CSC203L Computer Networks Lab

Submitted by:

Umair Arshad 2024-SE-38

Submitted to:

Prof. Noman Munir

Dated: November 21, 2025

Department of Computer Science

University of Engineering and Technology, New Campus

UDP MINI-FTP CLIENT & SERVER DOCUMENTATION

1. Overview

The UDP Mini-FTP project implements a simple file transfer system over UDP, supporting reliable file uploads and downloads using a stop-and-wait protocol with ACKs. It consists of two components:

1. Client – an interactive command-line interface (CLI) for sending commands to the server.
2. Server – handles client requests, manages files in a storage directory, and ensures reliable transfers.

2. Features

2.1 Client Features

- Interactive REPL (Read-Eval-Print Loop) for commands: PING, LIST, UPLOAD <filepath>, DOWNLOAD <filename>, DELETE <filename>, EXIT.
- Stop-and-wait reliability for file transfer.
- Chunked file transfer (4 KB per chunk).
- Retransmission on lost packets.
- Handles timeouts and server non-responsiveness.

2.2 Server Features

- Receives and executes client commands over UDP.
- Stop-and-wait reliability for uploads/downloads.
- Dedicated storage directory (`server_files`).
- Logs operations to console: upload/download start and completion, file deletion, unknown commands.
- Ensures safe file paths to prevent directory traversal attacks.
- Handles client-specific packets to avoid interference.

3. System Requirements

- Python 3.8+
- Standard Python libraries (`socket`, `os`, `time`)
- Network access (localhost for testing or LAN)

4. File Structure

```
project/
├── client.py        # UDP Mini-FTP Client
├── server.py        # UDP Mini-FTP Server
├── server_files/    # Server storage directory
└── README.md
```

5. Usage

5.1 Running the Server

python3 server.py

- Server listens on localhost:8080.
- Server will process commands from clients sequentially.

5.2 Running the Client

python3 client.py

Interactive REPL starts:

> PING

Server: PONG

> LIST

Files on server:

(file1.txt)

Upload/Download files:

> UPLOAD localfile.txt

> DOWNLOAD file_on_server.txt

6. Protocol & Data Flow

6.1 Stop-and-Wait for Reliability

- Client → Server: Sends a chunk with header DATA:<seq>\n<chunk>.
- Server → Client: Responds with ACK:<seq> upon receipt.
- Retransmit occurs if ACK not received within ACK_TIMEOUT.
- File transfer ends with END packet, followed by a confirmation message.

6.2 Command Handling

Command	Client Request	Server Response
PING	PING	PONG
LIST	LIST	List of files or (no files)
UPLOAD	UPLOAD <filename>	UPLOAD_OK → file transfer
DOWNLOAD	DOWNLOAD <filename>	SIZE:<bytes> → READY → transfer
DELETE	DELETE <filename>	DELETE_OK or ERROR:NOTFOUND
EXIT	EXIT	BYE (server shutdown)

7. Design Notes

7.1 Client

- Uses UDP socket with optional response timeout.
- Handles file chunking and retransmission.
- Interactive REPL parses and executes commands.

7.2 Server

- Single-threaded listening on UDP socket.
- Each transfer ensures client isolation by checking sender address.
- Stores files safely in server_files.
- Handles packet retransmission using sendto_retry() for downloads.

8. Limitations

- Single-threaded server: cannot handle multiple clients uploading/downloading simultaneously.
- EXIT command shuts down the server completely.
- No checksum verification for file integrity.
- Fixed chunk size (4000 bytes).
- Minimal logging and no persistent state between runs.

9. Potential Enhancements

1. Multi-client support using threading or asyncio.
2. File integrity verification (MD5/SHA256).
3. Progress feedback for uploads/downloads.
4. Improved command handling for EXIT per client.
5. Logging transfer activity to file.
6. Dynamic chunk size based on network conditions.

10. Security Considerations

- File paths sanitized using os.path.basename() to prevent directory traversal.
- Only accepts packets from requesting client during transfer.
- Ensure server directory permissions are restricted to avoid unauthorized access.

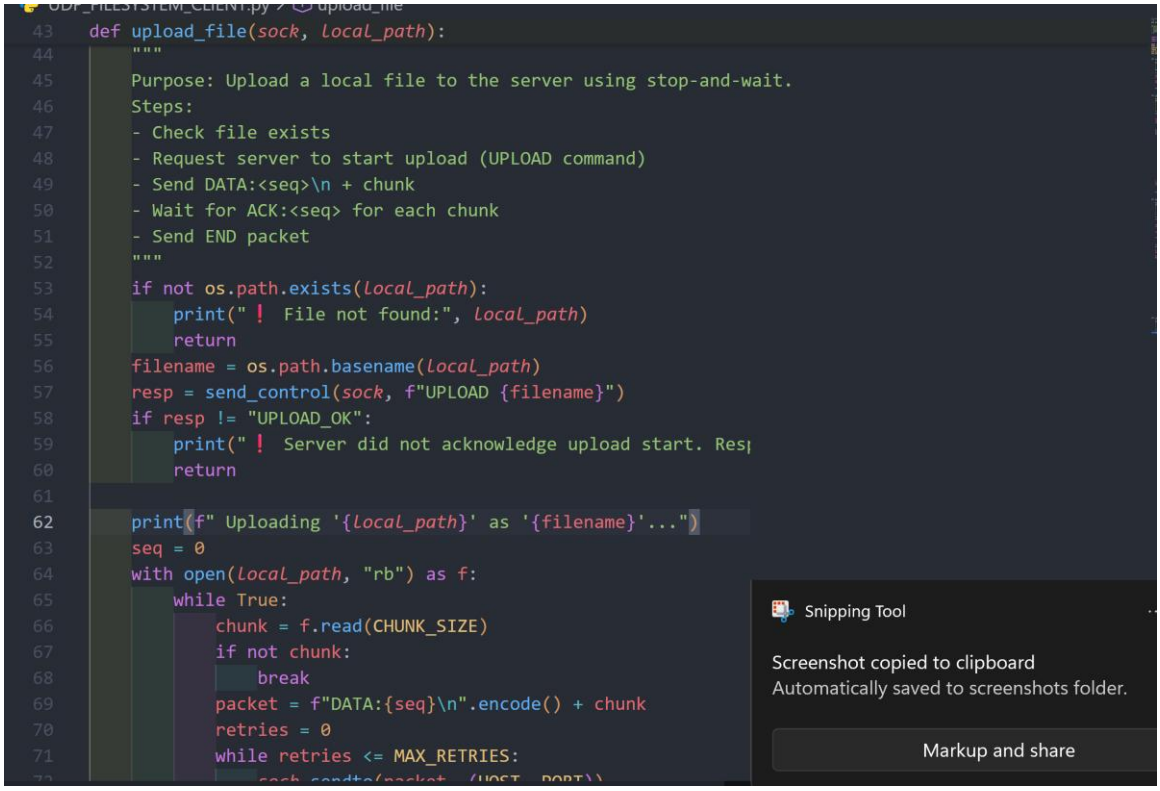
11. Conclusion

The UDP Mini-FTP system provides a lightweight, reliable file transfer solution over UDP using stop-and-wait. It is well-suited for learning network programming, understanding reliability protocols, and building a small FTP-like service.

Next Steps:

- Add multi-client concurrency.
- Implement checksums for file integrity.
- Enhance user feedback in both client and server.

Screen Shots



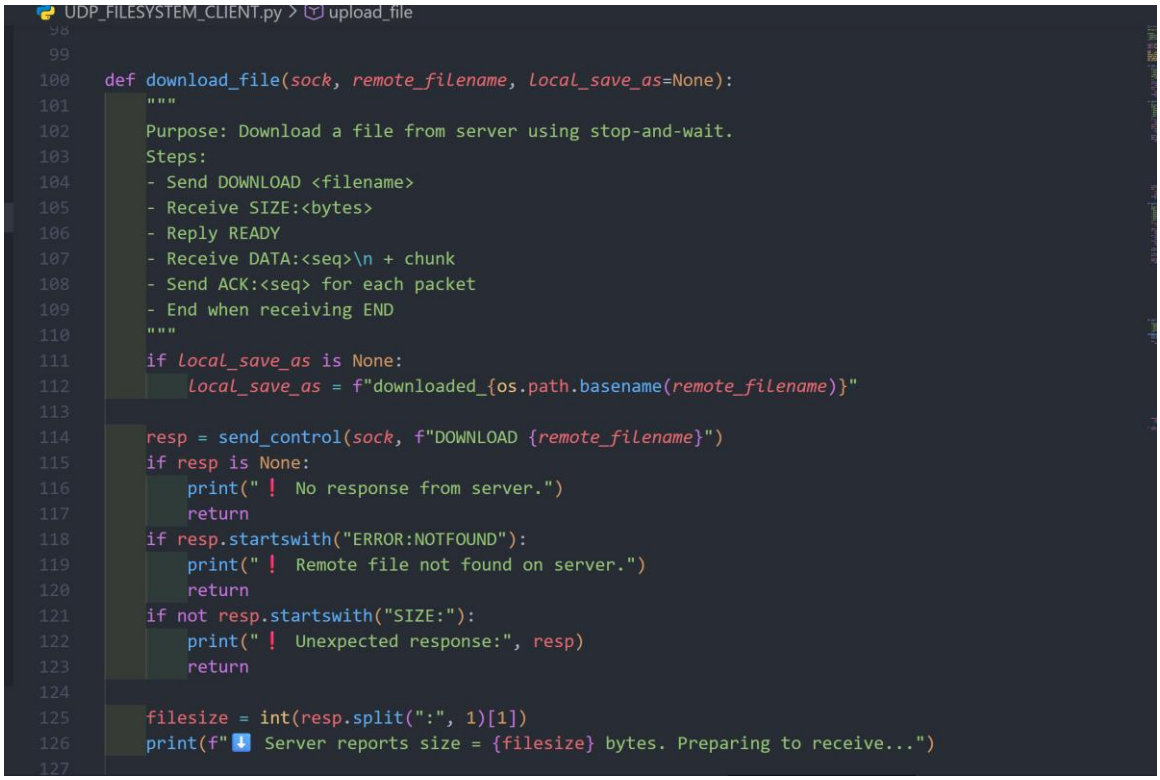
```python
def upload_file(sock, local_path):
    """
    Purpose: Upload a local file to the server using stop-and-wait.
    Steps:
    - Check file exists
    - Request server to start upload (UPLOAD command)
    - Send DATA:<seq>\n + chunk
    - Wait for ACK:<seq> for each chunk
    - Send END packet
    """
    if not os.path.exists(local_path):
        print(" ! File not found:", local_path)
        return
    filename = os.path.basename(local_path)
    resp = send_control(sock, f"UPLOAD {filename}")
    if resp != "UPLOAD_OK":
        print(" ! Server did not acknowledge upload start. Res
        return

    print(f" Uploading '{local_path}' as '{filename}'...")
    seq = 0
    with open(local_path, "rb") as f:
        while True:
            chunk = f.read(CHUNK_SIZE)
            if not chunk:
                break
            packet = f"DATA:{seq}\n".encode() + chunk
            retries = 0
            while retries <= MAX_RETRIES:
```

Snipping Tool

Screenshot copied to clipboard
Automatically saved to screenshots folder.

Markup and share

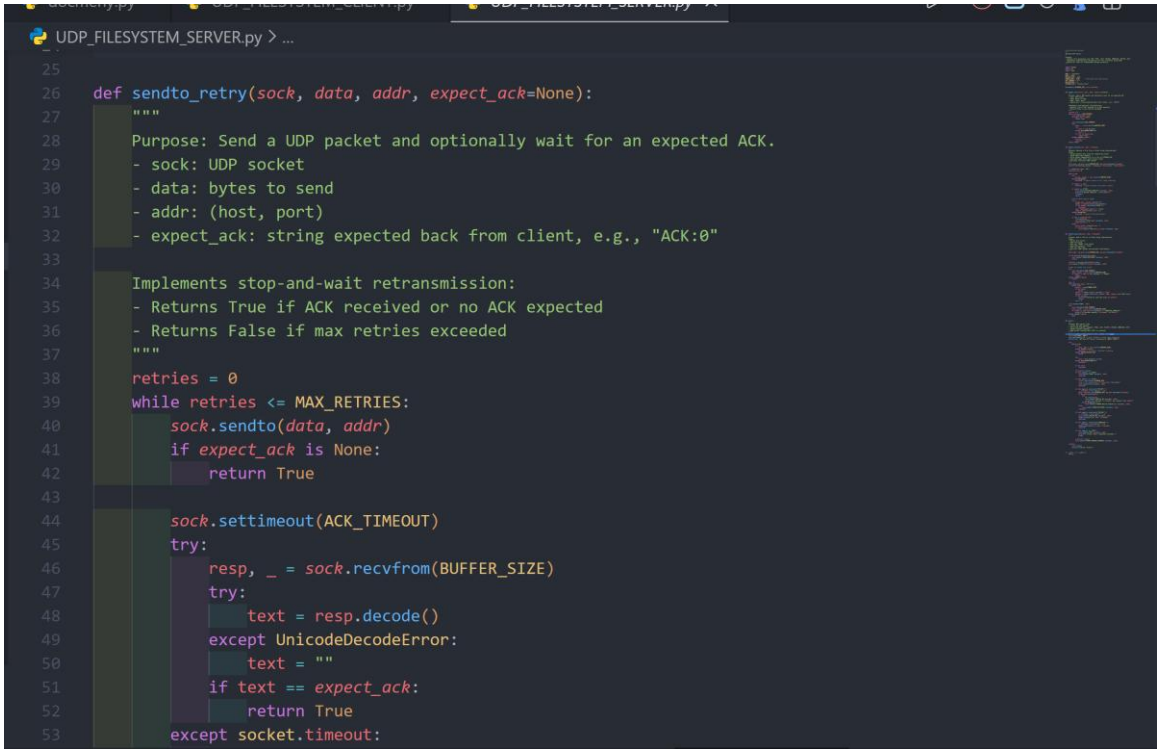


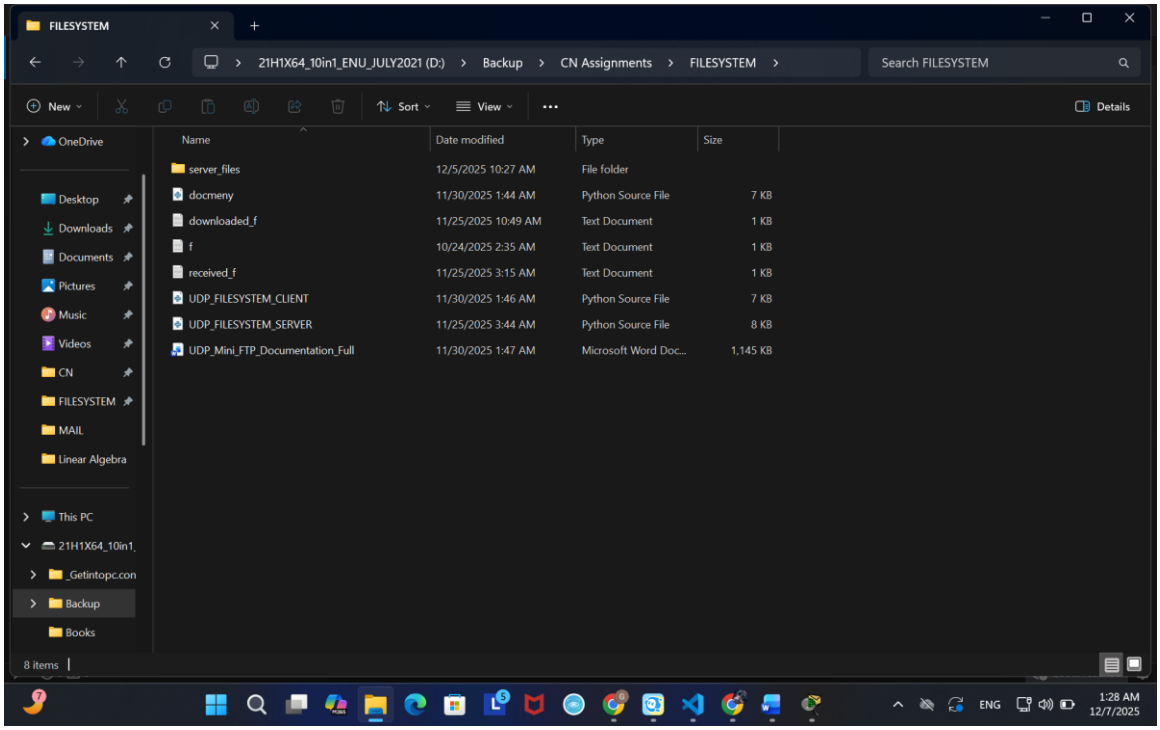
UDP_FILESYSTEM_CLIENT.py > upload_file

```python
def download_file(sock, remote_filename, local_save_as=None):
    """
    Purpose: Download a file from server using stop-and-wait.
    Steps:
    - Send DOWNLOAD <filename>
    - Receive SIZE:<bytes>
    - Reply READY
    - Receive DATA:<seq>\n + chunk
    - Send ACK:<seq> for each packet
    - End when receiving END
    """
    if local_save_as is None:
        local_save_as = f"downloaded_{os.path.basename(remote_filename)}"

    resp = send_control(sock, f"DOWNLOAD {remote_filename}")
    if resp is None:
        print(" ! No response from server.")
        return
    if resp.startswith("ERROR:NOTFOUND"):
        print(" ! Remote file not found on server.")
        return
    if not resp.startswith("SIZE:"):
        print(" ! Unexpected response:", resp)
        return

    filesize = int(resp.split(":", 1)[1])
    print(f"⬇ Server reports size = {filesize} bytes. Preparing to receive...")
```





The files available on server are in the server file folder while downloaded files from server by client are shown by prefix received_ filename .The f file is not with the prefix because it was a test file which client uploaded to the server and later downloaded it again.