Assignment 2: Peer-to-Peer File Synchronizer Report (Test Cases & Results)

Zifan Si (Student #400265867)

February 22, 2026

The complete implementation, including source code, batch scripts, and test suites, is available at the following repository:

GitHub: `https://github.com/ZifanSi/p2p-file-synchronizer`

Codes in the `src/` directory, documentation in the `docs/` directory.

1 Overview

This project implements a peer-to-peer (P2P) file synchronizer using a centralized tracker for peer discovery and directory aggregation. Each peer:

- establishes one persistent TCP connection to the tracker,
- advertises working-directory file metadata (name, integer mtime) on startup (Init),
- sends periodic keepalive messages to maintain liveness (every 5 seconds),
- receives a directory response from the tracker after each Init/KeepAlive,
- downloads files that are missing locally or have older mtimes,
- serves file requests to other peers using a TCP protocol with a `Content-Length` header and raw bytes.

2 Environment

- OS: Windows 10/11 (local testing)
- Python: 3.12.10
- Network: localhost (127.0.0.1)

3 Project Layout Used for Testing

```
src/
  bats/
    clean_peer.bat
    clean_port.bat
  Peer1/
    fileA.txt
  Peer2/
    fileB.txt
```

```
  Peer3/
    fileC.txt
    (generated in TC6) big.bin
  tests/
    compare_files.py
    tc_4.py
    tc_5.py
    tc_6.py
  fileSynchronizer.py
  run_all.bat
  tracker.py
```

4 How to Run

Reset scripts (recommended for repeatable tests)

```
cd C:\Users\sifra\Desktop\p2p-file-synchronizer
src\bats\clean_peer.bat (will reset peer folders to initial state with only fileA/B/C)
src\bats\clean_port.bat (will stop all peers and tracker, useful for testing port binding
     and clean restarts)
```

Start tracker + peers (opens multiple terminals)

```
cd C:\Users\sifra\Desktop\p2p-file-synchronizer
src\run_all.bat
```

5 Protocol Summary

5.1 Peer → Tracker

Messages are newline-terminated UTF-8 JSON objects.

- Init (sent exactly once on startup):

  ```
  {"port": <p>, "files": [{"name": "...", "mtime": <int>}, ...]}\n
  ```

- KeepAlive (sent every 5 seconds):

  ```
  {"port": <p>}\n
  ```

5.2 Tracker → Peer

For every Init/KeepAlive received, the tracker returns one directory response (newline-terminated JSON):

```
{"fileA.txt": {"ip":"...", "port":..., "mtime":...}, ...}\n
```

5.3 Peer ↔ Peer

Requester sends:

```
<filename>\n
```

Server responds:

```
Content-Length: <size>\n
<raw file bytes>
```

6 Rubric Coverage Map

The following test cases are designed to cover every graded requirement:

- TC0: `get_file_info()` (1pt)
- TC1: `get_next_available_port()` (1pt)
- TC2: `FileSynchronizer` initializer + tracker communication (1pt)
- TC3: `sync()` discovery + retrieve missing files (1pt)
- TC4: `sync()` overwrite newer mtime + `os.utime()` (1pt)
- TC5: `process_message()` serves file with correct `Content-Length` framing (2pt)
- TC6: timeout/failure handling + discard partial file (1pt)

7 Selected Runtime Output Evidence (localhost run)

The following excerpts are copied from an actual run on localhost using:

- Tracker port: **9000**
- Peer1 port: **8000**
- Peer2 port: **8001**
- Peer3 port: **8002**

Tracker (excerpt)

```
Waiting for connections on port 9000
Client connected with 127.0.0.1:51543
Client connected with 127.0.0.1:51545
Client connected with 127.0.0.1:51546
client server127.0.0.1:8000
client server127.0.0.1:8001
client server127.0.0.1:8002
```

Peer directory responses (excerpt)

```
Peer1:
Waiting for connections on port 8000
('connect to:127.0.0.1', 9000)
received from tracker: {"fileA.txt": {"ip": "127.0.0.1", "port": 8000, "mtime":
    1771289241}}
```

```
received from tracker: {"fileA.txt": {"ip": "127.0.0.1", "port": 8000, "mtime":
    1771289241},
 "fileB.txt": {"ip": "127.0.0.1", "port": 8001, "mtime": 1771289244},
 "fileC.txt": {"ip": "127.0.0.1", "port": 8002, "mtime": 1771289247}}

Peer2:
Waiting for connections on port 8001
('connect to:127.0.0.1', 9000)
received from tracker: {"fileA.txt": {"ip": "127.0.0.1", "port": 8000, "mtime":
    1771289241},
 "fileB.txt": {"ip": "127.0.0.1", "port": 8001, "mtime": 1771289244}}

Peer3:
Waiting for connections on port 8002
('connect to:127.0.0.1', 9000)
received from tracker: {"fileA.txt": {"ip": "127.0.0.1", "port": 8000, "mtime":
    1771289241},
 "fileB.txt": {"ip": "127.0.0.1", "port": 8001, "mtime": 1771289244},
 "fileC.txt": {"ip": "127.0.0.1", "port": 8002, "mtime": 1771289247}}
```

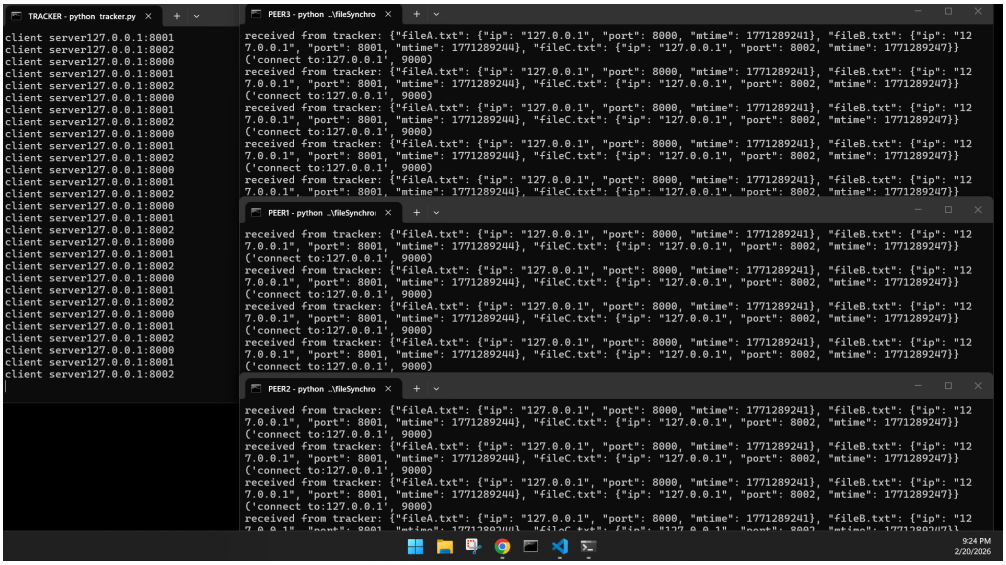

Figure 1: Output of the P2P File Synchronizer

8 Test Cases and Results

8.1 TC0 (1pt): Working-directory file filtering via `get_file_info()`

Goal: Verify only valid files in the working directory are advertised; ignore subdirectories and `.py`/`.dll`/`.so`; report integer mtimes.

Setup: In `src/Peer1/` create:

- `fileA.txt`
- `junk.dll`, `junk.so`, `temp.py`
- subfolder `sub/` containing `subfile.txt`

Steps:

1. Start tracker and peers.
2. Observe directory response received by any peer.

Expected:

- Only `fileA.txt` appears in the advertised directory from Peer1.
- `mtime` values are integers.

Observed: Only `fileA.txt` was listed (filtered files and subfolder were not). Example from directory response:

```
"fileA.txt": {"ip": "127.0.0.1", "port": 8000, "mtime": 1771289241}
```

Result: PASS.

8.2 TC1 (1pt): Next available port selection

Goal: Verify each peer binds to an available port (no collisions).

Steps:

1. Start tracker (9000).
2. Start three peers.

Expected: Each peer prints a listening port and no bind errors occur.

Observed (this run):

```
Peer1: Waiting for connections on port 8000
Peer2: Waiting for connections on port 8001
Peer3: Waiting for connections on port 8002
```

Result: PASS.

8.3 TC2 (1pt): Initializer + tracker connection + keepalive cycles

Goal: Verify sockets are initialized correctly, peers connect to tracker, send Init, and continue with KeepAlive (directory responses repeated over time).

Steps:

1. Start tracker and peers.
2. Observe tracker accepts connections and peers receive directory responses.
3. Wait ≥ 10 seconds and confirm additional directory responses occur (keepalive cycles).

Expected: Tracker logs 3 connections; peers receive directory JSON repeatedly.

Observed: Tracker showed three connections (ports 51543, 51545, 51546) and peers repeatedly printed `received from tracker: {...}`.

Result: PASS.

8.4 TC3 (1pt): Discovery and retrieve missing files (convergence)

Goal: Verify peers download missing files and converge to the same directory.

Setup (before run):

- Peer1 has only `fileA.txt`
- Peer2 has only `fileB.txt`
- Peer3 has only `fileC.txt`

Steps:

1. Run `src\bats\clean_peer.bat`.
2. Start tracker and peers.
3. Wait 1–2 sync cycles.
4. Check each peer folder contents.

Expected: Each peer ends with A/B/C.

Observed (after run):

```
Peer1: fileA.txt fileB.txt fileC.txt
Peer2: fileA.txt fileB.txt fileC.txt
Peer3: fileA.txt fileB.txt fileC.txt
```

Result: PASS.

8.5 TC4 (1pt): Overwrite newer version + verify mtime set via `os.utime()`

Goal: Verify a newer file version overwrites older copies and that the local mtime matches the advertised mtime.

Steps:

1. Stop all peers (or run `src\bats\clean_port.bat`).
2. Append a line to `src/Peer2/fileB.txt` and save (mtime increases).
3. Restart tracker and peers; wait 1–2 sync cycles.
4. Record mtimes using the helper script:

```
cd src
python tests\tc_4.py Peer2\fileB.txt
python tests\tc_4.py Peer1\fileB.txt
python tests\tc_4.py Peer3\fileB.txt
```

5. Confirm content equality using:

```
cd src
python tests\compare_files.py Peer2\fileB.txt Peer1\fileB.txt
python tests\compare_files.py Peer2\fileB.txt Peer3\fileB.txt
```

Expected:

- Peer1/Peer3 overwrite their `fileB.txt` to match Peer2.
- Peer1/Peer3 integer mtimes equal Peer2's newer mtime (set by `os.utime()`).

Observed: After restart and synchronization, Peer1 and Peer3 matched Peer2's updated file (hash match via `compare_files.py`) and mtimes (via `tc_4.py`).

Result: PASS.

8.6 TC5 (2pt): Peer serving protocol (`Content-Length` header + exact bytes)

Goal: Verify server replies with correct header and sends exactly that many bytes.

Steps:

1. Ensure peers are running. In this run, Peer3 served files on port **8002**.
2. Run:

```
cd src
python tests\tc_5.py --port 8002 --file fileC.txt
```

Expected:

- Prints `Content-Length:  N`
- Prints `bytes=N expected=N`

Observed:

```
Content-Length: 38
bytes=38 expected=38
```

Result: PASS.

8.7 TC6 (1pt): Timeout/failure + discard partial file

Goal: Verify failed transfers do not leave partial files and synchronization can recover after peer restart.

Setup: Create a larger file in Peer3 using the helper script:

```
cd src\Peer3
python ..\tests\tc_6.py
```

Steps:

1. Start tracker and peers; confirm directory includes `big.bin`.
2. Delete `src/Peer1/big.bin` if present.
3. Wait for sync to begin, then force-stop Peer3 while transfer is in progress (close the Peer3 terminal).
4. Verify Peer1 does not keep a partial file (no `big.bin.part` remains).
5. Restart Peer3 and wait another sync cycle.
6. Verify the final downloaded file matches Peer3 using:

```
cd src
python tests\compare_files.py Peer3\big.bin Peer1\big.bin
```

Expected:

- If the transfer is interrupted, Peer1 does not keep a partial file (no `.part` remains).
- After Peer3 restarts, Peer1 successfully downloads `big.bin` and it matches Peer3.

Observed: `tc_6.py` generated `big.bin`. When Peer3 was force-stopped mid-transfer, `big.bin.part` was not left behind. After restarting Peer3, Peer1 retrieved `big.bin` and it matched Peer3 (verified by `compare_files.py`).

Result: PASS.

9 Results Summary

Rubric Requirement / Test Case	Result
TC0: `get_file_info()` filtering rules (1pt)	PASS
TC1: `get_next_available_port()` (1pt)	PASS
TC2: Initializer + tracker comms + keepalive (1pt)	PASS
TC3: `sync()` discovery + retrieve missing (1pt)	PASS
TC4: `sync()` overwrite newer + set mtime (1pt)	PASS
TC5: `process_message()` Content-Length + exact bytes (2pt)	PASS
TC6: Failure handling + discard partial file (1pt)	PASS

10 Conclusion

Custom test cases were designed to validate all protocol requirements and grading rubric items. The results demonstrate correct working-directory discovery and filtering, available-port binding, persistent tracker communication and keepalives, peer serving with `Content-Length` framing and exact byte transfer, convergence across peers, propagation of newer file versions with mtime preservation via `os.utime()`, and robust handling of failures without leaving partial files.