

# 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 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:

- Client connects to server at `127.0.0.1:65432`
- 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 in 4096-byte chunks
- Client receives and writes data to local file
- Connection closes automatically after transfer (TCP EOF)

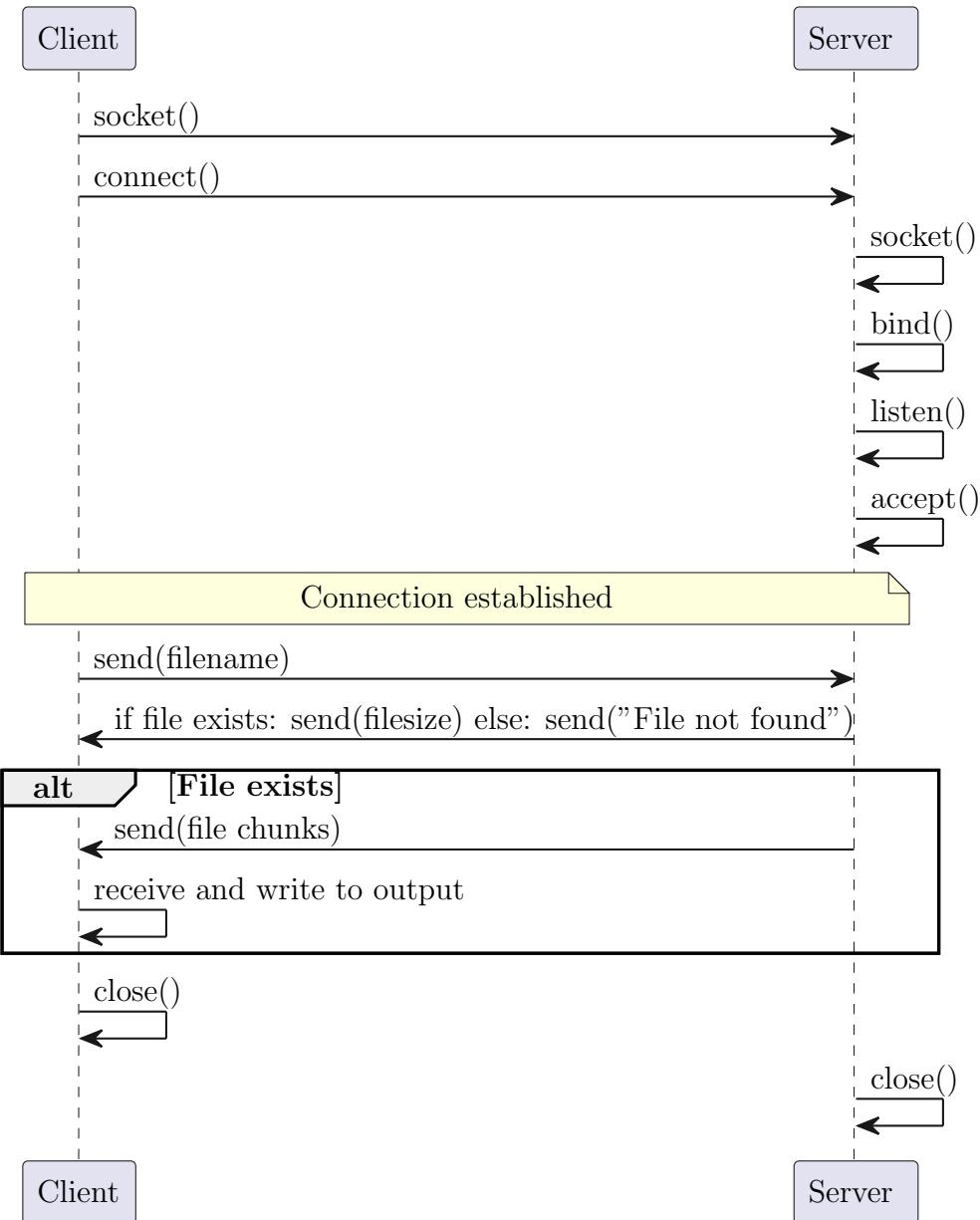


Figure 1: Sequence Diagram - TCP File Transfer Protocol

### 3 System Organization

The system follows 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.

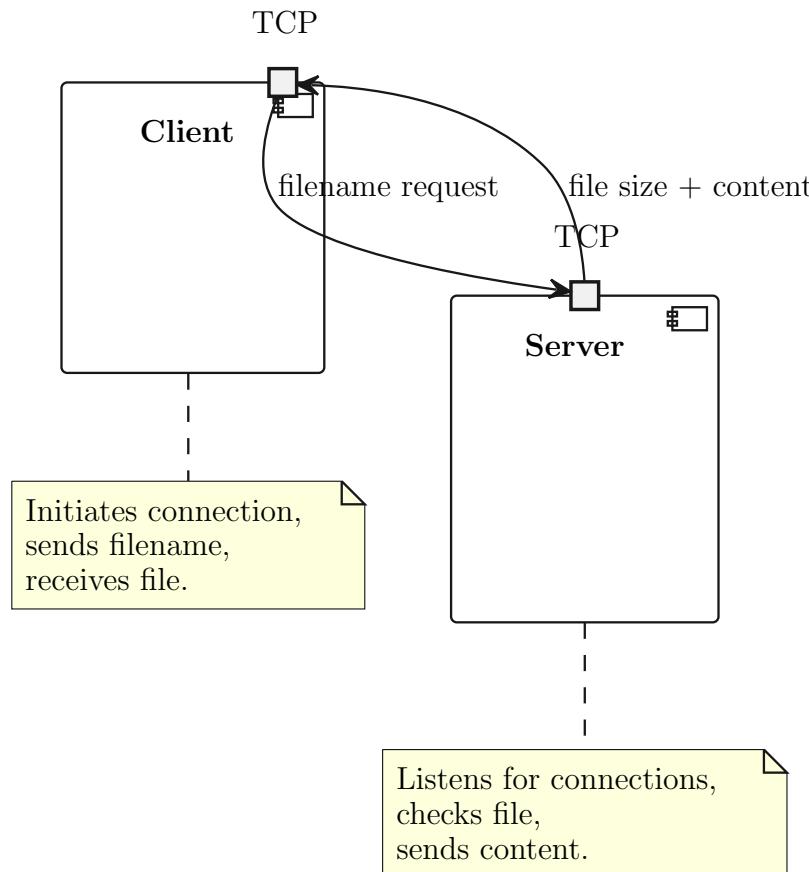


Figure 2: Component Diagram - Client-Server System Organization

## 4 File Transfer Implementation

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

### 4.1 Server-side Implementation

```
1 import socket
2 import os
3
4 HOST = '127.0.0.1'
5 PORT = 65432
6
7 with socket.socket(socket.AF_INET, socket.SOCK_STREAM)
8     as s:
9     s.bind((HOST, PORT))
10    s.listen()
11    print(f"Server listening on {HOST}:{PORT}")
12    conn, addr = s.accept()
13    with conn:
14        print(f"Connected by {addr}")
15        filename =
16            conn.recv(1024).decode('utf-8').strip()
17        print(f"Client requested: {filename}")
18
19        if not os.path.exists(filename):
20            conn.sendall(b'File not found')
21            print("File not found!")
22        else:
23            filesize = os.path.getsize(filename)
24            conn.sendall(str(filesize).encode('utf-8'))
25            print(f"Sending {filename} ({filesize}
26                  bytes)")
27
28            with open(filename, 'rb') as f:
29                while True:
30                    data = f.read(4096)
31                    if not data:
32                        break
33                    conn.sendall(data)
```

```
31     print("File sent successfully!")
```

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

## 4.2 Client-side Implementation

```
1 import socket
2 import sys
3
4 if len(sys.argv) != 3:
5     print("Usage: python client.py <filename>
6         <output_filename>")
7     sys.exit(1)
8
9 filename = sys.argv[1]
10 output_filename = sys.argv[2]
11
12 HOST = '127.0.0.1'
13 PORT = 65432
14
15 with socket.socket(socket.AF_INET, socket.SOCK_STREAM)
16     as s:
17         s.connect((HOST, PORT))
18         s.sendall(filename.encode('utf-8'))
19
20         response = s.recv(1024).decode('utf-8')
21         if response == 'File not found':
22             print("Server: File not found!")
23         else:
24             filesize = int(response)
25             print(f'Receiving {filename} ({filesize}
26                 bytes)')
27
28             received = 0
29             with open(output_filename, 'wb') as f:
30                 while received < filesize:
31                     data = s.recv(4096)
32                     if not data:
33                         break
34                     f.write(data)
```

```
31         f.write(data)
32         received += len(data)
33     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 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 desired filename
- Receives file size and data
- Writes to 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!