Assignment 2: Peer-to-Peer File Synchronizer

COMPSCI 4C03: Computer Networks

Name: Shivom Sharma

Student ID: 400332395

Instructor: Prof. Rong Zheng

1. Test Cases and Results

Note: This report documents the test cases and their execution results verifying the requirements of the File Synchronizer assignment based on the provided rubric.

1. Local File Info (`get_file_info`)

Objective: Verify that the `get_file_info()` method correctly returns file information in the local directory.

Test Design:

Describe your test setup here. For example, mention that you created sample files with known contents/timestamps in the local directory and called `get_file_info()`.

Results:

Describe the results here. You can include images of terminal output verifying that file names, timestamps, and other metadata are correctly retrieved.

2. Next Available Port (`get_next_available_port`)

Objective: Verify that the `get_next_available_port()` method correctly returns the next available port for the peer.

Test Design:

Describe how you tested port allocation. For example, testing sequential port binding or simulating ports already in use.

Results:

Describe the outcomes. State that it successfully bypassed bound ports and returned the first available open port.

3. Initialization (`__init__`)

Objective: Verify the complete initialization of the `FileSynchronizer` class.

Test Design:

Explain your initialization test. How did you verify that properties, local file states, and server sockets were initialized successfully?

Results:

Show that creating an instance successfully assigns the proper state without throwing errors.

4. Serving Files (`process_message`)

Objective: Verify that `process_message()` properly returns the requested file with the correct header to a peer.

Test Design:

Detail how you simulated a peer requesting a specific file. Explain the structure of the request message you sent and how the peer is expected to handle it.

Results:

Confirm that the receiving peer got the correct file content along with the expected HTTP-like response headers.

5. Discovering and Retrieving Files (`sync`)

Objective: Verify that `sync()` discovers and retrieves new files from other peers.

Test Design:

Describe the multi-peer scenario. Setup Peer A with a new file, and trigger `sync()` on Peer B to fetch the tracker's list and detect the new file.

Results:

Show that Peer B successfully discovered the new file, requested it from Peer A, and saved it properly.

6. Overwriting Outdated Files (`sync`)

Objective: Verify that `sync()` overwrites a local file if a newer version exists on another peer.

Test Design:

Explain the test: Peer A modifies an existing file (updating its modification time) while Peer B has the old version. Peer B calls `sync()`.

Results:

Provide evidence that the file was updated on Peer B, matching Peer A's content and timestamp.

7. Failure Handling

Objective: Verify that the implementation gracefully handles tracker and peer timeout/failure scenarios.

Test Design:

Detail the tests for connection timeouts or unexpected drop-offs. Examples: terminating the tracker unexpectedly, or terminating a peer midway through a request.

Results:

Show that the peer logs the failure and continues running without a fatal crash or handles the exception cleanly.