UNIVERSITY OF SCIENCE AND TECHNOLOGY OF HANOI

DEPARTMENT OF INFORMATION AND COMMUNICATIONS TECHNOLOGY

DISTRIBUTED SYTEM - PROJECT

FTP OVER MPI (AS FTP PROXY)

Group 13:

Nguyen Ngoc Lan - BA12-104

Tran The Bao - BA12-020

Tran Anh Duc - BA12-051

Truong Hoang Khanh Linh - 22BI13254

Contents

1 Introduction

FTP (File Transfer Protocol) is a standard communication protocol used for the file transfer between machines over networks. Its simplicity and reliability have made it one of the most of fundamental and widely used protocols in modern file-sharing systems. Originally, FTP-based file-transfer systems works directly between a client, which establishes a connection to the server, requests for files, and the server, which provides the files. However, in certain scenarios, communication between the client and server may be restricted or blocked due to various limitations like network configurations, security policies, etc. As a result, to ensure the file transfer operation between the client and the server, the process needs to be passed through an intermediate, or ”proxy”.

In this project, we implement the system with its client, proxy and server. By having the proxy act as a bridge between blocked network endpoints, the project develops a software system demonstrating secure and efficient file transfer operation across the network.

2 Design

2.1 Architecture

The system is designed to provide file transfer between a client and a server in scenarios where communication is restricted. It consists of the client, the proxy and the server. Each of them has different roles, working together to ensure secure and efficient file transfer.

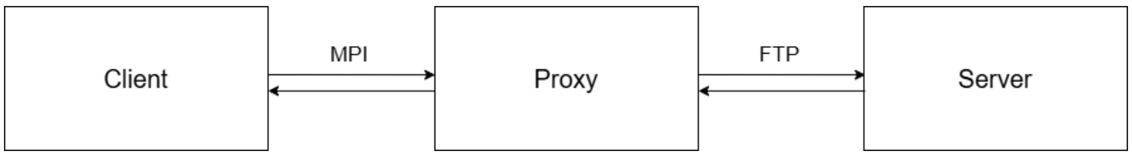


Figure 1: Architecture diagram

The client is the starting point or the sender of the file transfer process. Its primary role is to initiate file transfer requests. It is designed to interacts with the proxy using the MPI protocol, sending requests and data to the proxy.

The proxy acts as an intermediary that bridges the client and the server, allowing the system to carry out the transfer process even if they are not directly connected. It receives requests and data from the client through MPI, translates them into FTP, and forwards them to the server.

The server is operated by FTP protocol, responsible for processing requests and data, then sending responses back to the proxy, which then relays them to the client.

2.2 Communication flow

Communication flow of the system is designed to follow a structured manner:

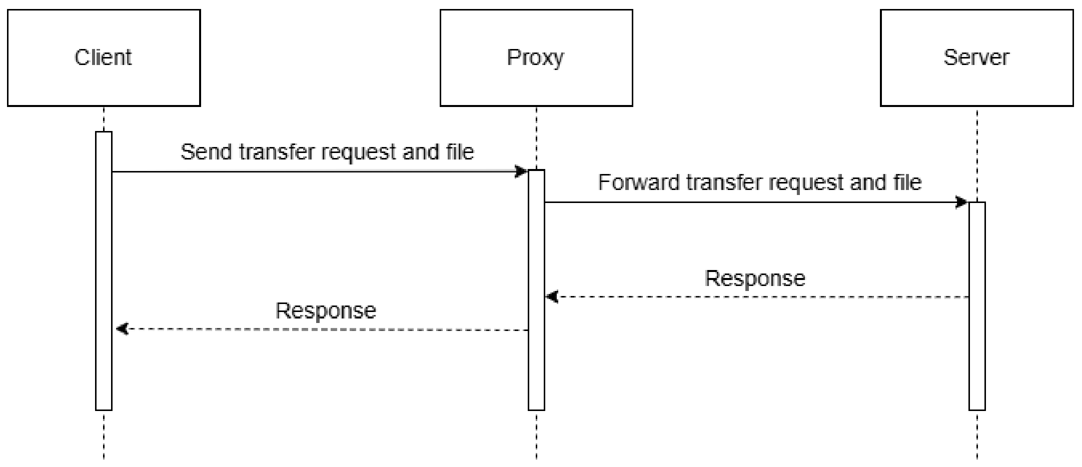

Client
Proxy
Server
Send transfer request and file
Forward transfer request and file
Response
Response

Figure 2: Sequence diagram

The communication begins with the client establishing a connection with the proxy via MPI protocol. Once the connection is formed, it sends file transfer requests along with the file data to the proxy.

Upon receiving the requests and data, the proxy requests a connection with the server using FTP protocol and forward the commands from the client to the server. With this, the proxy ensures that the file transfer requests reach the server despite restrictions.

The server then processes the requests received from the proxy. It performs necessary processes and generates responses based on requests. These responses are sent back to the proxy. After receiving the responses, the proxy relays them back to the client. This completes the communication cycle, with the client receiving the results of its file transfer requests.

3 Implementation

3.1 The client

The client is implemented using the mpi4py library (MPI for Python), enabling distributed communication between multiple processes, where each process is assigned a unique rank. Its main responsibility is to initiate file transfer requests and send files to the proxy server over HTTP.

The program consists of two main roles:

- The master process (rank 0) acts as the interface that interacts with the user. It is set to continuously listen for and handles user inputs. The user can input commands such as upload files or quit. When the user request to upload a file, the master process first checks if the specified file exists. If it does, the process sends the file to other worker processes. It also uploads the file itself to the proxy using HTTP request. If it does not, the system returns an error message. When the user request to quit, the master process signals other worker processes to terminate and exits.
- The worker processes (ranks > 0) act as secondary clients that wait for instructions from the master process. Upon receiving a file data, they independently

upload the file to the Proxy server using the same HTTP POST mechanism. If they receive a "QUIT" signal from the master process, they exit their execution after performing any necessary cleanup.

To handle file transfers, the program set up the `send_file()` function. This function reads the file in binary mode and sends it to the Proxy server's upload endpoint using an HTTP request. In case any exception occurs, the system returns an error message to the user.

Below is the code for the client:

```
from mpi4py import MPI
import requests
import os

comm = MPI.COMM_WORLD
rank = comm.Get_rank()
size = comm.Get_size()

def send_file(file_path):
    try:
        with open(file_path, "rb") as f:
            response = requests.post("http://100.68.97.89:6969/
                upload", files={"file": f})
        print(f"Server response from rank {rank}: {response.text}")
    except Exception as e:
        print(f"Error from rank {rank}: {e}")

if __name__ == "__main__":
    if rank == 0:
        print(f"Welcome to the Client! Total MPI processes: {size}"
            )
        while True:
            command = input("\nEnter a command (upload <file_path>
                / quit): ").strip()
            if command.startswith("upload "):
                _, file_path = command.split(" ", 1)
                if os.path.exists(file_path):
                    for i in range(1, size):
                        comm.send(file_path, dest=i, tag=11)
                    send_file(file_path)
                else:
                    print(f"File not found: {file_path}")
            elif command == "quit":
                print("Exiting Client.")
                for i in range(1, size):
                    comm.send("QUIT", dest=i, tag=11)
                break
            else:
                print("Invalid command. Try 'upload <file_path>' or
                    'quit'.")
    else:
        while True:
            file_path = comm.recv(source=0, tag=11)
            if file_path == "QUIT":
                print(f"Rank {rank} exiting.")
                break
            send_file(file_path)
```

3.2 The proxy

The proxy acts as an intermediary between the client and the server. It is a Flask-based HTTP server that receives file uploads from the Client via HTTP POST requests and forwards these files to the FTP server using the FTP protocol.

The Flask application is designed to run on 0.0.0.0, allowing it to accept connections from any network interface, and listens on port 6969.

When the client transfer a file, Flask receives it through the HTTP request. The file is then temporarily saved to a directory on the proxy. After that, a connection to the FTP server is established using ftplib module with designed credentials and transfers the file to the server. After the upload is complete, the FTP connection is terminated, and the previously temporary saved file is deleted to ensure storage cleanliness.

If any exceptions occur during the FTP upload process, an error message is returned to indicate the failure of the file upload.

Below is the proxy code:

```python
from flask import Flask, request
from ftplib import FTP
import os

app = Flask(__name__)

FTP_SERVER = "100.105.98.42"
FTP_USER = "ftpuser"
FTP_PASS = "ftppass"

@app.route('/upload', methods=['POST'])
def upload_file():
    file = request.files['file']
    file_path = f"/tmp/{file.filename}"
    file.save(file_path)

    try:
        ftp = FTP(FTP_SERVER)
        ftp.login(user=FTP_USER, passwd=FTP_PASS)
        with open(file_path, "rb") as f:
            ftp.storbinary(f"STOR {file.filename}", f)
        ftp.quit()
        os.remove(file_path)
        return "File uploaded to FTP Server successfully!"
    except Exception as e:
        return f"Failed to upload file: {e}"

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=6969)
```

3.3 The server

The server is an FTP server configured to store files received from the proxy. A setup script and configuration file are used to prepare the environment. The script creates an FTP user and sets up a directory for file storage.

Setup Script (`setup_ftp.sh`):

```bash
#!/bin/bash
sudo useradd -m ftpuser
echo "ftpuser:ftppass" | sudo chpasswd
sudo mkdir -p /home/ftpuser/files
sudo chown ftpuser:ftpuser /home/ftpuser/files
echo "Setup complete!"
```

The script sets up the system for FTP operations. First, it creates a user account and assign its password for authentication. The script then sets the directory for the local storage for transferred files. The user is granted read and write permission.

Configuration File (`vsftpd.conf`):

```
listen=YES
anonymous_enable=NO
local_enable=YES
write_enable=YES
chroot_local_user=YES
allow_writeable_chroot=YES
pasv_min_port=40000
pasv_max_port=50000
pasv_address=100.105.98.42
```

The file defines how the FTP server operates. It includes settings to manage connections and ensure secure file transfers. Key settings for the server are: disabling anonymous access, enable local users, granting write permission and restricting users to their home directory. The file also specifies the configuration for passive mode FTP connections: the range of ports (40000-50000) and the server's IP address (100.105.98.42).

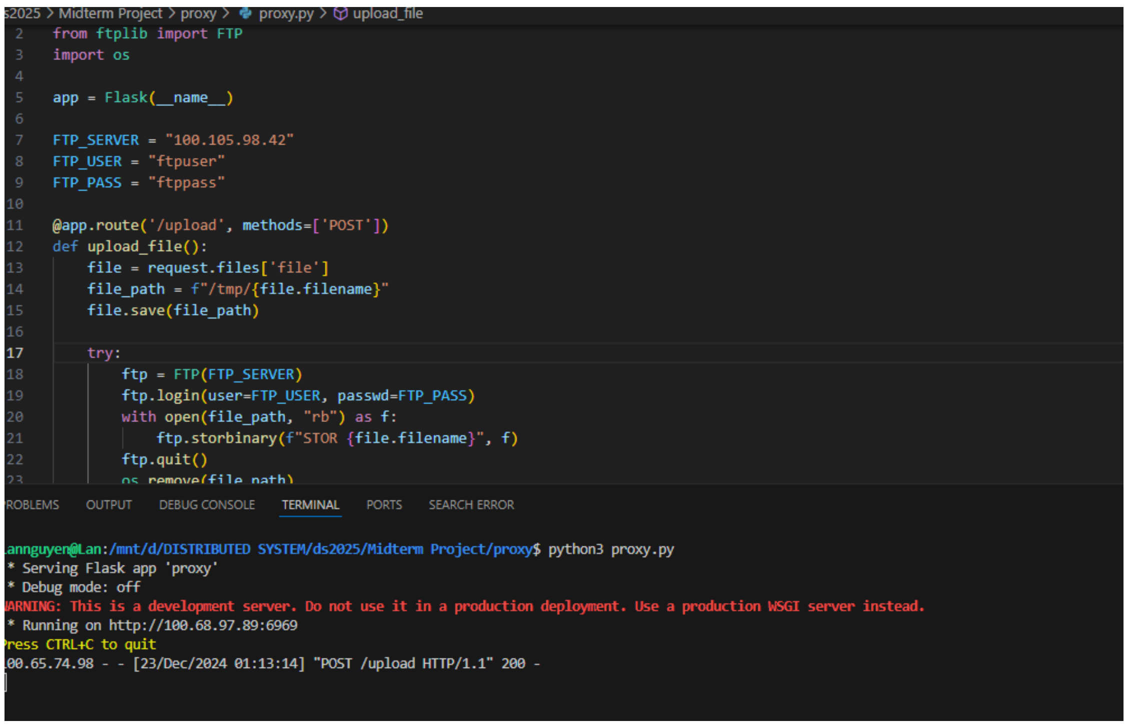
Once the server is set up, it listens for incoming connections on the IP address and requires authentication using the credentials. After that, the client are restricted to the configured directory, where it can upload and transfer files.

4 Result

First, run the client, proxy and server, let the components establish connection with each other to set up the system.

In the client interface, enter `"upload <file_path>"` to send file transfer request and start the transfer process.

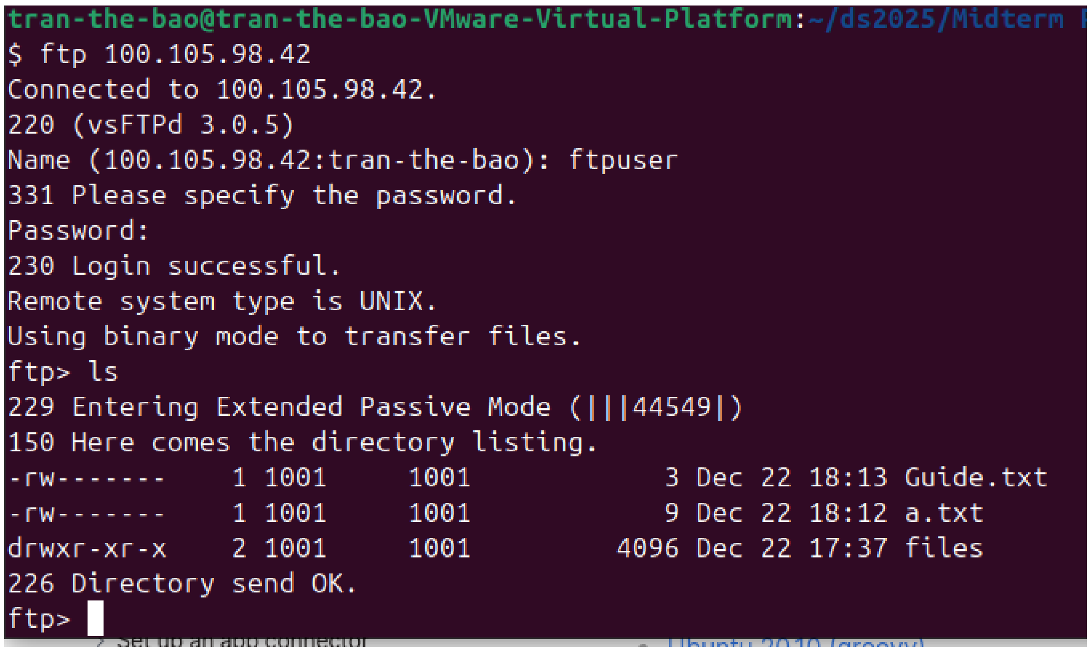
The proxy receives the request together with file data and forward them to the server.



```
s2025 > Midterm Project > proxy > proxy.py > upload_file
from ftplib import FTP
import os

app = Flask(__name__)

FTP_SERVER = "100.105.98.42"
FTP_USER = "ftpuser"
FTP_PASS = "ftppass"

@app.route('/upload', methods=['POST'])
def upload_file():
    file = request.files['file']
    file_path = f"/tmp/{file.filename}"
    file.save(file_path)

    try:
        ftp = FTP(FTP_SERVER)
        ftp.login(user=FTP_USER, passwd=FTP_PASS)
        with open(file_path, "rb") as f:
            ftp.storbinary(f"STOR {file.filename}", f)
        ftp.quit()
        os.remove(file_path)
```

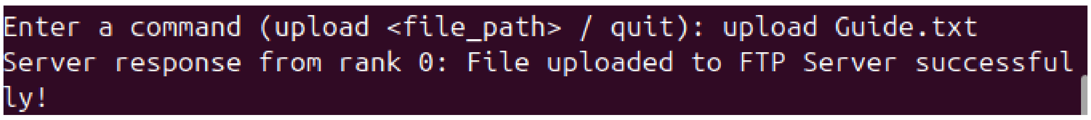
```
annguyen@Lan:/mnt/d/DISTRIBUTED SYSTEM/ds2025/Midterm Project/proxy$ python3 proxy.py
 * Serving Flask app 'proxy'
 * Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on http://100.68.97.89:6969
Press CTRL+C to quit
100.65.74.98 - - [23/Dec/2024 01:13:14] "POST /upload HTTP/1.1" 200 -
```

The server authenticates the client with credentials, receives file and send the response to the proxy, which then relays it back to the client.



```
tran-the-bao@tran-the-bao-VMware-Virtual-Platform:~/ds2025/Midterm
$ ftp 100.105.98.42
Connected to 100.105.98.42.
220 (vsFTPd 3.0.5)
Name (100.105.98.42:tran-the-bao): ftpuser
331 Please specify the password.
Password:
230 Login successful.
Remote system type is UNIX.
Using binary mode to transfer files.
ftp> ls
229 Entering Extended Passive Mode (|||44549|)
150 Here comes the directory listing.
-rw-------    1 1001     1001            3 Dec 22 18:13 Guide.txt
-rw-------    1 1001     1001            9 Dec 22 18:12 a.txt
drwxr-xr-x    2 1001     1001         4096 Dec 22 17:37 files
226 Directory send OK.
ftp> 
```

Lastly, the client receives the response from the proxy, completing the transfer cycle.



```
Enter a command (upload <file_path> / quit): upload Guide.txt
Server response from rank 0: File uploaded to FTP Server successful
ly!
```