Programming Assignment 3: P2P File Sharing Synchronizer

Due date: March 29th, 11:59pm (by timestamps in gitlab)

In this programming assignments, you will complete the simple P2P file sharing application that synchronize files among peers. It has the flavor of Dropbox but instead of storing and serving files using the cloud, we utilize a peer-to-peer approach. Recall that the key difference between the client-server architecture and the peer-to-peer architecture is that in the later, the end host acts both as a server (e.g., to serve file transfer requests) and as a client (e.g., to request a file). One challenge in peer-to-peer applications is that peers do not initially know each other's presence. As such, a server node, called tracker in this project, is needed to facilitate the "discovery" of peers. In PA2, you have developed the tracker program. In this assignment, you will develop the file synchronizer program for the peers.

In the PA3 folder, you will find

- ✓ "instructions.docx" - this document.
- ✓ "Skeleton.py" - the skeleton code for the **file synchronizer** you will develop.

Specifications:

- The synchronizer takes as command line arguments the IP address and port number (Port A) of the tracker running on.
- The synchronizer chooses **an available port** (Port B) to bind and listen for file request. This port will be used to accept incoming connection requests and **transfer files to other peers**.
- The synchronizer **communicates with the tracker** through a single TCP socket to
    1. send an **Initial message** containing names and the last modified time (round down to integer if it is a float) of its local files as well as **file serving port** (Port B) when it first starts.
    2. send **keepalive** messages containing the **file serving port** every **5** seconds to the tracker
    3. **receive the directory information** maintained by the tracker
- Upon receiving a **Directory Response message** from the tracker, the synchronizer parses the message and identifies files that are **either new or more up-to-date** than its own copy by comparing with the **modified time** of its local files. If such a file is found, it **connects the corresponding peer** using the peer's IP address and Port number contained in the message. Upon a successful connection, it sends a **File Request message** with the name of the requested file to the said peer. The file from the peer shall be stored in the same directory that the synchronizer runs. Make sure to set the modified time for the newly retrieved file to the **modified time** of the file in the directory response message. This process is repeated until all the new or more up-to-date files contained in the Directory Response Message are fetched.
- Upon receiving **a File Request message** from a peer, the synchronizer responds with the content of the file.
- In case that connection to a peer times out (e.g., using socket.settimeout) or there is no response to the **File Request message**, the synchronizer should proceed to the next file if any in the **Directory Response message**.
- The synchronizer should close client sockets that are not in use to other peers.

As clear from the specification, the file synchronizer implements both **a server and a client**. It **acts as a server to respond to file requests from other peers**. It is also **a client when requesting files from other peers**. Therefore, it is advisable to use multi-thread programming for this purpose. The state

diagram of the file synchronizer is given in Figure 1. Messages exchanged between tracker and peers (synchronizers) are encoded in JSON as specified in Table 1. Messages exchanged between peers are NOT JSON, they are either plain text (filename, text file content) or binary (binary file content) as listed in Table 2.

Table 1 Message Format between tracker and synchronizer

Message Type	Purpose	Message Format (key, value)	Actions Followed the Message
Initial Message	From **peers to the tracker**: (1) Upload file information and file serving port. (2) Request directory information stored in the tracker.	{'port':int, 'files': [{'name':string,'mtime':int}]} Note: round the mtime to integer if it is a float. e.g., {"port": 8001, "files": [{"mtime": 1548878750, "name": "fileA.txt"}]}	Tracker responds with a directory response message (See the **third row for the format**)
Keep-alive Messsage	From **peers to the tracker**: (1) inform the tracker that the peer is still alive.	{'port':int} e.g., {"port": 8001}	(1) The tracker refreshes its timer for the respective peer (2) The tracker responds with a directory response message (**third row** for format)
Directory Response Message	From **the tracker to peers**: (1) Return the directory at the tracker.	{filename:{'ip':,'port':,'mtime':} e.g., {"fileB.txt": {"ip": "127.0.0.1", "mtime": 1548878750, "port": 8002}, "fileA.txt": {"ip": "127.0.0.1", "mtime": 1548878750, "port": 8001}}	Peer retrieve new or modified files from other peers

Table 2 Message Format between peers

File Request Message	From **peer to peer:** Request a new or more up-to-date file on the peer.	**Plain text** of the **file name**	Peer responds with the content of the file.
File Response Message	From **peer to peer**: Send the content of the requested file	**Content of the file**, it can be binary or text	Peer saves the received file locally.

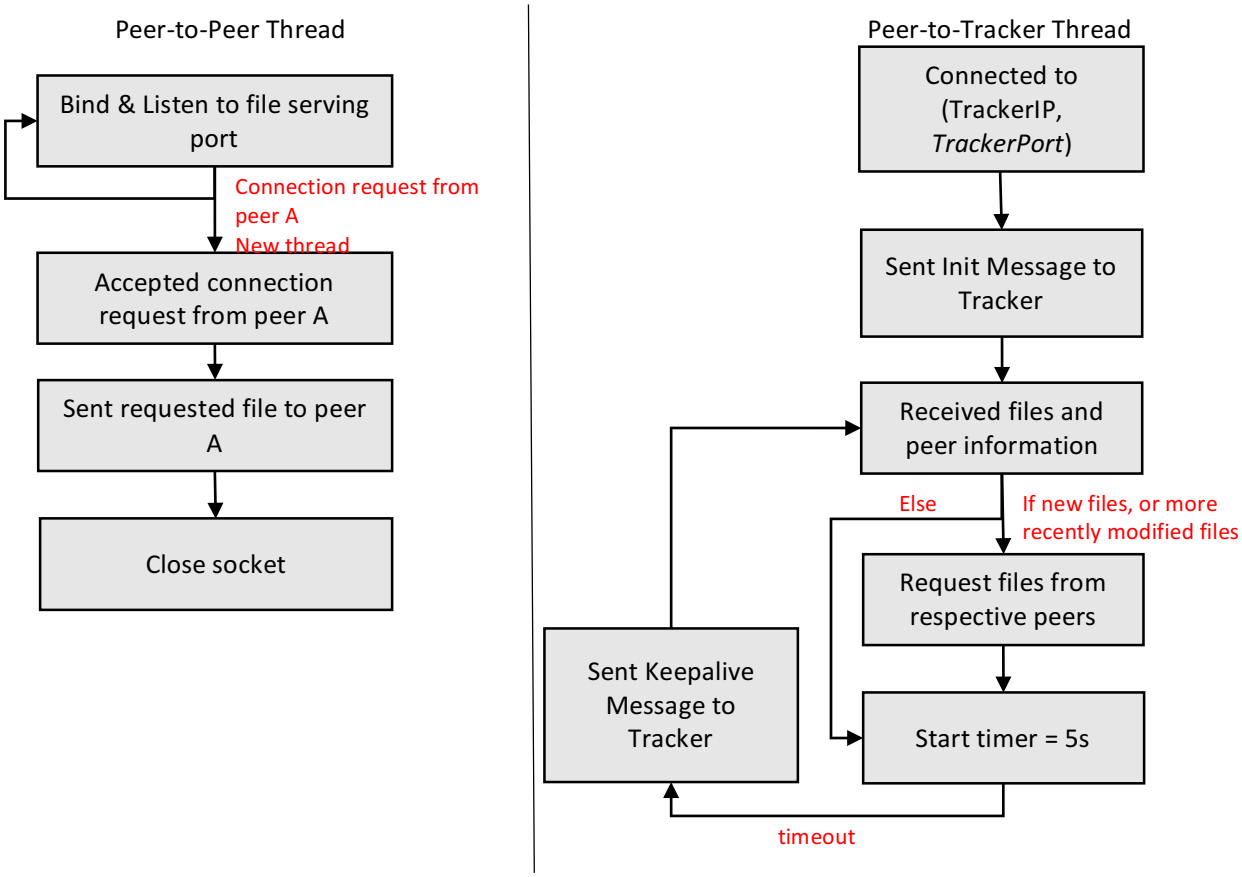

Figure 1 State Diagram of the file synchronizer.

For simplicity,
1) We only consider files within the same folder where the peer or synchronizer runs (**no subfolders**).
2) We do not consider changes **after** peers start, such as file deletion, modification, and addition. In other words, **the initial message that contains local file information is sent only once**.
3) If multiple files from different peers have the same file name, the one with the **last** modified timestamp will be kept by the tracker.
4) Only one TCP connection is needed between a peer and a tracker.

Steps to begin PA3:

1. Read this document carefully to understand the logic.
2. Download a binary tracker suitable for your platform from https://www.dropbox.com/sh/b4sjpezmy1icqx4/AABEsZC3ZELul_3IGXLo_b_-a?dl=0 for testing.

3. Copy Skeleton.py to “fileSynchronizer.py” and go over the skeleton to be familiar with the data structures used.
4. Start coding and testing.
5. Enjoy! Any questions please discuss through piazza or go to lab sessions.

Note:

1. Your implementation should be based on Python 3 for compatibility.
2. Use ‘ctrl’ + ‘\’ to terminate a running tracker or peer.
3. It is a good programming practice to perform unit testing for each step of your development when key new functionality is implemented. You may test your program on the same host with different port numbers or different hosts.
4. For environment setup or test procedure, please refer to PA2.

Submission:

1. Submit your source code in a SINGLE file called ‘fileSynchronizer.py’ to ‘PA3/ fileSynchronizer.py’ folder in your gitlab project (DO NOT USE OTHER NAMES!).
2. Document your test cases and results in ‘PA3/report.pdf’ in your gitlab project.