Practical 1: TCP File Transfer
A Simple Client-Server Implementation

Nguyen Viet Khoa - 23BI14223

November 21, 2025

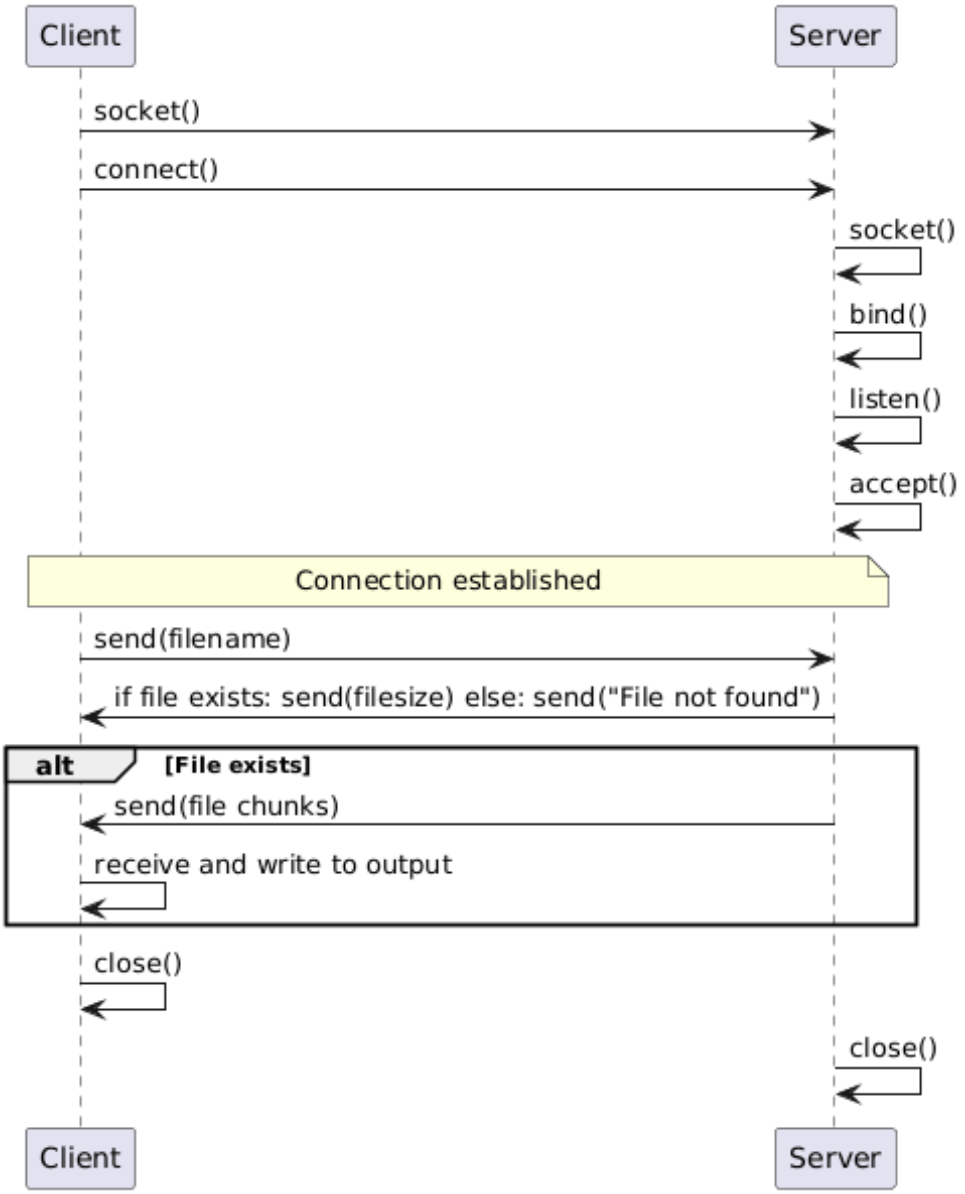
1 Introduction

This report presents a complete implementation of file transfer over TCP/IP using the command-line interface (CLI) and Python's built-in `socket` library. The system allows a client to request and download any file from the server reliably.

2 Protocol Design

The protocol is simple, reliable, and follows a request-response pattern over TCP:

- The client connects to the server at `127.0.0.1:65432`
- The client sends the requested filename (UTF-8 string)
- Server checks if file exists:
  - If not → sends `"File not found"`
  - If yes → sends file size (as string), then file content is in 4096-byte chunks
- The client receives and writes data to a local file
- The connection closes automatically after transfer (TCP EOF)



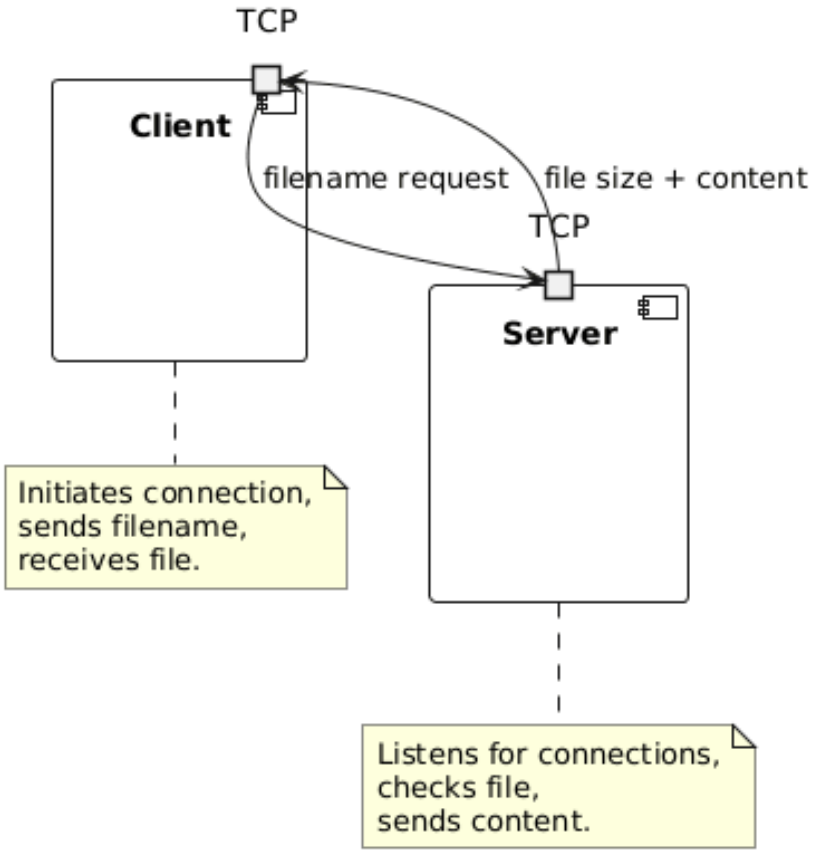
Hình 1: Sequence Diagram - TCP File Transfer Protocol

3 System Organization

The system follows a classic client-server architecture:

1. **Server**: Passive - listens on port 65432, accepts one connection, serves file
2. **Client**: Active - initiates connection, requests file by name, saves received file

Both programs run in the same folder for testing. All files (`server.py`, `client.py`, `example.txt`, `received.txt`) are placed together.



Hình 2: Component Diagram - Client-Server System Organization

4 File Transfer Implementation

The implementation uses only standard Python libraries. Run server first, then client in a separate terminal.

4.1 Server-side Implementation

```
import socket
import os

HOST = '127.0.0.1'
PORT = 65432

with socket.socket(socket.AF_INET, socket.SOCK_STREAM)
   as s:
    s.bind((HOST, PORT))
    s.listen()
    print(f"Server listening on {HOST}:{PORT}")
    conn, addr = s.accept()
    with conn:
        print(f"Connected by {addr}")
        filename =
           conn.recv(1024).decode('utf-8').strip()
        print(f"Client requested: {filename}")

        if not os.path.exists(filename):
            conn.sendall(b'File not found')
            print("File not found!")
        else:
            filesize = os.path.getsize(filename)
            conn.sendall(str(filesize).encode('utf-8'))
            print(f"Sending {filename} ({filesize}
               bytes)")

            with open(filename, 'rb') as f:
                while True:
                    data = f.read(4096)
                    if not data:
                        break
                    conn.sendall(data)
            print("File sent successfully!")
```

Listing 1: server.py - Corrected and Fully Working Version

4.2 Client-side Implementation

```
import socket
import sys

if len(sys.argv) != 3:
    print("Usage: python client.py <filename>
       <output_filename>")
    sys.exit(1)

filename = sys.argv[1]
output_filename = sys.argv[2]

HOST = '127.0.0.1'
PORT = 65432

with socket.socket(socket.AF_INET, socket.SOCK_STREAM)
   as s:
    s.connect((HOST, PORT))
    s.sendall(filename.encode('utf-8'))

    response = s.recv(1024).decode('utf-8')
    if response == 'File not found':
        print("Server: File not found!")
    else:
        filesize = int(response)
        print(f"Receiving {filename} ({filesize}
           bytes)")

        received = 0
        with open(output_filename, 'wb') as f:
            while received < filesize:
                data = s.recv(4096)
                if not data:
                    break
                f.write(data)
                received += len(data)
        print(f"File saved as {output_filename}")
```

Listing 2: client.py - Corrected and Fully Working Version

5 Role of Each Component

- **Server**:
  - Creates and binds the socket to port 65432
  - Listens and accepts client connection
  - Receives filename request
  - Sends file size + content in 4KB chunks
  - Handles "file not found" gracefully
- **Client**:
  - Connects to server
  - Sends the desired filename
  - Receives file size and data
  - Writes to a local file with progress tracking
  - Supports command-line arguments

6 How to Run

1. Place `example.txt` in the same folder
2. Terminal 1: `python server.py`
3. Terminal 2: `python client.py example.txt received.txt`
4. Check `received.txt` → identical to original!