

FTP-Lite: QUIC-based File Transfer Protocol

Course: CS544 - Computer Networks Project

Professor: Brian Mitchell

Student Name: Sarthak Vinod Shrungare

University ID:14744552

Overview

FTP-Lite is a custom stateful file-transfer protocol implemented using QUIC, designed to deliver secure, reliable, and efficient file uploads from clients to a central server. It utilizes modern transport technology (QUIC and TLS) along with application layer design principles. It uses deterministic finite automata (DFA), explicitly structured PDUs, and a stateful session. This project has both an aspect of practical protocol design, along with an aspect of robustness (e.g., asynchronously managing multiple clients, finding server dynamically through UDP broadcast, automated protocol testing through pytest. It serves as a realistic prototype of how lightweight file transfer systems can be designed and implemented with modern, secure, and extensible architecture.

Project Structure

```
FTP-Lite/
├── server.py
├── client.py
├── protocol.py
├── requirements.txt
├── cert.pem
├── key.pem
├── run_server.sh
├── run.sh
├── Makefile
├── tests/
│   └── test_ftplite.py
├── Result/
├── myfile.txt
├── fileA.txt
└── fileB.txt
```

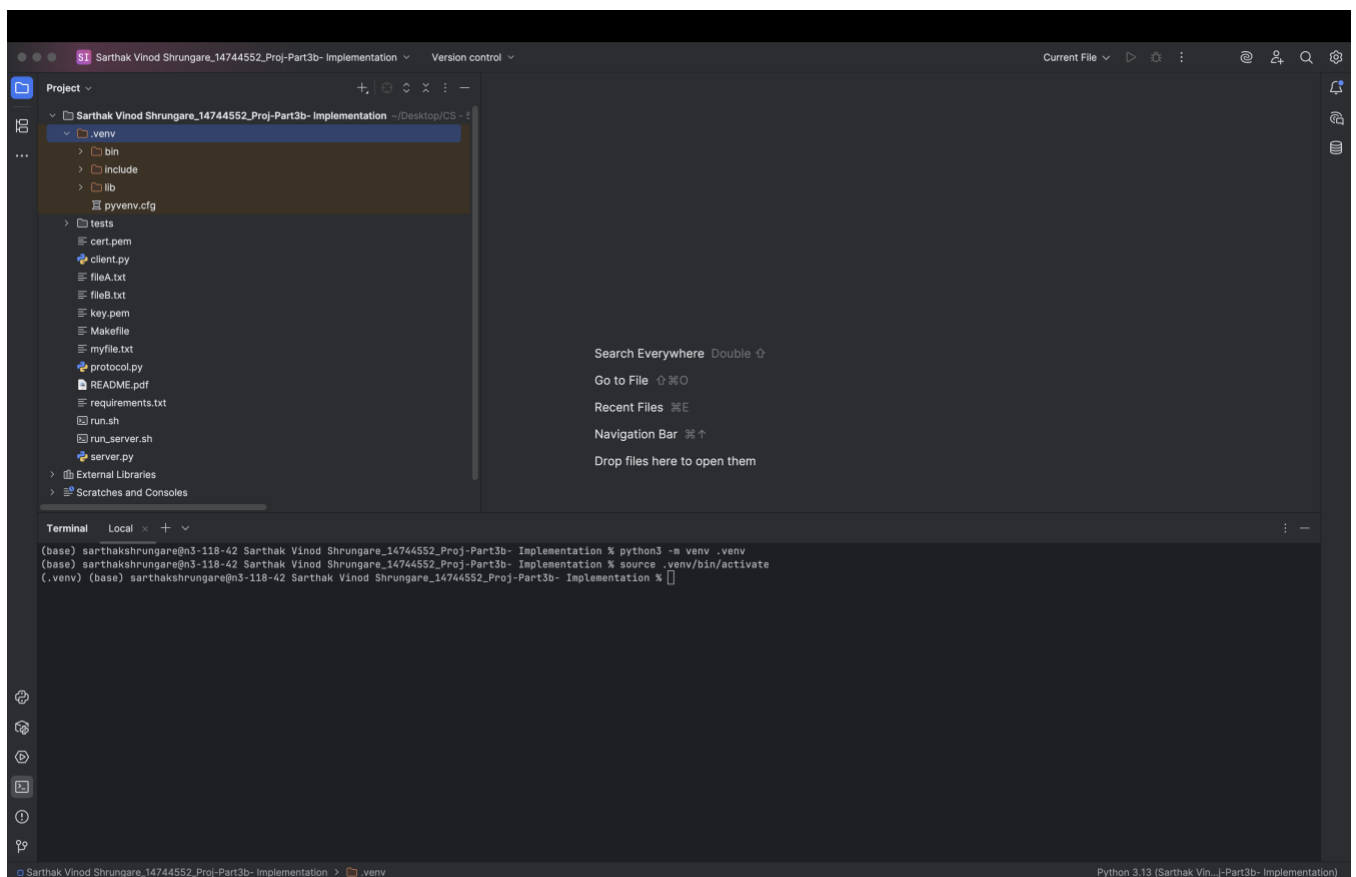
Installation & Setup

1. Prerequisites:

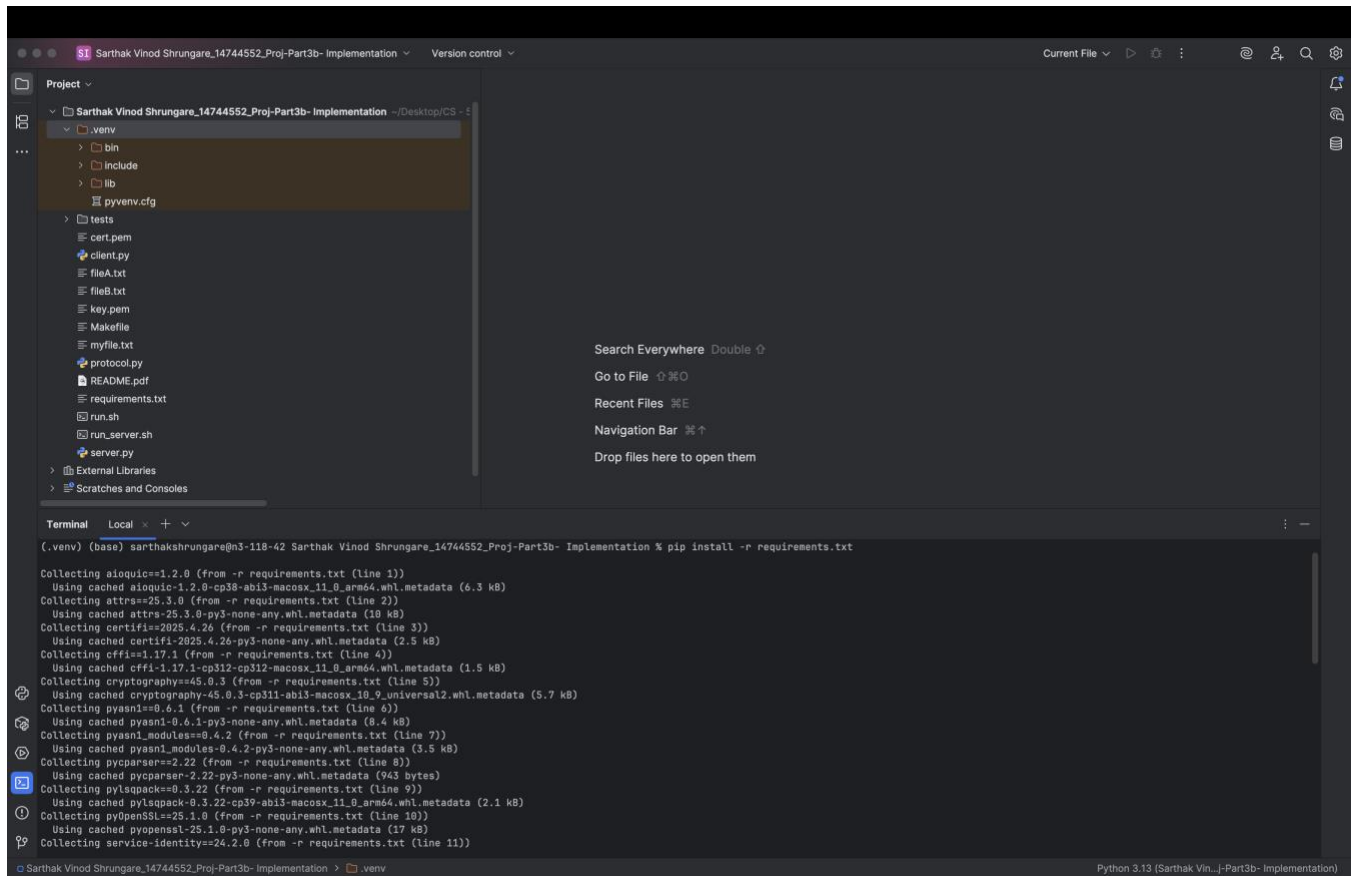
- Python 3.13+
- pip package manager
- OpenSSL for generating TLS certificates

2. Setup Steps:

- `python3 -m venv .venv`
- `source .venv/bin/activate`



- `pip install -r requirements.txt`



The screenshot shows a Visual Studio Code editor interface. The left sidebar displays the 'Project' view with a file tree for a project named 'Sarathak Vinod Shringare_14744552_Proj-Part3b- Implementation'. The tree includes a '.venv' directory with subfolders 'bin', 'include', and 'lib', and a 'pyvenv.cfg' file. Below these are 'tests' and various files like 'cert.pem', 'client.py', 'fileA.txt', 'fileB.txt', 'key.pem', 'Makefile', 'myfile.txt', 'protocol.py', 'README.pdf', 'requirements.txt', 'run.sh', 'run_server.sh', and 'server.py'. The 'External Libraries' and 'Scratches and Consoles' sections are also visible.

The main editor area shows a search bar with the text 'Search Everywhere Double' and a 'Go to File' button. Below this, there are 'Recent Files' and a 'Navigation Bar'.

The bottom panel shows a terminal window with the command `pip install -r requirements.txt` being executed. The output shows the installation progress for various packages, including `aiosqlite`, `attrs`, `certifi`, `cffi`, `cryptography`, `pyasn1`, `pyasn1-modules`, `pycparser`, `pylspack`, `pyOpenSSL`, and `pyOpenSSL`.

3. TLS certificates: cert.pem, key.pem are pre-generated and included in the project.

Platform Compatibility

This project was developed and tested on macOS. All scripts (*.sh), Makefile targets, and Python commands are fully compatible with Linux-based systems.

Protocol Description

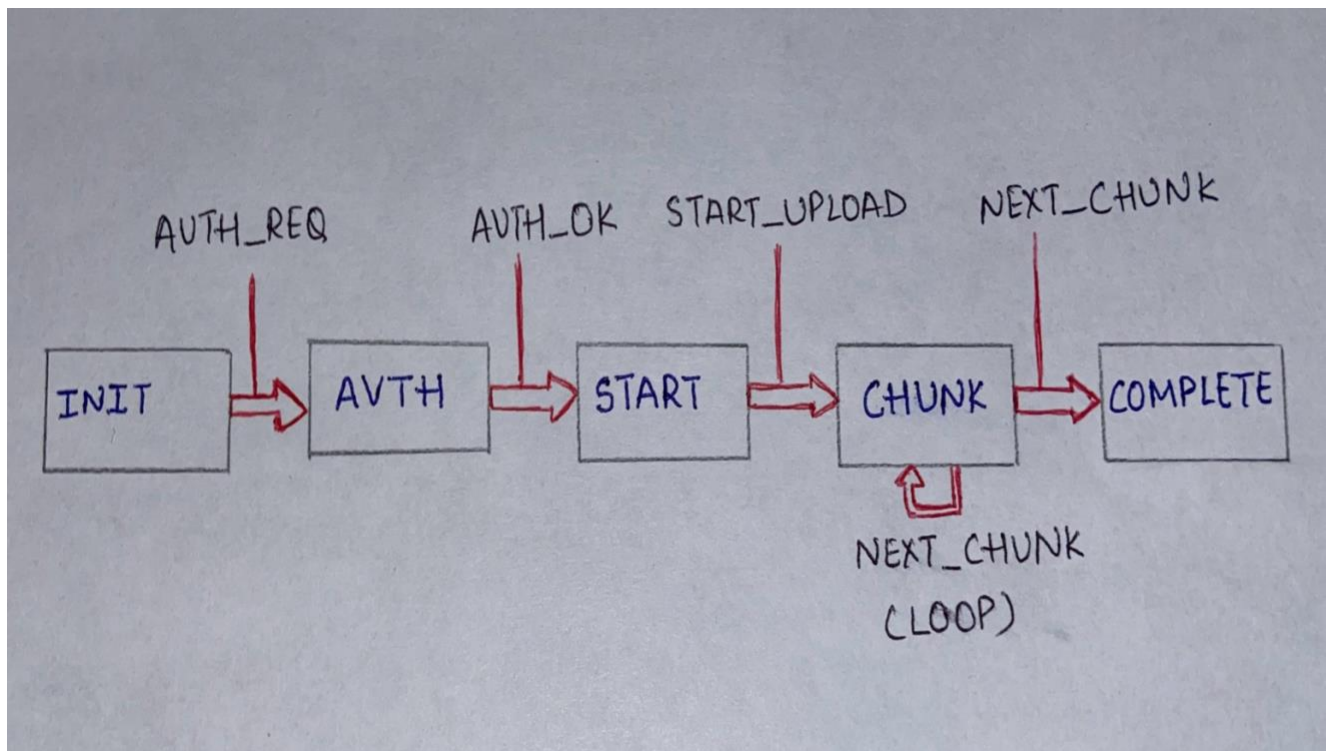
1. Service Description:

- FTP-Lite is a reliable client-server protocol that allows users to upload files to a server. It uses QUIC for transport and supports authentication, concurrency, and security using TLS.

2. DFA Protocol States:

- INIT → AUTH → START → SEGMENT → COMPLETE
- Each state governs how messages are sent and ensures that transitions occur in a validated and secure manner.

3. DFA Diagram:



4. Message Definitions (PDU's):

- Messages are defined with structured binary layouts using big-endian formatting. Each PDU includes a header with version, type, and length fields, followed by message-specific data. Examples include:
- **INIT**: Starts the protocol session and declares version compatibility.
- **AUTH**: Sends username and password for client authentication.
- **START**: Notifies the server about the incoming file name and size.
- **SEGMENT**: Transfers the file content in binary chunks.
- **COMPLETE**: Marks the end of file transmission and triggers upload finalization.

PDU Type	Field	Size (bytes)	Description
Common Header	Version	1	Protocol version (0x01)
Common Header	Type	1	Message type (INIT, AUTH, etc.)
Common Header	Length	2	Length of payload in bytes (big-endian)
INIT	(No payload)	0	Client initiates the protocol session
AUTH	Username	Variable	UTF-8 encoded username
AUT	Password	Variable	UTF-8 encoded password
STAR	Filename	Variable	Name of the file to upload
START	File Size	8	Size of the file (big-endian unsigned int)
SEGMENT	Segment Number	4	Segment index for chunk (big-endian)
SEGMEN	Chunk Data	Variable	Binary data of the file chunk
ACK	Segment Number	4	Server acknowledges received segment
COMPLETE	(No payload)	0	Signifies end of file transmission
ERROR	Error Message	Variable	UTF-8 error message describing failure

5. Extensibility:

- Protocol versioning supported with a version field (currently 0x01).
- New message types or optional fields can be added while maintaining backward compatibility.

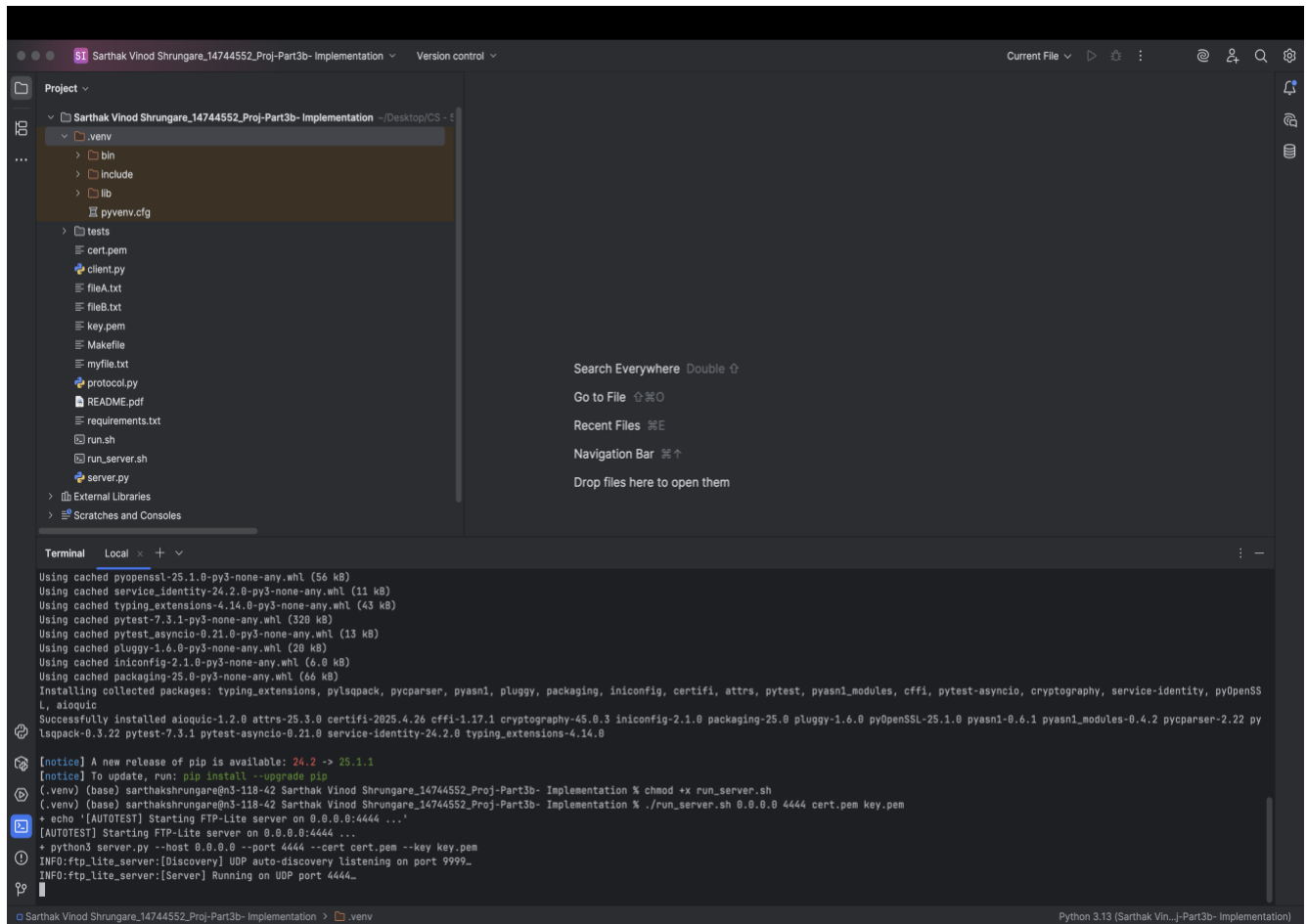
6. Authentication:

- Uses password-based login (e.g., user: bob, password: admin).
- Unauthenticated users are denied access with AUTH-ERR.
- In production, authentication should use hashed credentials and secure storage

HOW TO RUN:

1. Starting the Server:

- `chmod +x run_server.sh`
- `./run_server.sh 0.0.0.0 4444 cert.pem key.pem`



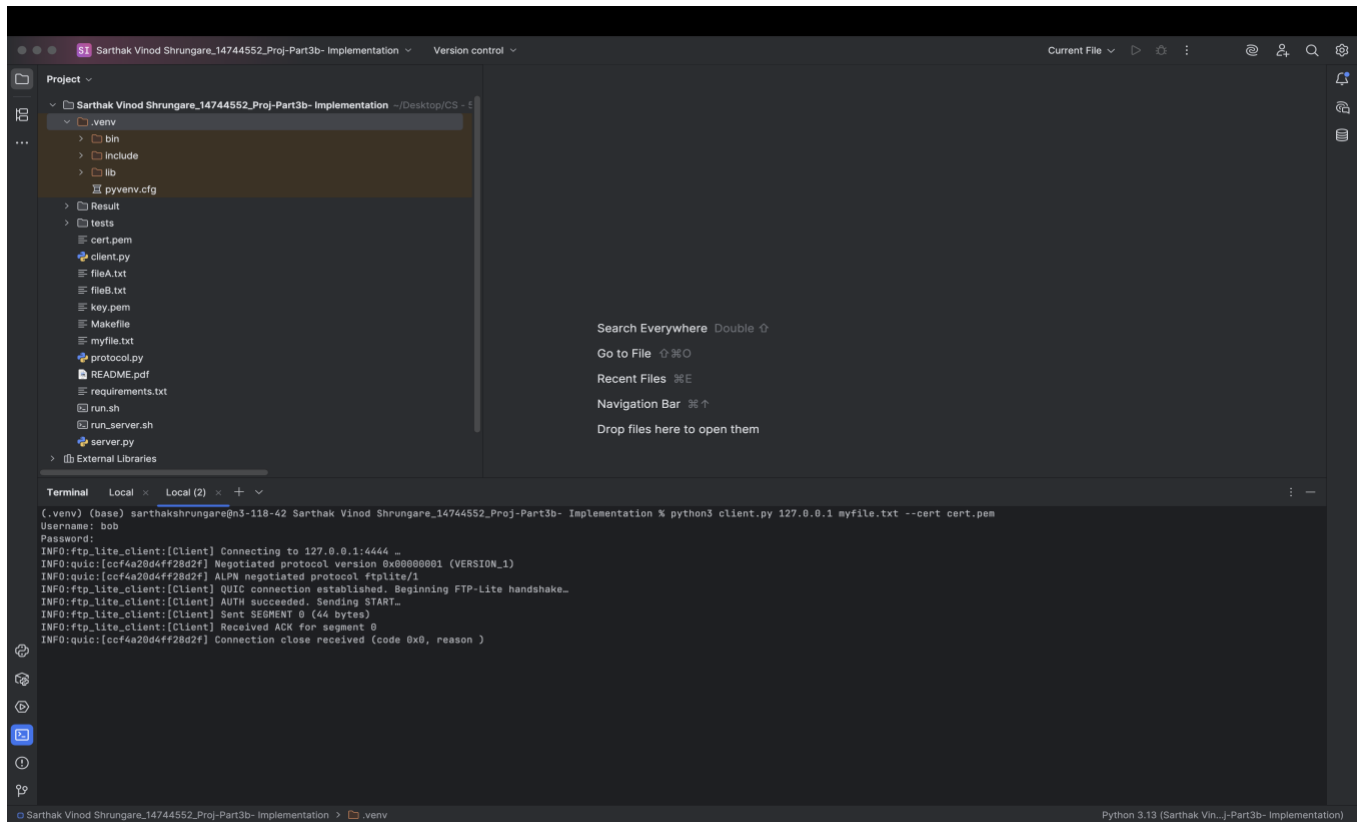
The screenshot shows a VS Code editor interface. The left sidebar displays the project structure for 'Sarthak Vinod Shrungare_14744552_Proj-Part3b- Implementation'. The file explorer shows a directory named '.venv' containing subdirectories 'bin', 'include', and 'lib', along with a 'pyvenv.cfg' file. Below this, a 'tests' directory contains files like 'cert.pem', 'client.py', 'fileA.txt', 'fileB.txt', 'key.pem', 'Makefile', 'myfile.txt', 'protocol.py', 'README.pdf', 'requirements.txt', 'run.sh', 'run_server.sh', and 'server.py'. The right sidebar shows a search bar and a list of recent files. The bottom panel is a terminal window showing the output of running 'pip install --upgrade pip' and then 'python3 server.py --host 0.0.0.0 --port 4444 --cert cert.pem --key key.pem'. The terminal output indicates that the server is running on UDP port 4444.

```
Using cached pyopenssl-25.1.0-py3-none-any.whl (56 kB)
Using cached service_identity-24.2.0-py3-none-any.whl (11 kB)
Using cached typing_extensions-4.14.0-py3-none-any.whl (43 kB)
Using cached pytest-7.3.1-py3-none-any.whl (320 kB)
Using cached pytest_asyncio-0.21.0-py3-none-any.whl (13 kB)
Using cached pluggy-1.6.0-py3-none-any.whl (20 kB)
Using cached iniconfig-2.1.0-py3-none-any.whl (6.0 kB)
Using cached packaging-25.0-py3-none-any.whl (66 kB)
Installing collected packages: typing_extensions, pysocks, pycparser, pyasn1, pluggy, packaging, iniconfig, certifi, attrs, pytest, pyasn1_modules, cffi, pytest-asyncio, cryptography, service_identity, pyOpenSSL, aioquic
Successfully installed aioquic-1.2.0 attrs-25.3.0 certifi-2025.4.26 cffi-1.17.1 cryptography-45.0.3 iniconfig-2.1.0 packaging-25.0 pluggy-1.6.0 pyOpenSSL-25.1.0 pyasn1-0.6.1 pyasn1_modules-0.4.2 pycparser-2.22 pysocks-0.3.22 pytest-7.3.1 pytest-asyncio-0.21.0 service_identity-24.2.0 typing_extensions-4.14.0

[notice] A new release of pip is available: 24.2 -> 25.1.1
[notice] To update, run: pip install --upgrade pip
(.venv) (base) sarthakshrungare@n3-118-42 Sarthak Vinod Shrungare_14744552_Proj-Part3b- Implementation % chmod +x run_server.sh
(.venv) (base) sarthakshrungare@n3-118-42 Sarthak Vinod Shrungare_14744552_Proj-Part3b- Implementation % ./run_server.sh 0.0.0.0 4444 cert.pem key.pem
+ echo '[AUTOTEST] Starting FTP-Lite server on 0.0.0.0:4444 ...'
[AUTOTEST] Starting FTP-Lite server on 0.0.0.0:4444 ...
+ python3 server.py --host 0.0.0.0 --port 4444 --cert cert.pem --key key.pem
INFO:ftp_lite_server:[Discovery] UDP auto-discovery listening on port 9999_
INFO:ftp_lite_server:[Server] Running on UDP port 4444_
```

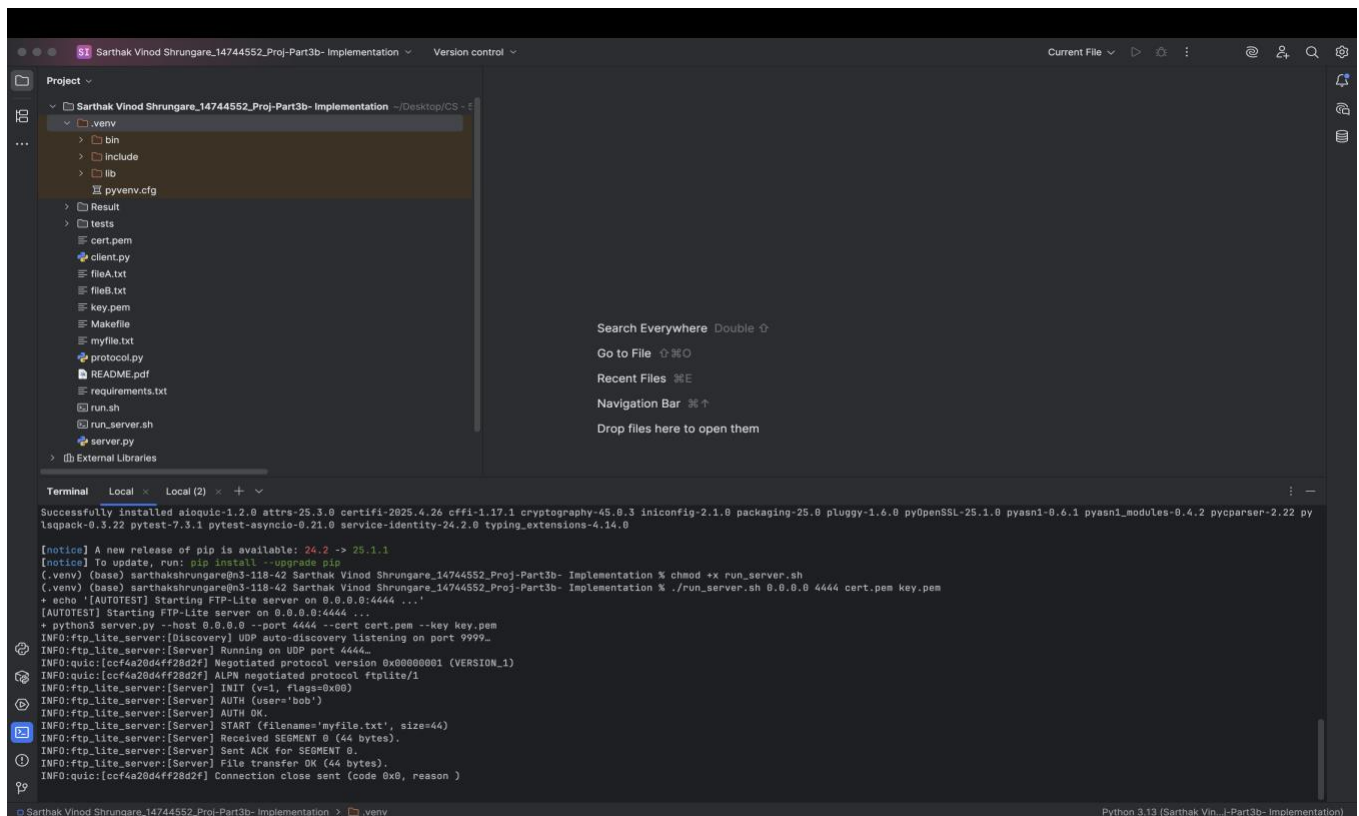
2. Interactive Mode:

- python3 client.py 127.0.0.1 myfile.txt --cert cert.pem



The screenshot shows the VS Code interface with a project named 'Sarthak Vinod Shrugare_14744552_Proj-Part3b- Implementation'. The file explorer on the left shows a directory structure with files like 'cert.pem', 'client.py', 'fileA.txt', 'fileB.txt', 'key.pem', 'Makefile', 'myfile.txt', 'protocol.py', 'README.pdf', 'requirements.txt', 'run.sh', 'run_server.sh', and 'server.py'. The terminal at the bottom shows the output of running 'python3 client.py 127.0.0.1 myfile.txt --cert cert.pem'. The output indicates a successful connection to 127.0.0.1:4444, negotiation of the ftplite/1 protocol, and a successful authentication process.

```
(.venv) (base) sarthakshrugare@n3-118-42 Sarthak Vinod Shrugare_14744552_Proj-Part3b- Implementation % python3 client.py 127.0.0.1 myfile.txt --cert cert.pem
Username: bob
Password:
INFO:ftp_lite_client:[Client] Connecting to 127.0.0.1:4444 _
INFO:quic:[ccf4a20d4ff28d2f] Negotiated protocol version 0x00000001 (VERSION_1)
INFO:quic:[ccf4a20d4ff28d2f] ALPN negotiated protocol ftplite/1
INFO:ftp_lite_client:[Client] QUIC connection established. Beginning FTP-Lite handshake..
INFO:ftp_lite_client:[Client] AUTH succeeded. Sending START..
INFO:ftp_lite_client:[Client] Sent SEGMENT 0 (44 bytes)
INFO:ftp_lite_client:[Client] Received ACK for segment 0
INFO:quic:[ccf4a20d4ff28d2f] Connection close received (code 0x0, reason )
```



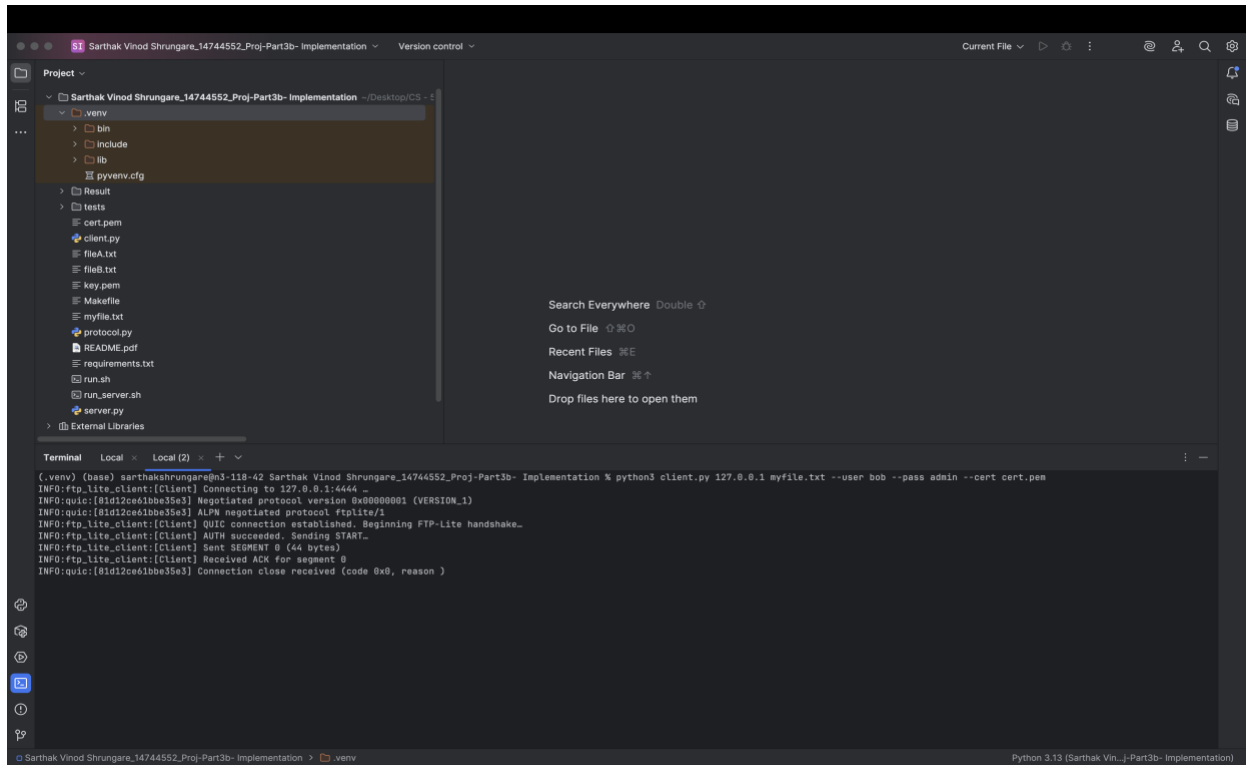
The screenshot shows the VS Code interface with the same project. The terminal at the bottom shows the output of running 'python3 server.py --host 0.0.0.0 --port 4444 --cert cert.pem --key key.pem'. The output indicates that the server is running on UDP port 4444, negotiated the ftplite/1 protocol, and successfully authenticated the client 'bob' who transferred the file 'myfile.txt'.

```
Successfully installed aioquic-1.2.0 attr-25.3.0 certifi-2025.4.26 cffi-1.17.1 cryptography-45.0.3 iniconfig-2.1.0 packaging-25.0 pluggy-1.6.0 pyOpenSSL-25.1.0 pyasn1-0.6.1 pyasn1_modules-0.4.2 pycparser-2.22 py
lsapack-0.3.22 pytest-7.3.1 pytest-asyncio-0.21.0 service-identity-24.2.0 typing_extensions-4.14.0

[notice] A new release of pip is available: 24.2 -> 25.1.1
[notice] To update, run: pip install --upgrade pip
(.venv) (base) sarthakshrugare@n3-118-42 Sarthak Vinod Shrugare_14744552_Proj-Part3b- Implementation % chmod +x run_server.sh
(.venv) (base) sarthakshrugare@n3-118-42 Sarthak Vinod Shrugare_14744552_Proj-Part3b- Implementation % ./run_server.sh 0.0.0.0 4444 cert.pem key.pem
+ echo '[AUTOTEST] Starting FTP-Lite server on 0.0.0.0:4444 ...'
[AUTOTEST] Starting FTP-Lite server on 0.0.0.0:4444 ...
+ python3 server.py --host 0.0.0.0 --port 4444 --cert cert.pem --key key.pem
INFO:ftp_lite_server:[Discovery] UDP auto-discovery Listening on port 9999..
INFO:ftp_lite_server:[Server] Running on UDP port 4444..
INFO:quic:[ccf4a20d4ff28d2f] Negotiated protocol version 0x00000001 (VERSION_1)
INFO:ftp_lite_server:[Server] ALPN negotiated protocol ftplite/1
INFO:ftp_lite_server:[Server] INIT (v=1, flags=0x00)
INFO:ftp_lite_server:[Server] AUTH (user='bob')
INFO:ftp_lite_server:[Server] AUTH OK.
INFO:ftp_lite_server:[Server] START (filename='myfile.txt', size=44)
INFO:ftp_lite_server:[Server] Received SEGMENT 0 (44 bytes).
INFO:ftp_lite_server:[Server] Sent ACK for SEGMENT 0.
INFO:ftp_lite_server:[Server] File transfer OK (44 bytes).
INFO:quic:[ccf4a20d4ff28d2f] Connection close sent (code 0x0, reason )
```

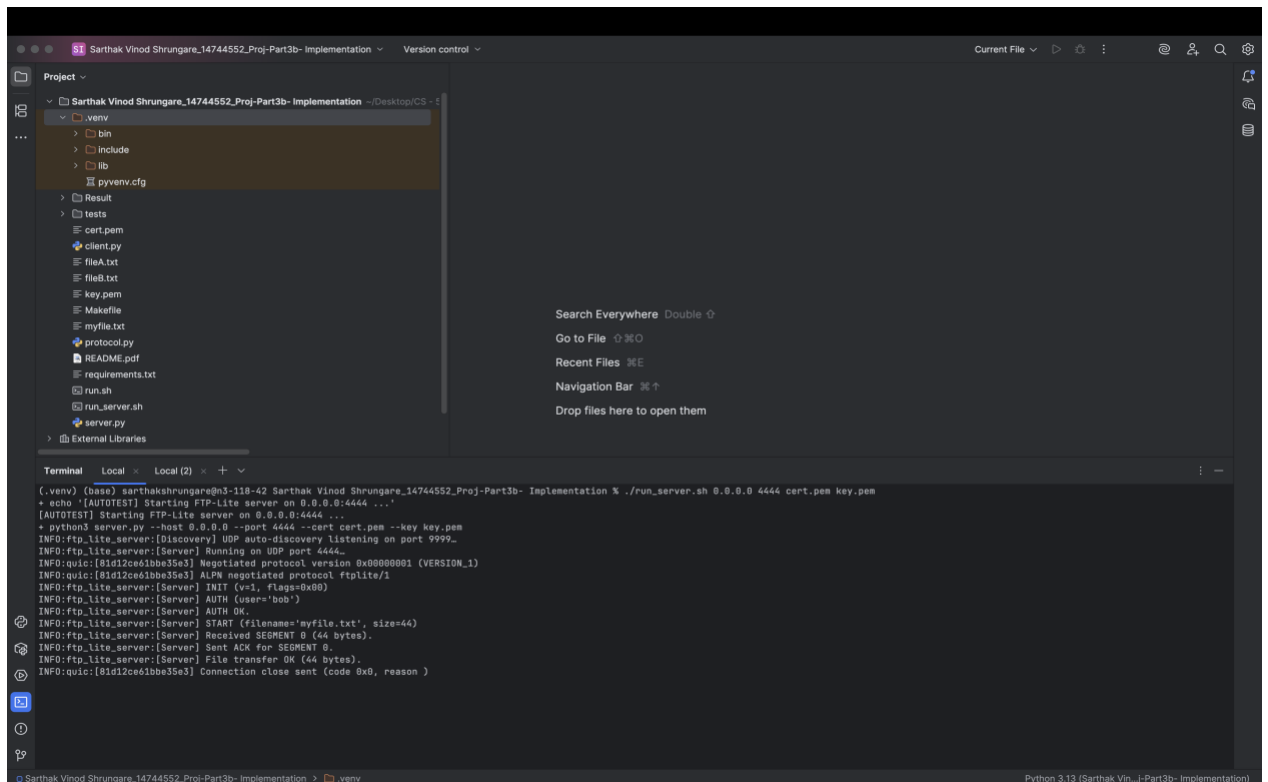

3. Running the Client:

- python3 client.py 127.0.0.1 myfile.txt --user bob --pass admin --cert cert.pem



The screenshot shows the Visual Studio Code interface with a project named 'Sarthak Vinod Shringare_14744552_Proj-Part3b- Implementation'. The file explorer on the left shows a directory structure with files like .venv, bin, include, lib, pyvenv.cfg, Result, tests, cert.pem, client.py, fileA.txt, fileB.txt, key.pem, Makefile, myfile.txt, protocol.py, README.pdf, requirements.txt, run.sh, run_server.sh, and server.py. The terminal at the bottom displays the output of running 'python3 client.py 127.0.0.1 myfile.txt --user bob --pass admin --cert cert.pem'. The output shows the client connecting to 127.0.0.1:4444, negotiating the FTP-Lite protocol, establishing a QUIC connection, and successfully authenticating the user 'bob'.

```
(.venv) (base) sarthakshringare@n3-118-42 Sarthak Vinod Shringare_14744552_Proj-Part3b- Implementation % python3 client.py 127.0.0.1 myfile.txt --user bob --pass admin --cert cert.pem
INFO:ftp_lite_client:[Client] Connecting to 127.0.0.1:4444 ...
INFO:quic:[81d12ce61bbe35e3] Negotiated protocol version 0x00000001 (VERSION_1)
INFO:quic:[81d12ce61bbe35e3] ALPN negotiated protocol ftp-lite/1
INFO:ftp_lite_client:[Client] QUIC connection established. Beginning FTP-Lite handshake...
INFO:ftp_lite_client:[Client] AUTH succeeded. Sending START_
INFO:ftp_lite_client:[Client] Sent SEGMENT 0 (44 bytes)
INFO:ftp_lite_client:[Client] Received ACK for segment 0
INFO:quic:[81d12ce61bbe35e3] Connection close received (code 0x0, reason )
```



The screenshot shows the Visual Studio Code interface with the same project. The terminal at the bottom displays the output of running './run_server.sh 0.0.0.0 4444 cert.pem key.pem'. The output shows the server starting an FTP-Lite server on 0.0.0.0:4444, listening on port 9999, and successfully authenticating the user 'bob'. It also shows the server sending an ACK for segment 0 and receiving a file transfer of 44 bytes.

```
(.venv) (base) sarthakshringare@n3-118-42 Sarthak Vinod Shringare_14744552_Proj-Part3b- Implementation % ./run_server.sh 0.0.0.0 4444 cert.pem key.pem
+ echo '[AUTOTEST] Starting FTP-Lite server on 0.0.0.0:4444 ...'
[AUTOTEST] Starting FTP-Lite server on 0.0.0.0:4444 ...
+ python3 server.py --host 0.0.0.0 --port 4444 --cert cert.pem --key key.pem
INFO:ftp_lite_server:[Discovery] UDP auto-discovery listening on port 9999...
INFO:ftp_lite_server:[Server] Running on UDP port 4444...
INFO:quic:[81d12ce61bbe35e3] Negotiated protocol version 0x00000001 (VERSION_1)
INFO:quic:[81d12ce61bbe35e3] ALPN negotiated protocol ftp-lite/1
INFO:ftp_lite_server:[Server] INIT (v=1, flags=0x00)
INFO:ftp_lite_server:[Server] AUTH (user='bob')
INFO:ftp_lite_server:[Server] AUTH OK
INFO:ftp_lite_server:[Server] START (filename='myfile.txt', size=44)
INFO:ftp_lite_server:[Server] Received SEGMENT 0 (44 bytes)
INFO:ftp_lite_server:[Server] Sent ACK for SEGMENT 0.
INFO:ftp_lite_server:[Server] File transfer OK (44 bytes)
INFO:quic:[81d12ce61bbe35e3] Connection close sent (code 0x0, reason )
```

4. Auto Discovery:

- python3 client.py auto myfile.txt --user bob --pass admin --cert cert.pem

The screenshot shows an IDE window with a project named "Sarthak Vinod Shringare_14744552_Proj-Part3b- Implementation". The file explorer on the left shows a directory structure with files like .venv, bin, include, lib, pyenv.cfg, Result, tests, cert.pem, client.py, fileA.txt, fileB.txt, key.pem, Makefile, myfile.txt, protocol.py, README.pdf, requirements.txt, run.sh, run_server.sh, and server.py. The terminal at the bottom shows the execution of the command: `python3 client.py auto myfile.txt --user bob --pass admin --cert cert.pem`. The output logs show the client attempting auto-discovery on UDP port 9999, discovering a server at 127.0.0.1, connecting to 127.0.0.1:4444, negotiating protocol version 0x00000001 (VERSION_1), establishing a QUIC connection, and sending a START segment. The connection is then closed with code 0x0 and reason.

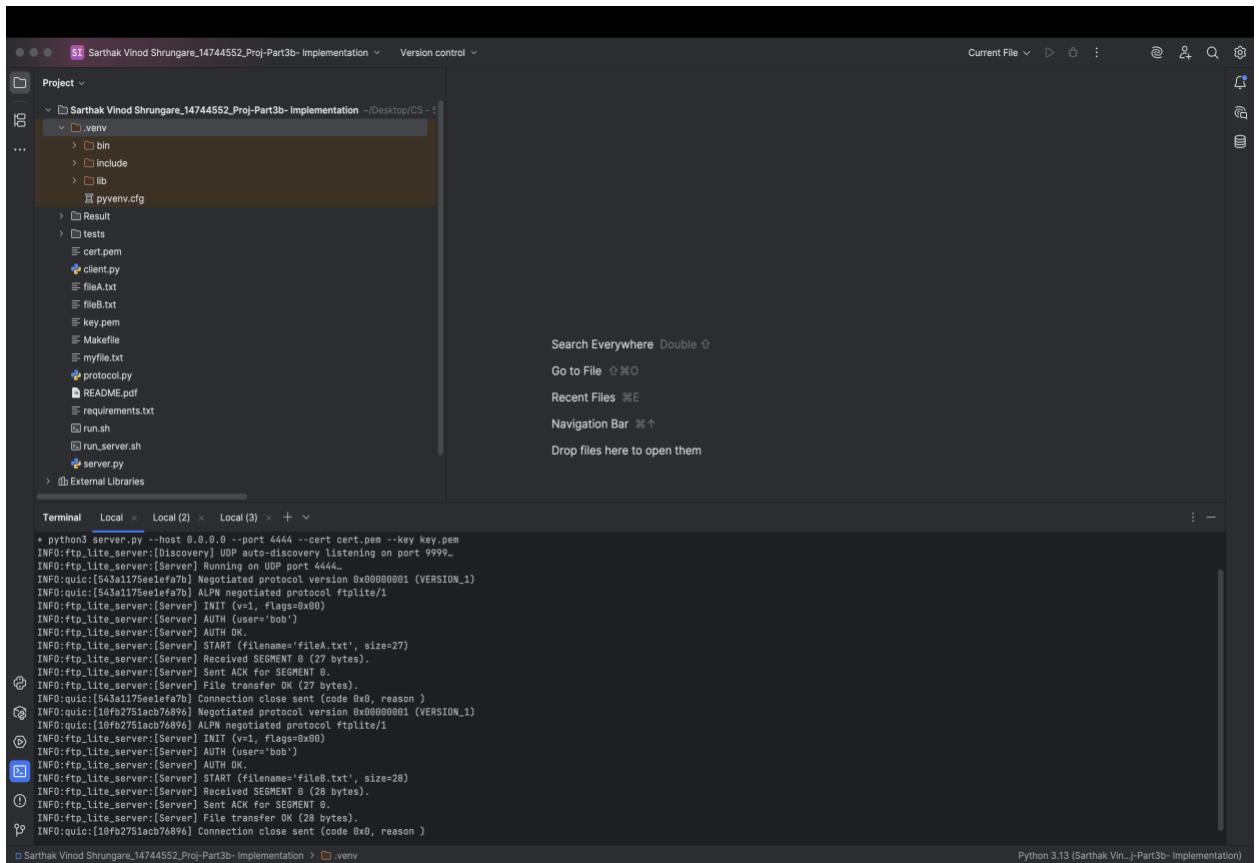
The screenshot shows the same IDE window with the same project. The terminal at the bottom shows the execution of the command: `./run_server.sh 0.0.0.0 4444 cert.pem key.pem`. The output logs show the server starting an FTP-Lite server on 0.0.0.0:4444, listening on port 9999, negotiating protocol version 0x00000001 (VERSION_1), establishing a QUIC connection, and receiving a START segment from the client. The server then sends an ACK for the segment and a file transfer OK message. The connection is then closed with code 0x0 and reason.

Testing Concurrency

Use 3 separate terminals:

- **Terminal 1 (Server):**

```
./run_server.sh 0.0.0.0 4444 cert.pem key.pem
```

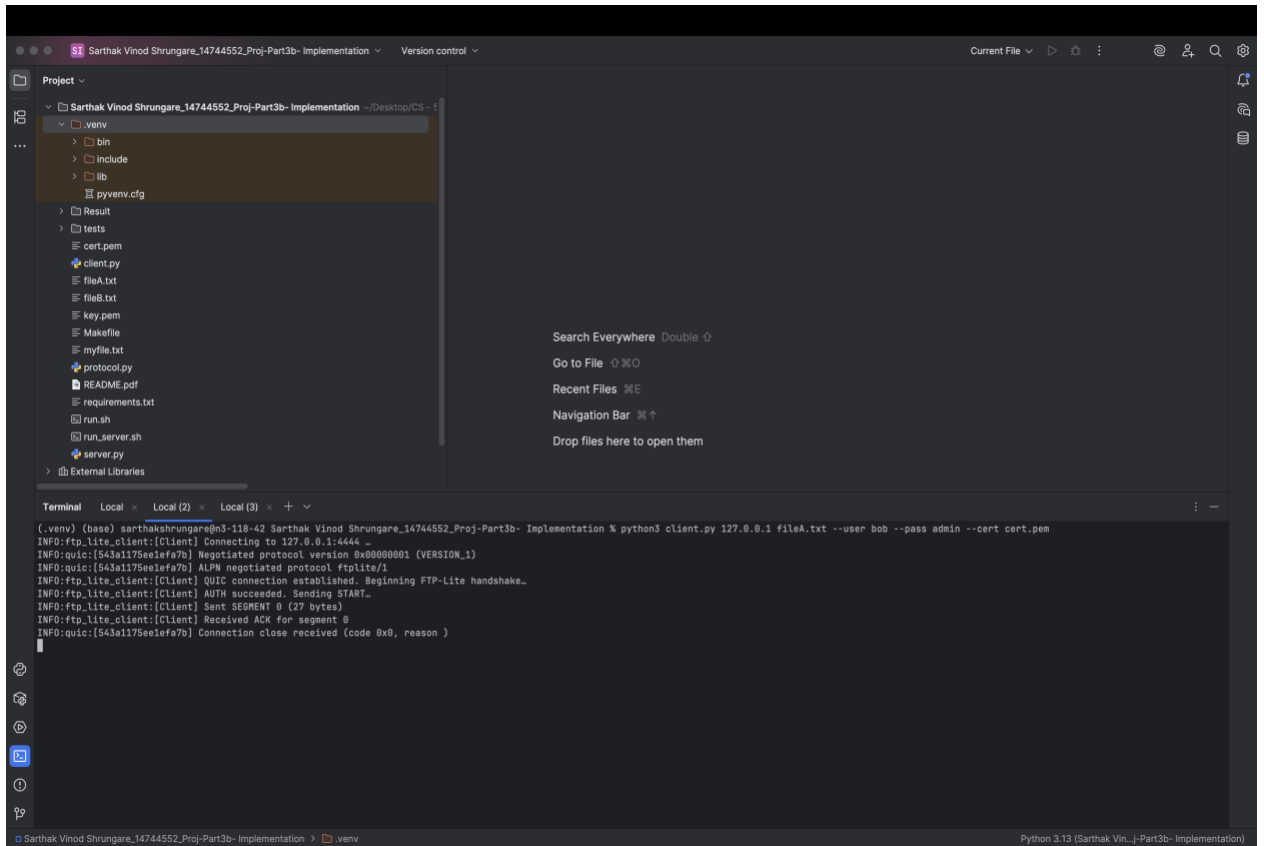


The screenshot shows an IDE interface with a project named "Sarthak Vinod Shringare_14744552_Proj-Part3b- Implementation". The project structure includes a ".venv" directory, "bin", "include", "lib", "pyvenv.cfg", "Result", "tests", "cert.pem", "client.py", "fileA.txt", "fileB.txt", "key.pem", "Makefile", "myfile.txt", "protocol.py", "README.pdf", "requirements.txt", "run.sh", "run_server.sh", and "server.py". The terminal window displays the following logs:

```
+ python3 server.py --host 0.0.0.0 --port 4444 --cert cert.pem --key key.pem
INFO:ftp_lite_server:[Discovery] UDP auto-discovery listening on port 9999..
INFO:ftp_lite_server:[Server] Running on UDP port 4444..
INFO:quic:[543a1175ee1efa7b] Negotiated protocol version 0x00000001 (VERSION_1)
INFO:quic:[543a1175ee1efa7b] ALPN negotiated protocol ftplite/1
INFO:ftp_lite_server:[Server] INIT (v=1, flags=0x00)
INFO:ftp_lite_server:[Server] AUTH (user='bob')
INFO:ftp_lite_server:[Server] AUTH OK.
INFO:ftp_lite_server:[Server] START (filename='fileA.txt', size=27)
INFO:ftp_lite_server:[Server] Received SEGMENT 0 (27 bytes).
INFO:ftp_lite_server:[Server] Sent ACK for SEGMENT 0.
INFO:ftp_lite_server:[Server] File transfer OK (27 bytes).
INFO:quic:[543a1175ee1efa7b] Connection close sent (code 0x0, reason )
INFO:quic:[10fb2751acb76896] Negotiated protocol version 0x00000001 (VERSION_1)
INFO:quic:[10fb2751acb76896] ALPN negotiated protocol ftplite/1
INFO:ftp_lite_server:[Server] INIT (v=1, flags=0x00)
INFO:ftp_lite_server:[Server] AUTH (user='bob')
INFO:ftp_lite_server:[Server] AUTH OK.
INFO:ftp_lite_server:[Server] START (filename='fileB.txt', size=28)
INFO:ftp_lite_server:[Server] Received SEGMENT 0 (28 bytes).
INFO:ftp_lite_server:[Server] Sent ACK for SEGMENT 0.
INFO:ftp_lite_server:[Server] File transfer OK (28 bytes).
INFO:quic:[10fb2751acb76896] Connection close sent (code 0x0, reason )
```

- **Terminal 2 (Client 1):**

```
python3 client.py 127.0.0.1 fileA.txt --user bob --pass admin --cert cert.pem
```

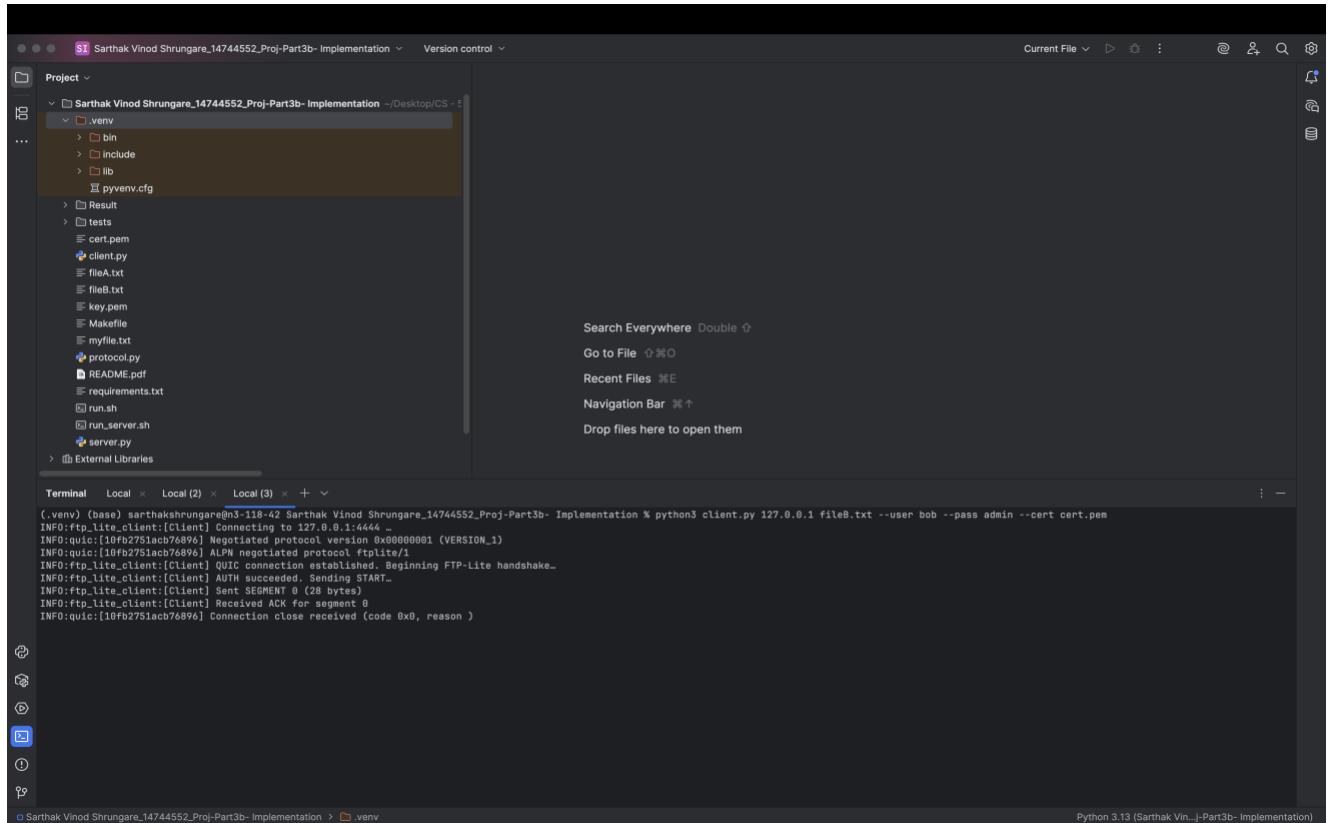


The screenshot shows a Visual Studio Code editor window with a project named "Sarathak Vinod Shrugare_14744552_Proj-Part3b- Implementation". The file explorer on the left shows a directory structure with files like `cert.pem`, `client.py`, `fileA.txt`, `fileB.txt`, `key.pem`, `Makefile`, `myfile.txt`, `protocol.py`, `README.pdf`, `requirements.txt`, `run.sh`, `run_server.sh`, and `server.py`. The terminal window at the bottom shows the execution of the command `python3 client.py 127.0.0.1 fileA.txt --user bob --pass admin --cert cert.pem`. The output of the command is as follows:

```
(.venv) (base) sarthakshrugare@n3-118-42 Sarthak Vinod Shrugare_14744552_Proj-Part3b- Implementation % python3 client.py 127.0.0.1 fileA.txt --user bob --pass admin --cert cert.pem
INFO:ftp_lite_client:[Client] Connecting to 127.0.0.1:4444 ...
INFO:quic:[543a1175ee1efa7b] Negotiated protocol version 0x00000001 (VERSION_1)
INFO:quic:[543a1175ee1efa7b] ALPN negotiated protocol ftplite/1
INFO:ftp_lite_client:[Client] QUIC connection established. Beginning FTP-Lite handshake..
INFO:ftp_lite_client:[Client] AUTH succeeded. Sending START..
INFO:ftp_lite_client:[Client] Sent SEGMENT 0 (27 bytes)
INFO:ftp_lite_client:[Client] Received ACK for segment 0
INFO:quic:[543a1175ee1efa7b] Connection cClose received (code 0x0, reason )
```

- **Terminal 3 (Client 2):**

```
python3 client.py 127.0.0.1 fileB.txt --user bob --pass admin --cert cert.pem
```



TP-Lite supports simultaneous uploads from multiple clients using Python's asyncio and QUIC.

This is tested by launching the server in one terminal and running two clients in parallel from separate terminals:

- **Client 1:** Uploads fileA.txt
- **Client 2:** Uploads fileB.txt

Both clients authenticate, initiate transfer, and send file chunks concurrently. The server maintains independent stateful sessions, processes PDUs asynchronously, and safely stores each file in the Result/ directory.

This demonstrates:

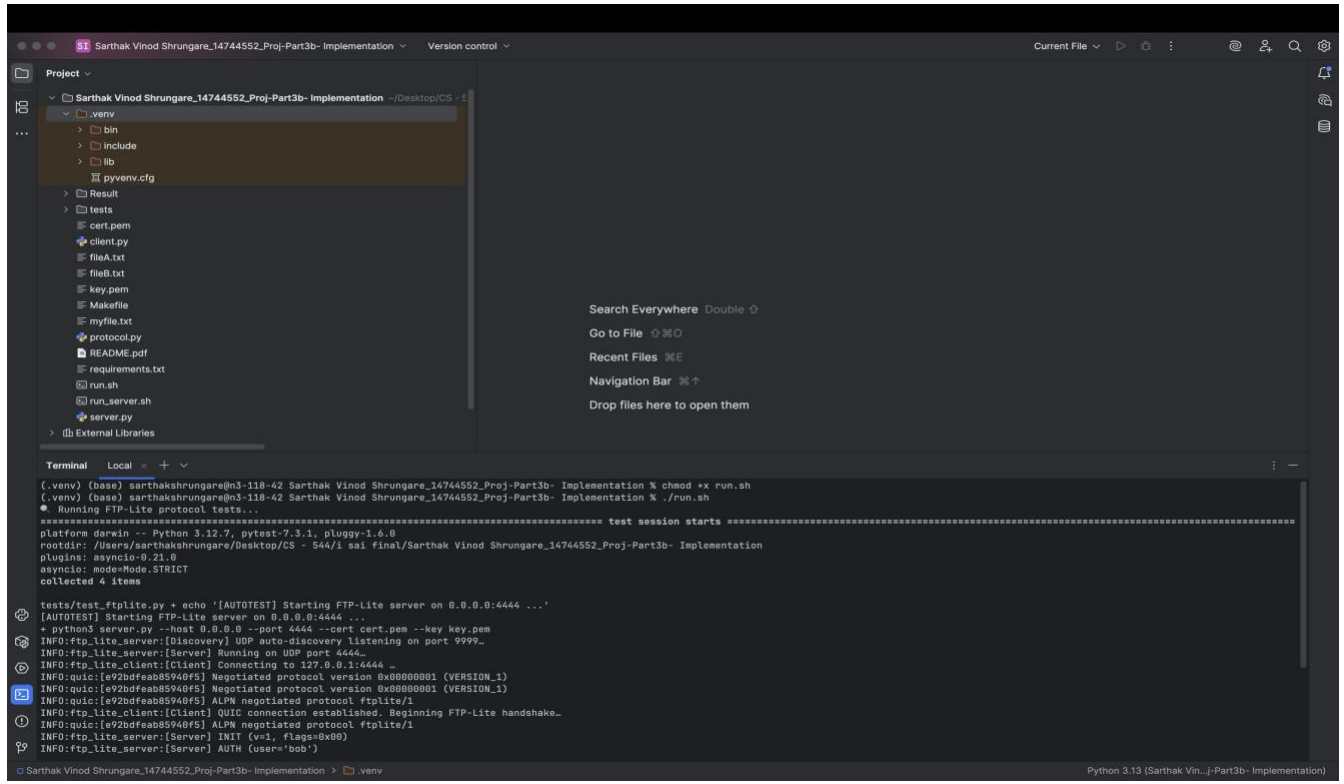
- True asynchronous concurrency using aioquic
- Accurate and reliable file transfer without blocking
- Protocol correctness under simultaneous load

This showcases FTP-Lite's scalability, reliability, and protocol robustness in real-world concurrent upload scenarios.

Automated Testing

Run:

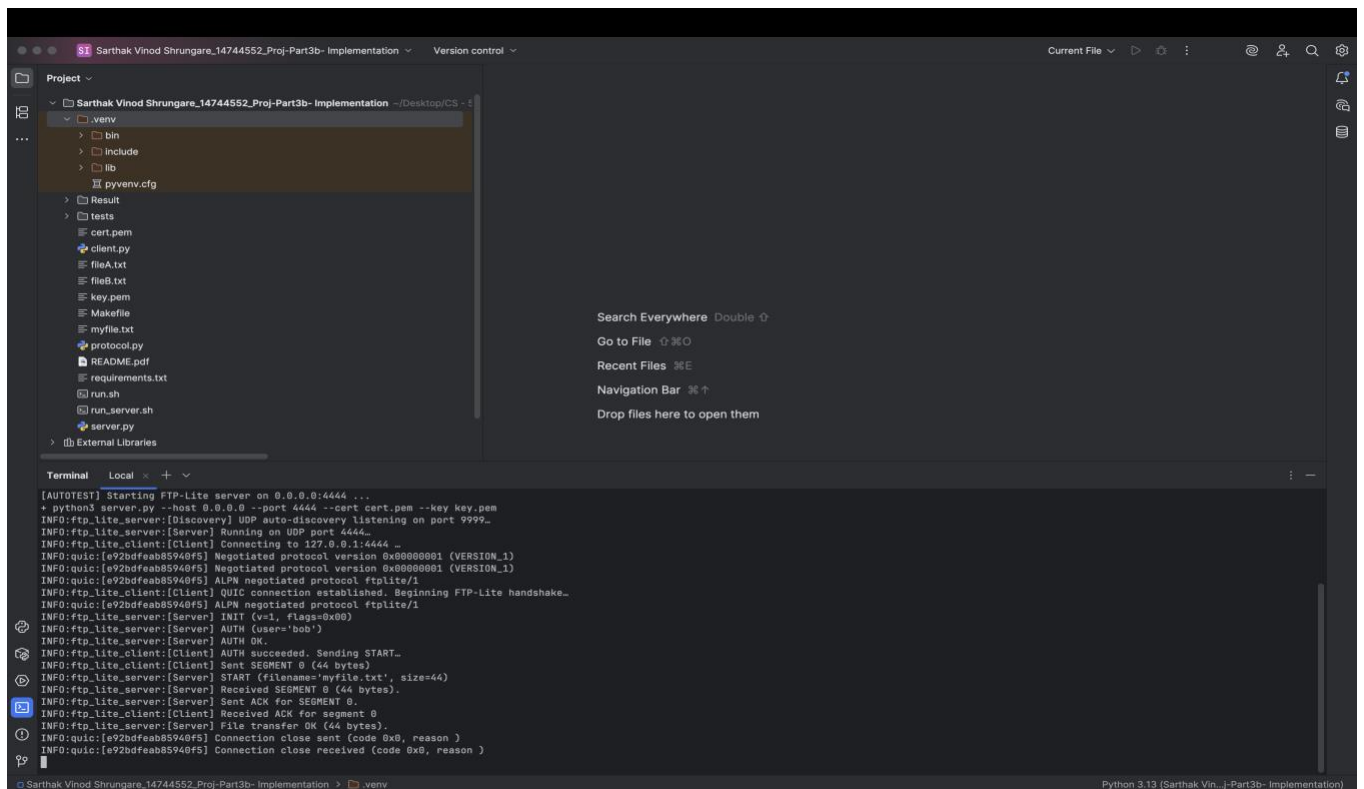
- `chmod +x run.sh`
- `./run.sh`



The screenshot shows an IDE interface with a project named "Sarthak Vinod Shrungare_14744552_Proj-Part3b- Implementation". The project structure includes a ".venv" directory, "bin", "include", "lib", "pyvenv.cfg", "Result", "tests", "cert.pem", "client.py", "fileA.txt", "fileB.txt", "key.pem", "Makefile", "myfile.txt", "protocol.py", "README.pdf", "requirements.txt", "run.sh", "run_server.sh", and "server.py". The terminal window displays the following output:

```
(.venv) (base) sarthakshrungare@n3-118-42 Sarthak Vinod Shrungare_14744552_Proj-Part3b- Implementation % chmod +x run.sh
(.venv) (base) sarthakshrungare@n3-118-42 Sarthak Vinod Shrungare_14744552_Proj-Part3b- Implementation % ./run.sh
• Running FTP-Lite protocol tests...
===== test session starts =====
platform darwin -- Python 3.12.7, pytest-7.3.1, pluggy-1.6.0
rootdir: /Users/sarthakshrungare/Desktop/CS - 544/i sai final/Sarthak Vinod Shrungare_14744552_Proj-Part3b- Implementation
plugins: asyncio-0.21.0
asyncio: mode=Mode.STRICT
collected 4 items

tests/test_ftplite.py + echo '[AUTOTEST] Starting FTP-Lite server on 0.0.0.0:4444 ...'
[AUTOTEST] Starting FTP-Lite server on 0.0.0.0:4444 ...
+ python3 server.py --host 0.0.0.0 --port 4444 --cert cert.pem --key key.pem
INFO:ftp_lite_server:[Discovery] UDP auto-discovery listening on port 9999_
INFO:ftp_lite_server:[Server] Running on UDP port 4444_
INFO:ftp_lite_client:[Client] Connecting to 127.0.0.1:4444 _
INFO:quic:[e92bdfcab85940f5] Negotiated protocol version 0x00000001 (VERSION_1)
INFO:quic:[e92bdfcab85940f5] Negotiated protocol version 0x00000001 (VERSION_1)
INFO:quic:[e92bdfcab85940f5] ALPN negotiated protocol ftplite/1
INFO:ftp_lite_client:[Client] QUIC connection established. Beginning FTP-Lite handshake_
INFO:quic:[e92bdfcab85940f5] ALPN negotiated protocol ftplite/1
INFO:ftp_lite_server:[Server] INIT (v=1, flags=0x00)
INFO:ftp_lite_server:[Server] AUTH (user='bob')
```



The screenshot shows the same IDE interface as the previous one, but with a different terminal output. The project structure is identical. The terminal window displays the following output:

```
[AUTOTEST] Starting FTP-Lite server on 0.0.0.0:4444 ...
+ python3 server.py --host 0.0.0.0 --port 4444 --cert cert.pem --key key.pem
INFO:ftp_lite_server:[Discovery] UDP auto-discovery listening on port 9999_
INFO:ftp_lite_server:[Server] Running on UDP port 4444_
INFO:ftp_lite_client:[Client] Connecting to 127.0.0.1:4444 _
INFO:quic:[e92bdfcab85940f5] Negotiated protocol version 0x00000001 (VERSION_1)
INFO:quic:[e92bdfcab85940f5] Negotiated protocol version 0x00000001 (VERSION_1)
INFO:quic:[e92bdfcab85940f5] ALPN negotiated protocol ftplite/1
INFO:ftp_lite_client:[Client] QUIC connection established. Beginning FTP-Lite handshake_
INFO:quic:[e92bdfcab85940f5] ALPN negotiated protocol ftplite/1
INFO:ftp_lite_server:[Server] INIT (v=1, flags=0x00)
INFO:ftp_lite_server:[Server] AUTH (user='bob')
INFO:ftp_lite_server:[Server] AUTH OK
INFO:ftp_lite_client:[Client] AUTH succeeded. Sending START_
INFO:ftp_lite_client:[Client] Sent SEGMENT 0 (44 bytes)
INFO:ftp_lite_server:[Server] START (filename='myfile.txt', size=44)
INFO:ftp_lite_server:[Server] Received SEGMENT 0 (44 bytes).
INFO:ftp_lite_server:[Server] Sent ACK for SEGMENT 0.
INFO:ftp_lite_client:[Client] Received ACK for segment 0
INFO:ftp_lite_server:[Server] File transfer OK (44 bytes).
INFO:quic:[e92bdfcab85940f5] Connection close sent (code 0x0, reason )
INFO:quic:[e92bdfcab85940f5] Connection close received (code 0x0, reason )
```

Above Automated Testing Includes tests for:

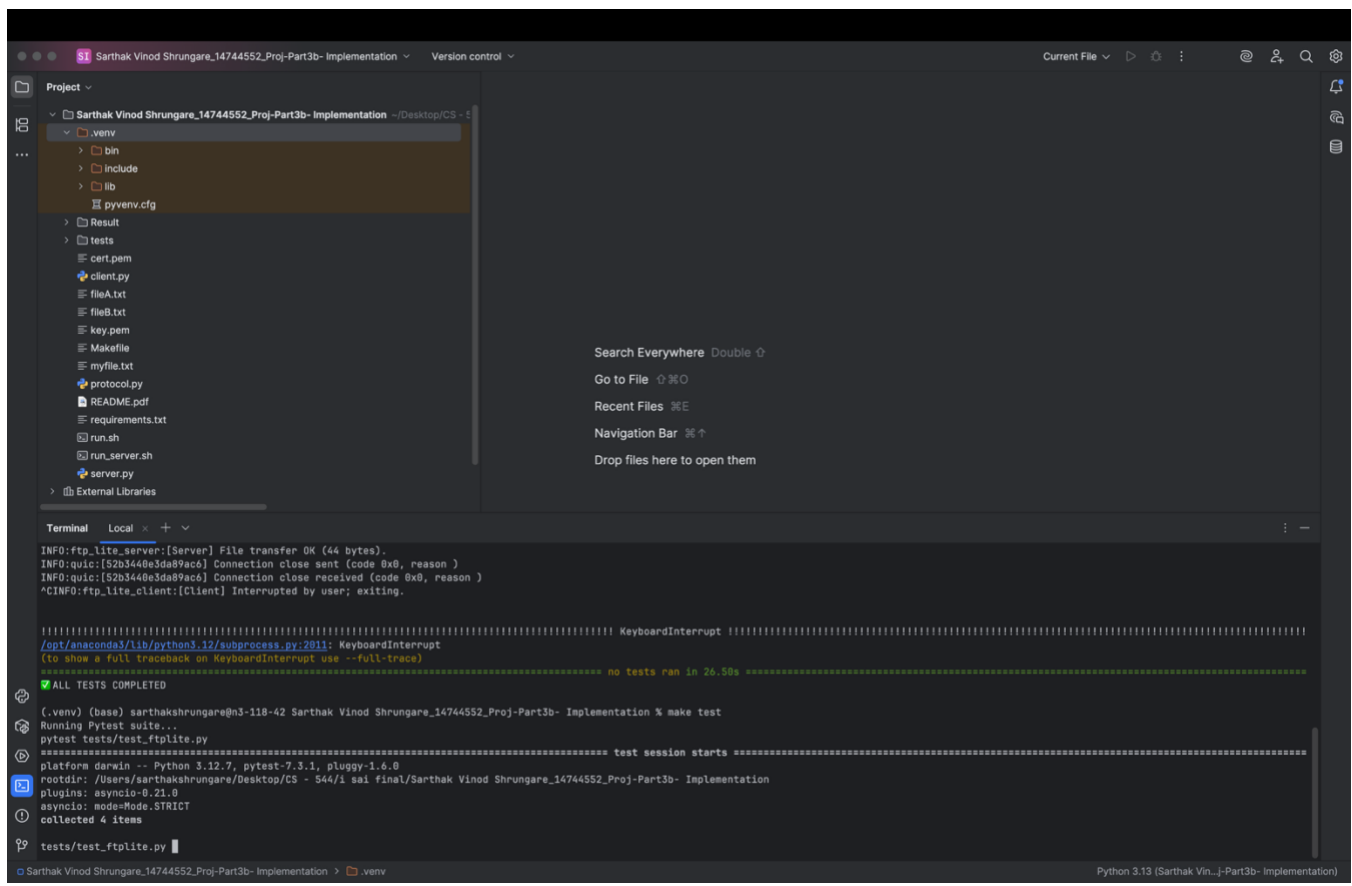
- Handshake + file upload.
- Authentication failure.
- Auto discovery.
- Fuzzing with invalid packets.

Makefile Support

To build or clean:

- make

Run server



The screenshot shows an IDE interface with a project structure on the left and a terminal window at the bottom. The project structure includes a `.venv` directory with `bin`, `include`, `lib`, and `pyvenv.cfg` files, as well as `Result`, `tests`, `cert.pem`, `client.py`, `fileA.txt`, `fileB.txt`, `key.pem`, `Makefile`, `myfile.txt`, `protocol.py`, `README.pdf`, `requirements.txt`, `run.sh`, `run_server.sh`, and `server.py` files. The terminal window shows the output of a `make test` command, indicating that all tests completed successfully. The terminal output includes the following text:

```
INFO:ftp_lite_server:[Server] File transfer OK (44 bytes).
INFO:quic:[52b3440e3da9ace] Connection close sent (code 0x0, reason )
INFO:quic:[52b3440e3da9ace] Connection close received (code 0xb, reason )
^CINFO:ftp_lite_client:[Client] Interrupted by user; exiting.

KeyboardInterrupt
/opt/anaconda3/lib/python3.12/subprocess.py:2811: KeyboardInterrupt
(to show a full traceback on KeyboardInterrupt use --full-trace)

no tests ran in 26.50s

ALL TESTS COMPLETED

(.venv) (base) sarthakshrungare@n3-118-42 Sarthak Vinod Shrungare_14744552_Proj-Part3b- Implementation % make test
Running Pytest suite...
pytest tests/test_ftplite.py
platform darwin -- Python 3.12.7, pytest-7.3.1, pluggy-1.6.0
rootdir: /Users/sarthakshrungare/Desktop/CS - 544/i sai final/Sarthak Vinod Shrungare_14744552_Proj-Part3b- Implementation
plugins: asyncio-0.21.0
asyncio: mode=Mode.STRICT
collected 4 items

tests/test_ftplite.py
```

- make test

Run test suite

```

INFO:ftp_lite_server:[Server] File transfer OK (44 bytes)
INFO:quic:[52b3440e3da89ac6] Connection close sent (code 0x0, reason )
INFO:quic:[52b3440e3da89ac6] Connection close received (code 0x0, reason )
^CINFO:ftp_lite_client:[Client] Interrupted by user; exiting.

===== KeyboardInterrupt =====
/opt/anaconda3/lib/python3.12/subprocess.py:2011: KeyboardInterrupt
(to show a full traceback on KeyboardInterrupt use --full-trace)
===== no tests ran in 26.50s =====
✓ ALL TESTS COMPLETED

(.venv) (base) sarthakshrungare@n3-118-42 Sarthak Vinod Shrungare_14744552_Proj-Part3b- Implementation % make test
Running pytest suite...
python -m pytest tests/test_ftplite.py
===== test session starts =====
platform darwin -- Python 3.12.7, pytest-7.3.1, pluggy-1.6.0
rootdir: /Users/sarthakshrungare/Desktop/CS - S44/1 sai final/Sarthak Vinod Shrungare_14744552_Proj-Part3b- Implementation
plugins: asyncio-0.21.0
asyncio: mode=Mode.STRICT
collected 4 items

tests/test_ftplite.py

```

- make clean

Clean up *.pyc or temp files

```

(.venv) (base) sarthakshrungare@n3-118-42 Sarthak Vinod Shrungare_14744552_Proj-Part3b- Implementation % make clean
Cleaning up...
rm -rf __pycache__ Result/*.txt *.pyc
(.venv) (base) sarthakshrungare@n3-118-42 Sarthak Vinod Shrungare_14744552_Proj-Part3b- Implementation %

```


File Transfer Workflow: Client-to-Server Interaction

FTP-Lite follows a structured, stateful sequence to securely and reliably transfer files from a client to a server using QUIC and TLS. The following steps summarize the end-to-end workflow:

1. Connection Initialization:

- The client initiates a secure QUIC connection to the server (host:port, default 127.0.0.1:4444) using the provided TLS certificate (cert.pem).
- If the auto mode is used, the client broadcasts on UDP port 9999 to discover the server dynamically.

2. INIT State:

- The client sends an INIT PDU indicating protocol version support.
- The server validates the version and responds with an acknowledgment, transitioning to the authentication phase.

3. AUTH State:

- The client submits credentials using an AUTH PDU (e.g., username: bob, password: admin).
- The server validates the credentials:
 - a) On success, it sends AUTH-OK.
 - b) On failure, it responds with AUTH-ERR and closes the connection.

4. START State:

- The client sends a START PDU containing the file name and file size.
- The server prepares to store the incoming file under the Result/ directory, preserving the original filename.

5. SEGMENT State:

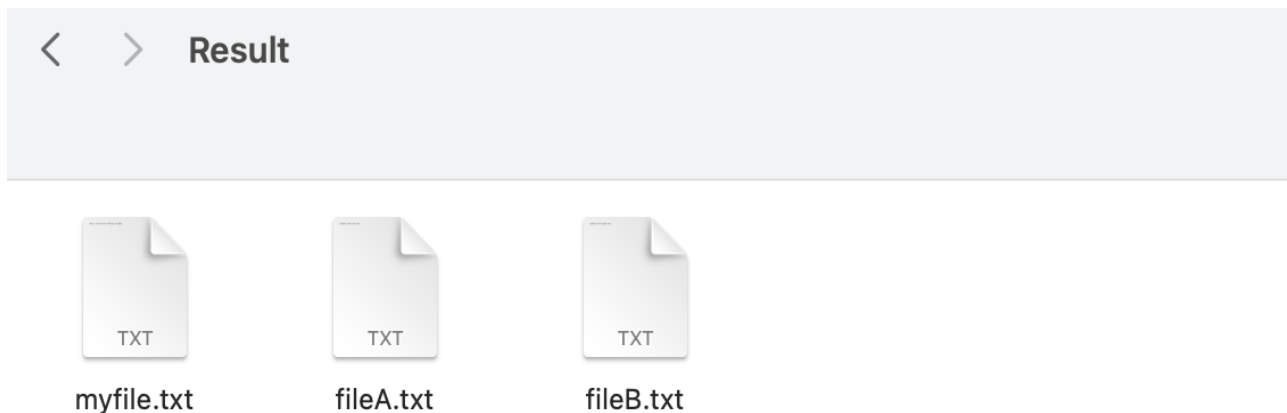
- The file is sent in binary chunks encapsulated in SEGMENT PDUs.
- Each segment includes a sequence number and payload.
- After receiving each segment, the server sends an ACK to confirm successful reception.
- This mechanism ensures ordered, reliable delivery and supports retransmission in case of failure.

6. COMPLETE State:

- Once all chunks are sent, the client issues a COMPLETE PDU.
- The server finalizes the file, closes the session, and logs a successful upload.

Result:

- All uploaded files are saved under the Result/ directory.
- The server maintains the original filenames (e.g., myfile.txt, fileA.txt, fileB.txt) and ensures data integrity.
- Concurrent uploads from multiple clients are supported via asyncio.



Feedback-Driven Design Evolution

During implementation, several changes were made to improve on the original design:

- Simplified PDU layout for easier parsing.
- Added file chunking and ACKs to improve large file reliability.
- Introduced auto discovery for user-friendly startup.
- Used asyncio and aiohttp to enable real-time concurrent uploads.

This feedback loop highlights how real-world constraints and testing shaped protocol decision.

Extra Credit Features Implemented

- **Concurrent Server:** Handles multiple uploads concurrently using asyncio
- **Automated Test Suite:** Located in tests/test_ftplite.py, includes handshake, auth, auto-discovery, fuzzing.
- **Dynamic Server Discovery:** Via UDP broadcast on port 9999 (client auto mode).
- **Extensibility:** Version field in PDUs supports upgrades.
- **Code Quality:** Modular Python code, commented functions, Makefile for build/test automation.
- **Feedback & Refinement:** Improved from initial specification during implementation.
- **GitHub Repository:** <https://github.com/shrungaresarthak/FTP-Lite.git>
- **README Documentation:** This file includes setup, usage, testing, protocol design, and extra credit coverage.

Conclusion:

FTP-Lite developed from a design specification to a working and testable system capable of simulating many of the complications of real-world protocol stacks. It also includes modern transport technologies (QUIC and TLS) combined with structured and consistent application-layer design principles such as deterministic finite states automata (DFA), application-layer PDU structure, and stateful sessions. It serves to illustrate both practical experience in protocol design and emphasize robustness through capabilities such as asyncio for concurrent client handling, UDP broadcast server/client discovery, and pytest for automated protocol conformance testing. It serves as a practical prototype of how lightweight file transfer systems for the modern world may be designed and implemented using contemporary architectures that are secure and extensible.