection. The SMTP server is always on listening mode. As soon as it listens for a TCP connection from any client, the SMTP process initiates a connection on that port (25). After successfully establishing the TCP connection the client process sends the mail instantly.

3 SMTP Protocol

SMTP uses commands and responses to transfer messages between an MTA client and an MTA server. The command is from an MTA client to an MTA server; the response is from an MTA server to the MTA client. Each command or reply is terminated by a two- character (carriage return and line feed) end-of-line token.

Keyword	*Argument(s)*	*Description*
HELO	Sender's host name	Identifies itself
MAIL FROM	Sender of the message	Identifies the sender of the message
RCPT TO	Intended recipient	Identifies the recipient of the message
DATA	Body of the mail	Sends the actual message
QUIT		Terminates the message
RSET		Aborts the current mail transaction
VRFY	Name of recipient	Verifies the address of the recipient
NOOP		Checks the status of the recipient
TURN		Switches the sender and the recipient
EXPN	Mailing list	Asks the recipient to expand the mailing list
HELP	Command name	Asks the recipient to send information about the command sent as the argument
SEND FROM	Intended recipient	Specifies that the mail be delivered only to the terminal of the recipient, and not to the mailbox
SMOL FROM	Intended recipient	Specifies that the mail be delivered to the terminal *or* the mailbox of the recipient
SMAL FROM	Intended recipient	Specifies that the mail be delivered to the terminal *and* the mailbox of the recipient

Figure 1: SMTP Commands

3.1 Commands

Commands are sent from the client to the server. It consists of a keyword followed by zero or more arguments.

3.2 Responses

Responses are sent from the server to the client. A response is a three- digit code that may be followed by additional textual information.

Code	*Description*
Positive Completion Reply	
211	System status or help reply
214	Help message
220	Service ready
221	Service closing transmission channel
250	Request command completed
251	User not local; the message will be forwarded
Positive Intermediate Reply	
354	Start mail input
Transient Negative Completion Reply	
421	Service not available
450	Mailbox not available
451	Command aborted: local error
452	Command aborted; insufficient storage
Permanent Negative Completion Reply	
500	Syntax error; unrecognized command

Figure 2: SMTP Responses

Code	*Description*
501	Syntax error in parameters or arguments
502	Command not implemented
503	Bad sequence of commands
504	Command temporarily not implemented
550	Command is not executed; mailbox unavailable
551	User not local
552	Requested action aborted; exceeded storage location
553	Requested action not taken; mailbox name not allowed
554	Transaction failed

Figure 3: SMTP Responses (contd.)

4 Mail Transfer Phases

4.1 Connection Establishment

After a client has made a TCP connection to port 25, the server starts the connection phase. This phase includes 3 phases:

- The server sends code 220 (Service Ready) to tell the client that it is ready to recieve the mail. If the server is not ready, it sends 421 (service unavailable).
- The client sends HELO message to identify itself using its domain name and address. This step informs the server of the domain name of the client.
- The server responds with a code 250 (Request Command Completed) or some other code depending on the situation.

4.2 Message Transfer

After connection has been established between the SMTP client and server, a single message between a sender and one or more recipients can be exchanged. This phase involves eight steps. Steps 3 and 4 are repeated if there is more than one recipient.

- The client sends the MAIL FROM message to introduce the sender of the message. It includes the mail address of the sender (mailbox and the domain name). This step is needed to give the server the return mail address for returning errors and reporting messages.
- The server responds with code 250 or some other appropriate code.
- The client sends the RCPT TO (recipient) message, which includes the mail address of the recipient.
- The server responds with code 250 or some other appropriate code.
- The client sends the DATA message to initialize the message transfer.
- The server responds with code 354 (start mail input) or some other appropriate message.

- The client sends the contents of the message in consecutive lines. Each line is ter- minated by a two-character end-of-line token (carriage return and line feed). The message is terminated by a line containing just one period.
- The server responds with code 250 (OK) or some other appropriate code.

4.3 Connection Termination

After the message is transferred successfully, the client ter- minates the connection. This phase involves two steps.

- The client sends the QUIT command.
- The server responds with code 221 or some other appropriate code.

5 Code

5.1 Server

```
    import socket
import sys

HOST = '127.0.0.1'
PORT = 3000

serverSocket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
serverSocket.bind((HOST, PORT))
serverSocket.listen(5)

domains = {}
sender_mail_address = ""
recipient_mail_address = ""
while True:
    conn, addr = serverSocket.accept()
    conn.send('220: Service ready'.encode())
    # TODO: send 421 if server is not ready
    request = conn.recv(1024).decode()
    if(request[:4] == 'HELO'):
        domains[conn] = request[5:]
```

```
        conn.send('250: Request command completed'.encode())

    request = conn.recv(1024).decode()
    if(request[:9] == 'MAIL FROM'):
        sender_mail_address = request[10:]
        conn.send('250: Request command completed'.encode())

    request = conn.recv(1024).decode()
    if(request[:7] == 'RCPT TO'):
        recipient_mail_address = request[10:]
        conn.send('250: Request command completed'.encode())

    request = conn.recv(1024).decode()
    if(request[:4] == 'DATA'):
        conn.send('354: Start mail input'.encode())


    request = conn.recv(1024).decode()
    email_data = request[:]
    conn.send('250: OK'.encode())



    request = conn.recv(1024).decode()
    if(request[:4] == 'QUIT'):
        conn.send('221: Service closing transmission channel'.encode())

    conn.close()
    break

serverSocket.close()
```

5.2 Client

```
    import socket
import sys

SERVER_HOST = '127.0.0.1'
SERVER_PORT = 3000
```

```
clientSocket = socket.socket()
clientSocket.connect((SERVER_HOST, SERVER_PORT))

def printMessageFromServer():
    message = clientSocket.recv(1024)
    print (message.decode())

print('=========================================')
print ('HELO: <domain-name>')
print('MAIL FROM: <mail-address-of-sender>')
print('RCPT TO: <mail-address-of-recipient>')
print('DATA')
print('the contents of the message')
print('QUIT')
print('=========================================')

print("\n")

printMessageFromServer()

request = input()
if (request[:4] == 'HELO'):
    clientSocket.send(request.encode())
printMessageFromServer()

request = input()
if (request[:9] == 'MAIL FROM'):
    clientSocket.send(request.encode())
printMessageFromServer()


request = input()
if (request[:7] == 'RCPT TO'):
    clientSocket.send(request.encode())
printMessageFromServer()

request = input()
if (request[:4] == 'DATA'):
    clientSocket.send(request.encode())
printMessageFromServer()
```

```
print()
delim = ""
request = ""
data = []
while True:
    line = input()
    if line == '.':
        break;
    else:
        data.append(line)

request = '\n'.join(data)
clientSocket.send(request.encode())
printMessageFromServer()

request = input()
if (request[:4] == 'QUIT'):
    clientSocket.send(request.encode())
printMessageFromServer()

clientSocket.close()
```

6 Output

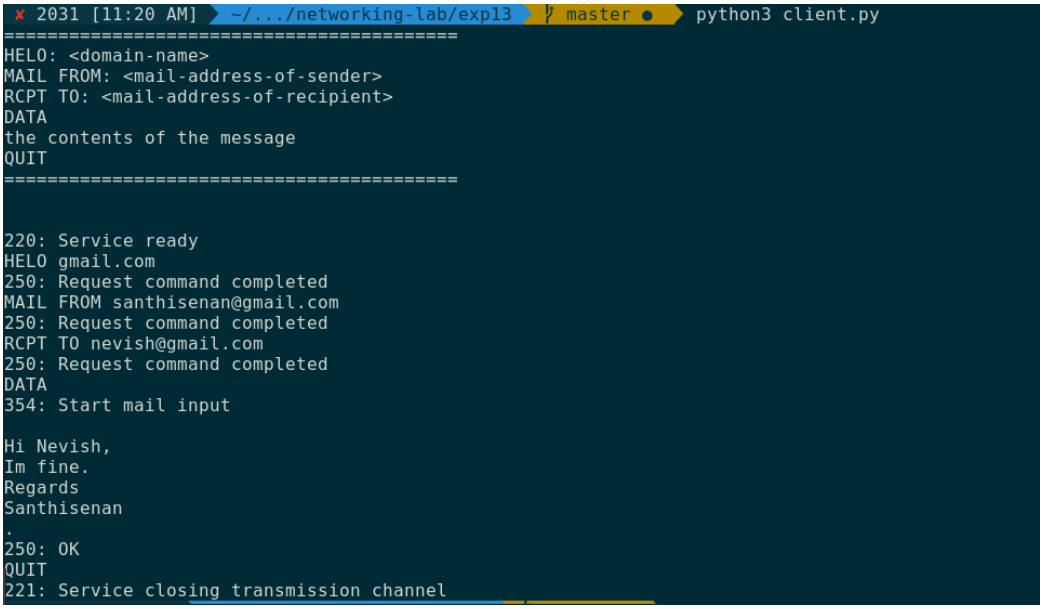

Figure 4: Output