Introduction to Computer Networking

Second Assignment

Academic year 2025-2026

Abstract

In this assignment, students will have to implement a mail server using Java Sockets.
The server uses the SMTP, POP3 and IMAP protocols to allow sending, storing and retrieving emails. Through this project, students will explore the mechanisms behind email delivery, retrieval, and synchronization, as well as key networking concepts such as DNS, concurrent request handling, and text-based protocol parsing.
Students will work in teams of 2 students.
The **Hard Deadline** for this project is December 14th, 2025.

Sections 1 to 3 define your assignment objectives. Sections 4 and further are an executive summary of the technologies we use.

1 Project Overview

In this project, students will implement a Mail Server using Java Sockets that supports the three main mail protocols: **SMTP**, **POP3**, and **IMAP**. The project will take place in a simulated network environment composed of three independent domains, each with its own DNS server and Mail Server.

Each mail server must therefore be capable of:

- Sending emails to users within its own domain and other domains via SMTP
- Receiving emails via SMTP and storing messages locally in user mailboxes
- Allowing clients to retrieve messages through POP3
- Allowing clients to synchronize mailboxes through IMAP
- Resolving remote domains using DNS to locate the corresponding mail servers.

Each mail server will thus act both as a Mail Transfer Agent (MTA) for inter-domain communication and as a Mail Delivery/Access Agent (MDA/MAA) for local storage and retrieval. Together, the three domains will form a complete mail exchange system where messages can be routed, delivered, and accessed across the network using standard Internet mail protocols.

Students do not need to implement the mail client that communicates with their servers instead they will use **Mozilla Thunderbird** client and configure it to use the appropriate mail server.

1.1 Network Configuration

The simulated network will be deployed using **Docker** and defined through a provided `docker-compose.yml` file. This configuration automatically creates the virtual network, assigns IP addresses to each component, configures the DNS servers, and launches the Mail Servers with the students' Java implementations.

A schematic representation of the environment is shown in Figure 1.

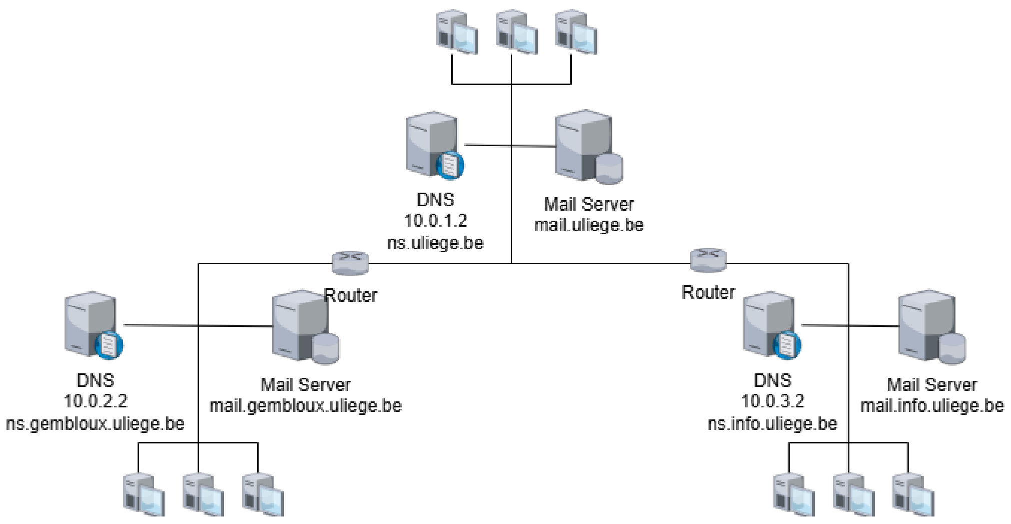

Figure 1: Overview of the network deployed with Docker

The network consists of three independent domains, each with its own DNS server and Mail Server:

Domain	**Mail Server**	**Assigned Range**
`uliege.be`	`mail.uliege.be`	10.0.1.0/24
`gembloux.uliege.be`	`mail.gembloux.uliege.be`	10.0.2.0/24
`info.uliege.be`	`mail.info.uliege.be`	10.0.3.0/24

Each DNS server will contain the appropriate records for its own domain, including at least one **A record** for its Mail Server and an **MX record** to identify the host responsible for receiving mail for that domain. For example:

```
uliege.be.          IN  MX  10 mail.uliege.be.
mail.uliege.be.     IN  A   10.0.1.10
```

The DNS servers are configured hierarchically. The DNS servers of the subdomains (`gembloux.uliege.be` and `info.uliege.be`) will forward any unresolved queries to the main DNS server of `uliege.be`. This setup ensures that inter-domain name resolution functions correctly across the entire simulated network.

Additionally, the DNS server of `uliege.be` will include the following record to ensure that messages destined for `info.uliege.be` are routed through its Mail Server:

```
info.uliege.be.       IN  MX  10 mail.uliege.be.
```

This configuration makes `mail.uliege.be` act as a relay for messages between the subdomains, ensuring that all inter-domain traffic passes through a common point of control. For most parts of the project, students should be able to test their server on their own machine with a **Mozilla Thunderbird** client. However, to test the full configuration involving all three domains, students will use a Thunderbird clients running inside a Docker containers attached to the same virtual network.

1.2 Launch

Each Mail Server will be executed inside its own **Docker container**, as defined in the provided `docker-compose.yml` file. All the Java source files required for the Mail Server must be located together in a folder named `MailServer`, placed in the same directory as the Docker compose file.

To deploy the full environment, students must run the following command in the directory containing the `docker-compose.yml` file:

```
docker compose up
```

The Docker configuration automatically builds and launches each container, creates the virtual network, assigns the appropriate IP ranges, starts the DNS servers, and runs the Mail Servers with the students' Java implementations.

Within each container, the Mail Server is started using the following command:

```
java MailServer domain maxThreads
```

The `domain` parameter specifies the domain name of the Mail Server instance (e.g. `uliege.be`) while the `maxThreads` parameter defines the maximum number of concurrent connections that the server may handle simultaneously.

1.3 SMTP

SMTP (Simple Mail Transfer Protocol) (cfr. 4) is the protocol used for sending and transmitting emails between clients and mail servers, as well as between mail servers themselves. Although SMTP is primarily used to send messages from the client to the server and between servers, it is also responsible for receiving and saving emails on the server when they are delivered.

SMTP is responsible for routing messages based on the destination domain. The steps involved are as follows:

- Local Delivery: If the recipient's domain matches the server's domain, the email is stored in the user's mailbox locally.
- Forwarding to Known Domains: If the recipient's domain is known (i.e., it has a pre-configured forwarding rule), the server forwards the email to the correct server.

- Forwarding to Unknown Domains: If the recipient's domain is not known, the server performs a **DNS lookup** to find the appropriate SMTP server to forward the email.

1.3.1 DNS lookup

When sending emails to remote domains, the Mail Server must perform a DNS lookup to obtain the IP address of the destination Mail Server, typically through its MX record. Students may rely on the system's DNS client to perform these lookups using the `dig` command. This approach is sufficient to meet the project requirements.

Alternatively, students may implement their own DNS client in Java that directly queries DNS servers using UDP. Implementing such a client correctly will be rewarded with a bonus point.

1.3.2 SMTP Commands and Status Codes

Each Mail Server must support a minimal set of SMTP commands to ensure proper communication between clients and other Mail Servers. Each Mail Server must support at least the following SMTP commands:

```
HELO <domain>, MAIL FROM:<address>, RCPT TO:<address>, DATA, QUIT
```

SMTP uses three-digit status codes to indicate the result of each command. Each code informs the client whether the command was successfully executed or if an error occurred. The server must handle and return at least the following status codes:

```
220, 250, 354, 421, 501, 550, 553
```

1.4 POP3

POP3 (Post Office Protocol version 3) (cfr. 5) is used by clients to retrieve and delete messages stored on the Mail Server. A POP3 session consists of three states: authentication, transaction, and update. The server must support at least the following commands and status codes.

1.4.1 POP3 Commands and Status Codes

Each Mail Server must support a minimal set of POP3 commands to ensure proper communication with clients. Each Mail Server must support at least the following POP3 commands:

```
USER <username>, PASS <password>, STAT, LIST, RETR <id>, DELE <id>, QUIT
```

POP3 uses textual replies beginning with either `+OK` or `-ERR` to indicate success or failure. Each response line must start with one of these prefixes.

1.5 IMAP

IMAP (Internet Message Access Protocol) (cfr. 6) allows clients to access, manage, and synchronize mailboxes while keeping all messages stored on the Mail Server. Unlike POP3, IMAP supports folder management and the concept of persistent message identifiers (UIDs), which enable consistent synchronization across multiple clients.

Students **must implement `IMAP4rev1`** in order to be compatible with modern mail client such as Thunderbird.

1.5.1 IMAP Commands and Status Codes

Each IMAP command begins with a tag (e.g. `A1`) that uniquely identifies the request. These commands follow the structure:

```
<tag> <command> <arguments>
```

The following list of commands goes beyond the minimal IMAP requirements but represents the essential subset required for compatibility with Mozilla Thunderbird, which will be used for testing. Students must handle properly at least the single required `INBOX` mailbox. Multiple mailbox management is optional for a bonus point. The blue commands of the following list are therefore optional.

Commands

```
## Session and Capability
CAPABILITY
NOOP
LOGIN <username> <password>
LOGOUT
## Mailbox and Folder Management
LIST "" "*"
LSUB "" "*"
SELECT <mailbox>
CREATE <mailbox>
DELETE <mailbox>
RENAME <old> <new>
SUBSCRIBE <mailbox>
UNSUBSCRIBE <mailbox>
## Mailbox Selection and Message Handling
UID FETCH <range> (FLAGS BODY[] RFC822.SIZE)
UID STORE <range> <data item name> <value>
UID COPY <range> <mailbox>
EXPUNGE
CLOSE
```

Responses

Each server response must start with the same tag used in the request, followed by a status keyword (`OK`, `No` or `BAD`) indicating the result of the command.

1.6 Users and Authentication

For simplicity, user authentication will rely on a predefined list of users shared across the Mail Servers. Each user account belongs to one of the three domains and uses the same password: `password`. The predefined users are: `dcd@gembloux.uliege.be`, `vj@gembloux.uliege.be`, `dcd@info.uliege.be`, `vj@info.uliege.be`, `dcd@uliege.be`, `vj@uliege.be`.

All authentication requests in POP3 and IMAP must validate the username and password against this predefined list. Each Mail Server should only authenticate users belonging to its own domain and authentication failures should return the appropriate error message.

1.7 Mail Storage

The internal organization of mail storage is left to the students' discretion. Each Mail Server must, however, maintain a persistent structure that allows both POP3 and IMAP access to the same user mailbox.

The default mailbox used by POP3 must correspond to the user's primary `INBOX` folder as seen through IMAP.

Students are free to choose how emails and mailboxes are represented on disk, provided that the following metadata is correctly managed and preserved:

- A unique and persistent `UID` assigned to each message;
- The set of message flags: `\Seen`, `\Answered`, `\Flagged`, `\Deleted`
- The mailbox-level `UIDVALIDITY` value, which must remain constant as long as the mailbox is unchanged.

2 Report

Your program will be completed by a .pdf report that follows the provided template and doesn't exceed 6 pages. The report will address the following points:

Software architecture: How have you broken down the problem to come to the solution? Name the major classes responsible for requests processing.

Mail forwarding and resolution: For an email form the `gembloux.uliege.be` to the `info.uliege.be` with empty caches, describe and explain each steps (SMTP and DNS) required for this email to be delivered.

Multi-thread coordination: How have you synchronized the activity of the different threads?

Limits: Describe the limits of your program, esp. in terms of robustness.

Possible Improvements: This is a place where you're welcome to describe missing features or revisions of these specifications that you think would make the server richer or more user-friendly.

3 Guidelines

- You will implement the programs using Java 1.17, with packages `java.lang`, `java.io`, `java.net`, `java.awt`, `javax.imageIO` and `java.util`[1],
- You will ensure that your program can be terminated at any time simply using CTRL+C, and avoid the use of ShutdownHooks
- You will ensure your main class is named `MailServer`, located in `MailServer.java` at the root of the archive, and does not contain any `package` instruction.
- You will not cluster your program in packages or directories. All java files should be found in the same directory.

Submissions that do not follow these guidelines could be ignored during evaluation.

Your commented source code alongside the report will be delivered no later than 14th December 2025 (**Hard deadline**) to the Montefiore Submission platform (`https://submit.montefiore.uliege.be/`) as a .zip package.

[1] Don't be shy and ask for more packages if you feel like there's one you would really like to use

4 SMTP

SMTP (Simple Mail Transfer Protocol) is the protocol used to transfer electronic mail between clients and Mail Servers, or between Mail Servers themselves. Defined initially in RFC 821 and updated by RFC 5321, SMTP operates as a text-based protocol over TCP, usually on port 25. It follows a request-response model where each command sent by the client is acknowledged by the server with a numeric status code and a short textual message.

4.1 Session Structure

A typical SMTP session begins when a client connects to the Mail Server on port 25 and receives a greeting message. The client then identifies itself and initiates the transmission of one or more messages. All exchanges are plain text lines terminated by `<CRLF>`.

A minimal example of a session between a client and the Mail Server is shown below. In this example, the `dcd@uliege.be` user connects his mail client to the `mail.uliege.be` Mail Server and sends an email to `vj@uliege.be`.

```
S: 220 mail.uliege.be Service ready
C: HELO uliege.be
S: 250 mail.uliege.be greets uliege.be
C: MAIL FROM:<dcd@uliege.be>
S: 250 OK
C: RCPT TO:<vj@uliege.be>
S: 250 OK
C: DATA
S: 354 End data with <CRLF>.<CRLF>
C: From: dcd@uliege.be
C: To: vj@uliege.be
C: Subject: Test message
C:
C: Hello,
C: This is a test message sent within uliege.be.
C: .
S: 250 OK Message accepted for delivery
C: QUIT
S: 221 Bye
```

4.2 Commands

Below is the list of commands students must implement for the project with a small description of their purpose. For more details, students are encourage to consult RFC 5321 or make their own research.

Command	Description
`HELO <domain>`	Start session, identify client
`MAIL FROM:<addr>`	Set sender address
`RCPT TO:<addr>`	Add recipient address
`DATA`	Send message body, end with "."
`QUIT`	Close connection

Table 1: Minimal SMTP command

4.3 Message Format

SMTP transmits messages as plain text, where each message consists of two parts: the **header** and the **body**. Headers contain structured information such as the sender, recipient, subject, and date, and are separated from the body by an empty line. Lines are terminated using the sequence `<CRLF>`. When the message is transmitted using the `DATA` command, the end of the message is marked by a single line containing a dot (`.`) on its own.

An example of an email message sent via SMTP is shown below:

```
From: dcd@uliege.be
To: vj@uliege.be
Subject: Test message
Date: Tue, 4 Nov 2025 10:23:00 +0100
Content-Type: text/plain; charset=UTF-8

Hello,
This is a test message sent within uliege.be.
```

5 POP3

POP3 (Post Office Protocol version 3) is a simple protocol used by mail clients to retrieve messages stored on a Mail Server. Defined in RFC 1939, POP3 operates as a text-based protocol over TCP, usually on port 110. It follows a text-based request-response model similar to SMTP.

Clients connect to the server, authenticate with their credentials, retrieve messages, and optionally delete them from the mailbox.

5.1 Session Structure

A POP3 session consists of three main states: authentication, transaction, and update. During the authentication state, the client identifies itself and provides a username and password. Once authenticated, the session enters the transaction state, where the client can list, retrieve, and mark messages for deletion. When the client sends the `QUIT` command, the session transitions to the update state, during which any messages marked for deletion are permanently removed.

A typical POP3 session between a client and the Mail Server is shown below. In this example the `dcd@uliege.be` user connects his mail client to the `mail.uliege.be` Mail Server and authenticates. Then, he lists his emails and retrieves the first one and deletes it from the server.

```
S: +OK POP3 server ready
C: USER dcd@uliege.be
S: +OK
C: PASS password
S: +OK Mailbox locked and ready
C: STAT
S: +OK 2 680
C: LIST
S: +OK 2 messages (680 octets)
S: 1 320
S: 2 360
S: .
C: RETR 1
S: +OK 320 octets
S: From: vj@uliege.be
S: To: dcd@uliege.be
S: Subject: Test message
S:
S: Hello,
S: This is a message retrieved through POP3.
```

```
S: .
C: DELE 1
S: +OK Message marked for deletion
C: QUIT
S: +OK Goodbye
```

5.2 Commands

Below is the list of commands students must implement for the project with a small description of their purpose. For more details, students are encourage to consult RFC 1939 or make their own research.

Command	**Description**
`USER <name>`	Provide username
`PASS <password>`	Provide password
`STAT`	Show message count and total size
`LIST`	List messages and sizes
`RETR <id>`	Retrieve a message
`DELE <id>`	Mark a message for deletion
`QUIT`	End session and apply deletions

Table 2: Minimal POP3 commands

6 IMAP

IMAP (Internet Message Access Protocol) allows mail clients to access, manage, and synchronize messages stored on a Mail Server. Unlike POP3, which downloads and optionally deletes messages, IMAP keeps emails on the server, enabling multiple clients to access the same mailbox concurrently. Initially standardized in RFC 1064, it has been refined many times, notably in RFC 3501, which defines the IMAP4rev1 version required for this project. It operates over TCP, typically on port 143, and exchanges text-based commands and responses that include tags identifying each client request.

6.1 Session Structure

An IMAP session begins when the client connects to the server and receives an initial greeting. The client usually starts by sending a `CAPABILITY` command to discover the server's supported features, followed by authentication using `LOGIN`. After authentication, the client can list mailboxes, select one to access its messages, and perform operations such as fetching messages, modifying flags, copying messages, or synchronizing state.

A typical IMAP session between a client and the Mail Server might look like this:

```
S: * OK IMAP server ready
C: A1 CAPABILITY
S: * CAPABILITY IMAP4rev1 UID
S: A1 OK CAPABILITY completed
C: A2 LOGIN dcd@uliege.be password
S: A2 OK LOGIN completed
C: A3 LIST "" "*"
S: * LIST (\HasNoChildren) "/" INBOX
S: A3 OK LIST completed
C: A4 SELECT INBOX
S: * 2 EXISTS
S: A4 OK [UIDVALIDITY 12345] SELECT completed
C: A5 UID FETCH 1:* (UID FLAGS BODY[])
S: * 1 FETCH (UID 1001 FLAGS (\Seen) BODY[] {120}
S: From: vj@uliege.be
S: To: dcd@uliege.be
S: Subject: Meeting reminder
S:
S: Reminder: Meeting at 10:00.
S: )
S: A5 OK FETCH completed
C: A6 LOGOUT
S: * BYE IMAP server logging out
S: A6 OK LOGOUT completed
```

In this session, the client first requests the server capabilities, then authenticates, lists available mailboxes, selects the `INBOX`, retrieves message content and flags using `UIDs`, and finally logs out. Each command begins with a unique tag (e.g. `A1`, `A2`), which the server repeats in its final response to that command. The server must advertise at least `IMAP4rev1` in its capability response, as this is the version required by Thunderbird.

6.2 Commands

Below is the list of commands students must implement for the project with a small description of their purpose. For more details, students are encourage to consult RFC 3501 or make their own research.

Command	**Description**
`CAPABILITY`	List server features
`NOOP`	Keep connection alive
`LOGIN <user> <pass>`	Authenticate user
`LOGOUT`	End IMAP session
`LIST "" "*"`	List all mailboxes
`LSUB "" "*"`	List subscribed mailboxes
`SELECT <mailbox>`	Open mailbox
`CREATE <mailbox>`	Create mailbox
`DELETE <mailbox>`	Delete mailbox
`RENAME <old> <new>`	Rename mailbox
`SUBSCRIBE <mailbox>`	Subscribe to mailbox
`UNSUBSCRIBE <mailbox>`	Unsubscribe mailbox
`UID FETCH <range> (FLAGS BODY[] RFC822.SIZE)`	Fetch message data
`UID STORE <range> <item> <value>`	Change message flags
`UID COPY <range> <mailbox>`	Copy messages
`EXPUNGE`	Remove deleted messages
`CLOSE`	Close mailbox, expunge

Table 3: Common IMAP commands for a minimal client/server

6.3 UIDs, UIDVALIDITY and Mailboxes

Each mailbox in IMAP (for instance, the required `INBOX` folder) has its own independent list of emails. In IMAP terminology, one mailbox corresponds to one folder. Every message within a mailbox is assigned a unique identifier (`UID`), a numeric value that must be unique and persistent within that mailbox.

When new messages are added, they receive new, higher `UIDs`, while messages that are deleted keep their `UID` until they are permanently removed.

The persistence of UIDs is guaranteed by a mailbox attribute called UIDVALIDITY. This value uniquely identifies a mailbox instance. If a mailbox is deleted and recreated, its UIDVALIDITY must change to signal to clients that all previous UIDs are no longer valid.

A popular techniques to manage UIDs and UIDVALIDITY is to store this information in a metadata file for each folder/mailbox. For UIDs, they are sometimes encoded in the file name (also for flags sometimes).

6.4 Fetching Messages

Most IMAP requests are straightforward enough. In this section, we only emphasize one of the most important commands used by modern mail clients: UID FETCH.

The UID FETCH command allows clients to retrieve messages or specific parts of messages based on their UID values. The general syntax is:

```
UID FETCH <range> (<data items>)
```

The range specifies which messages to fetch (for example, 1:* means all messages), and the data items define what information to retrieve. For instance:

```
C: A5 UID FETCH 1:* (UID RFC822.SIZE FLAGS BODY.PEEK[HEADER.FIELDS (From To)])
S: * 1 FETCH (UID 1 RFC822.SIZE 120 FLAGS (\Seen) BODY[HEADER.FIELDS (From To)] {39}
S: From: vj@uliege.be
S: To: dcd@uliege.be
S: )
S: * 2 FETCH (UID 2 RFC822.SIZE 200 FLAGS () BODY[HEADER.FIELDS (From To)] {39}
S: From: vj@uliege.be
S: To: dcd@uliege.be
S: )
S: A5 OK FETCH completed
```

Each line beginning with an asterisk (*) is an untagged server response describing a message. The number following the asterisk identifies the message's sequence number in the selected mailbox. The keyword FETCH is followed by parentheses enclosing pairs of data item / value entries corresponding to the client's requested items and their corresponding values.

The values of some data items are sometimes simple values (e.g., UID, RFC822.SIZE) and others are more complex. For instance:

- FLAGS returns a list of message flags, enclosed in parentheses. For example (\Seen \Flagged).
- BODY[HEADER.FIELDS (...)] retrieves content from the body with some filtered specified in the brackets. The value starts with a size indicator in braces (e.g., {39}), specifying the number of bytes that are part of the requested body. It is followed by the requested filtered body.

7 Thread Pool

When a server accepts a connection, it usually invokes a new *thread* that will handle that connection, so that the server can go back to listening to the port. This is very convenient to guarantee a certain level of accessibility but also has a flaw.

The *(Distributed) Denial of Service* (or(D)Dos) is an attack that targets servers with this kind of behavior. In this attack, one (for DoS) or several (for DDoS) machines initiate many bogus connections. If the server launches a new thread for each of these connections, it will soon encounter performance problems or even crash.

To circumvent this problem, one can use a *Thread Pool* that limits the number of threads that can be executed concurrently, while keeping the other jobs on hold until new threads become available. This thread pool can be implemented through the use of `java.util.concurrent.Executors` by calling the `newFixedThreadPool(int maxThreads)` method to create a fixed-size pool of *maxThreads* threads and calling the `execute(Runnable worker)` method to assign the work represented by *worker* to one of a thread in the pool, when available.

You can (and are encouraged to) use a thread pool in your assignment.

8 Plagiarism

Plagiarism is the practice of taking someone else's work or ideas and passing them off as one's own. As of ULiege regulation, plagiarism is a fraud and is sanctioned with a grade of 0/20 <u>for the course</u>. In particular, giving his/her source code to another student, or copying the source code from another student submission, past or present, is strictly prohibited. Copying from an external source is allowed if the part that is copied is well identified and the source quoted. Note that, in that case, the quoted material will not award any points in the grading. In case of suspicion of fraud, an oral examination will be conducted to determine if the project is indeed the student's work.

Good programming...