



## Email Client Application

### Objective

Your task is to develop a Python-based (or Java-based) **Email Client Application** that can send and receive emails. The application should be able to establish a TCP connection with a mail server, dialogue with the mail server using the **SMTP** and **IMAP** protocols, send an e-mail message to a recipient via the mail server, fetch the latest email from the mailbox, and finally close the TCP connection with the mail server.

In addition, the application should communicate with a simple **TCP Notification Server**, and you will use **Wireshark** to capture and analyze the network packets generated during the whole process.

### Part 1 — Email Client (SMTP + IMAP)

Your application should be able to:

1. Establish a TCP connection to a mail server.
2. Send emails using the **SMTP protocol**.
3. Receive the latest email using the **IMAP protocol**.
4. Interact correctly with the server following proper protocol sequences.
5. Close the connection properly.

### Requirements

#### 1. Sending Emails (SMTP)

Your application must include a function that:

- Uses SMTP to send an email.
- You should use `smtplib` and `email.mime.multipart` libraries in Python for this purpose.
- The email should include:
  - Subject.
  - Body text.
- The sender's email, password, recipient's email, subject, and body should be parameters to your function.
- Prints a clear success or failure message.



## 2. Receiving Emails (IMAP)

Your application must include a function that:

- Connects to a mail server using IMAP.
- You should use the `imaplib` and `email` libraries in Python to fetch the latest email from the inbox.
- Displays at least:
  - Email subject.
  - Email body.
- Prints a clear success or failure message.

## 3. Mail Server for Testing

To protect your personal email accounts and to maintain privacy, you must use a testing mail server such as:

- **mail.tm (recommended).**
- Any temporary or educational email server.

## 4. Error Handling

Your code should include proper exception handling.

If any error occurs during sending or receiving, your program should:

- Catch the exception.
- Print a meaningful error message.
- Continue execution without crashing.

## 5. Code Structure

Your code should:

- Be clean and readable (well-structured).
- Contain comments (where necessary) explaining key steps.
- Separate functionality into logical functions.

## 6. Testing

Test your application by:

- Sending emails to different user accounts and through different servers.



## Part 2 — TCP Notification Server

You must implement a **simple TCP server** that runs locally and prints notification messages sent by your email client.

### **1. Server Requirements**

- Listens to a chosen port (e.g., **9999**).
- Accepts a TCP connection from the email client.
- Receives a short text message such as:
  - “Email Sent Successfully”.
  - “Email Received Successfully”.
  - “Email Sending Failed”.
- Prints the message on the server side.

### **2. Client-Side Integration**

Your email client (SMTP/IMAP program) must:

- Connect to the notification server after sending/receiving an email.
- Send one short notification message.
- Close the connection properly.

## Part 3 — Wireshark Packet Capture & Analysis

You must use **Wireshark** to capture and analyze packets generated by your program.

### **1. Required Packet Filters**

Use the following Wireshark filters:

**SMTP traffic:** `tcp.port == 465 or tcp.port == 587.`

**IMAP traffic:** `tcp.port == 993.`

**Notification server traffic:** `tcp.port == 9999.`

### **2. Required Screenshots**

You must capture and include screenshots of:

- **TCP 3-way handshake** (SYN, SYN/ACK, ACK).
- **SMTP packets** (e.g., LOGIN, MAIL FROM, RCPT TO, DATA...).
- **IMAP packets** (LOGIN, SELECT, FETCH...).
- **TCP notification message packets.**



### 3. Required Analysis (For Part4)

For each operation (SMTP send, IMAP fetch, notification send):

**Provide:**

- Number of packets sent and received.
- Total bytes transmitted.
- Latency measurements.
- Observations about TLS encryption (if applicable).
- Explanation of the packet flow.

## Part 4 — Performance Measurements

You must measure:

### 1. Latency

Compute the total time required to:

- Send an email.
- Fetch an email.

### 2. Packet Count

From Wireshark:

- Count total sent packets.
- Count total received packets.

### 3. Throughput

Compute:  $\text{Throughput} = \text{total\_bytes} / \text{total\_time}$ .

### 4. Summary Table (Include in your report a final table like below)

Operation	Time (s)	Packets	Bytes	Throughput
SMTP Send				
IMAP Fetch				
Notification TCP				



## **Part 5 — Bonus**

- **GUI:** Develop a graphical user interface for your application. The GUI should allow users to input their email, password, recipient's email, subject, and body, and have buttons to send and receive emails. [use [tkinter](#)]
- **Push Notifications:** Instead of printing the latest email to the console, implement a push notification system that alerts the user when a new email arrives in the mailbox. [use [Plyer](#)]

## **Important Notes**

1. SMTP on ports **465/587** requires TLS/SSL.  
IMAP on **993** is always SSL.  
You will get errors if you don't wrap sockets properly. So, to avoid the message "Authentication error!", You can use:
  - `smtplib.SMTP_SSL` or `SMTP.starttls()`.
  - `imaplib.IMAP4_SSL`.
2. Use `email.message_from_bytes()` to decode the IMAP FETCH response.
3. For the time measurement, you can use:
  - Python `time.time()`.
  - Wireshark round-trip timing.



## **Deliverables**

### **You must submit:**

- A working implementation of the email client application (Your Project) developed in either Python or Java, including scripts (or source files) for sending and receiving emails.
- A comprehensive technical report (PDF File) documenting your implementation, usage instructions, dependencies, and testing results (with all supporting screenshots).

Through this form link: <https://shorturl.at/fiAh8>.

### **Policy:**

- The deadline for delivery is **Sunday 21/12/2025 at 10 am**.
- A discussion will be held in your lab time as in the shared distribution\_files.
- The discussion will be based on the files submitted by the specified deadline. All project dates will be carefully reviewed, and any files modified after the deadline will not be accepted, even if delayed by one minute.
- You must work in groups of 4 (Same Group).
- Your code should be clean, readable, and the report should be supported with screenshots.
- Cheating will be severely penalized.
- Any code generated primarily by AI tools **will receive a grade of zero**.
- No late submission is allowed.