Project Report

CS 4759

Myles Hann
Chris Healey

As in discussed in the proposal, our task in this project was to develop a simple mail client, that sends an email to any recipient, as well as an image attachment. This mail client was done with the Python language. It uses SMTP protocol and a TCP connection, which provides reliable data transfer. When the script is started, a GUI will display to the user. With this you will be able to enter the information needed to use the mail client we created.

Using socket programming, we created a connection to a mail server, for our example we used gmail for demonstration. This is where we began to experience our first troubles. We managed to connect to a locally created mailserver using a program for windows called “Free smtp server”. Initially this connected using port 25. Through the localhost connection, we were able to send emails with text to any recipient without authentication of emails. After this we began to research how to connect to a gmail mail server.

When attempting to connect to gmail, our first troubles were attempting to use port 25. Testing with telnet commands, we could not connect to gmail through port 25, and was also printing error messages in our terminal. We found out that this port is usually blocked by the Internet service provider in order to prevent spamming by its customer. With some research we tried two other port connections, 465 and 587. Port 587 was the successful winner. Using port 587 we were able to successfully create a socket connection to the gmail server.

Therefore, this also created more errors. Although we were using the SMTP commands discussed in the RFC manual, HELO, MAIL FROM, RCPT TO, and DATA. We could not successfully send any emails. Therefore we also needed to add a STARTTLS command, which initiates a secured socket connection for logging to a public mail server. Thus with the use of STARTTLS, the HELO command needed to be resent. Followed by AUTH LOGIN.

When AUTH LOGIN is sent using sockets, a successful 334 message would be sent to the user. If the 334 message is received, they would be able to input the email they are logging in with. However, for encryption purposes, this email must be encoded using base64, which was a python library we imported.

Since we do not want the user to directly type in a encoded base64 message for logging in, we let them type there regular email address in via the GUI provided, and the encoding is done when the client socket sends the information to the server.

After a successful encoded email has been sent, another 334 status message will provide the user with an opportunity to input their password for the mail account, which also encoded with base64 when the client socket sends the information to the server.

Providing the authentication of a gmail account was successful, the client socket can continue to send SMTP RFC commands as needed.

MAIL FROM: The address of the sender
RCPT: The address of the recipient (Also used for the CC addresses)
DATA: Essentially the data needed for sending an email

In the DATA command, we needed to provide information such as the subject and the message. For additional header information, we realized that we had to implement the same as adding a subject header. Within this, MIME (Multi-Purpose Internet Mail Extension) are also setup to attach files.

Then the end message command would need to be sent through the socket.

Briefly reviewing the RFC and the commands provided, we implemented the end of sending a message by making a new line and adding period ("."), which caused many errors in our project. Digging deeper into the RFC, and reading the syntax, we came to the realization that the period not only needed to be on a separate line, but has to be the **only** single character on that line. For example in our code we created:

endmsg = "\r\n. \r\n" ,

which caused errors because of the extra white space. Replacing this with:

endmsg = "\r\n.\r\n"

Fixed this error, thanks to the specific details of the RFC which was slightly ignored in the first few days of implementation.

After completing the gmail server authentication, and successfully being able to send plain text messages to any recipient, we decided to upgrade our GUI with a file chooser, which enables to user the select files to attach. In our case, this project is currently limited to only sending JPEG images. Because of complications getting our jpg files to send, and also time constraints, we have decided to just attach jpg image files.

In order to be able to send images, we need to add another command into our DATA stream. This would be known as "MIME setup". Using a separate files, we imported the text, and encoding syntax needed to send attachments through telnet.

To finish this accomplishment, once the file is chosen, and opened using python, it also must be encoded using base64 encoding. We tried multiple approaches that did originally work, until we found out that there needed to be an exact syntax for line breaks after the initial mime setup. There needs to be one completely blank line in between the mime setup, and encoded image text. Thus, when client sockets send this information to the server, it will read the syntax provided as an attachment, and decode the image into a JPEG file, with the name we provided in the setup. In our case, we call the file picture.jpg.

Thus our final product for this project, provides some basic features, but covers the important concepts of RFC standards, while using socket programming. Using the client socket to send commands and data, we provide the user with a simple GUI that will allow them to login with their gmail account, provide an recipient email address, choose the subject, a message body for sending plain text, and also a file chooser for optionally sending a JPEG attachment. When the user hits send, some information of what is happening with the socket responses will displayed in the terminal. Such as responses, and errors. At the moment this client only accepts data for JPEG images, but the idea of how to send attachments using MIME syntax is there, and can be edited in the future to send any file extension.

Also, with some additional work, we successfully implemented the ability to be able to CC the email to other recipients. Initially with some research of the RFC, we found that (and assumed) it was done the same way as sending the Subject header (see code for more details). Although the CC showed up in the sent/inbox emails of the sender and original recipient, the CC would not get sent to the emails provided. However, we realized after that the headers sent in the DATA command were just for visual effects, and had no real effect to the sending process. So, we simply added any CC recipient as another RCPT TO command.

We have implemented the following SMTP commands in order using RFC standards:

- HELO
- STARTTLS
- HELO (Second time around for STARTTLS)
- AUTH LOGIN
- username insert (base64)
- password insert (base64)
- MAIL FROM
- RCPT TO
- DATA
- Subject:
- message
- mime
- encoded image (base64)
- QUIT

For further testing, please use your gmail or MUN email address and send as many emails as you wish.