CS 3251: Computer Networking I (Section A and B) **Spring 2023**

Programming Assignment 2

Handed Out: Feb 25^{th}*,* 2023 *Due: 11:59pm, April* 13^{th}*,* 2023

1 Early Release

This assignment is released in "early release" mode. We are releasing the text of the programming assignment so that you can get started with the program architecture and collecting all the related information you might need. We are working on creating a robust Gradescope test system for this programming assignment and therefore might have to make small modifications to the text of this programming assignments to facilitate testing.

Note: This assignment has been created completely from scratch for the CS3251 course in Spring 2023. Do not attempt to find any solutions online. They do not exist and might lead you to previous similar looking assignments which are significantly different.

2 Introduction

We will be creating a simple Peer-to-Peer (P2P) file transfer system in this assignment. The basic idea in this assignment is that, from the start, each peer (a program called P2PClient) has access to a subset of the chunks of data that makes up a full file-set. Your programs are expected to implement a system that will allow a peer to obtain all the chunks of the file-set from other peers. To help the P2PClient find other P2PClients, a different server program called P2PTracker++ keeps track of which clients have which chunks of the file (We call it P2PTracker++ since its functionality goes beyond that of a BitTorrent tracker which maintains only a list of "live clients"). In short, when a P2PClient starts, it declares the chunks it has to the P2PTracker++. Then, the P2PClient asks the P2PTracker++ for the location (IP address and port number) of the P2PClients who have the other remaining chunks of the file-set and proceed to obtain those chunks from the respective P2PClients. You will create the programs for P2PClient as well as P2PTracker++.

Below, we specify the exact interactions between the P2PClient and P2PTracker++ and those between P2PClients.

3 P2PClient Responsibilities

A P2PClient is invoked using the command line command: p2pclient.py -folder <my-folder-full-path> -transfer_port <transfer-port-num> -name <entity-name>

1. The P2PClient connects to a "well-known" P2PTracker++, which is using "localhost" as the IP and Port #5100 to allow connections from P2PClients.

2. The P2PClient receives a folder name on its command line parameters. The P2PClient reads the file named local_chunks.txt which has the following information, one per line:

   `<chunk_index>,<local_filename>`

   which enlists all the local files already present in the P2PClient's folder. The final line in this file will define the total chunks in the file-set and will have the following format:

   `<num_of_chunks>,LASTCHUNK`

   Note that the `LASTCHUNK` is a text word identifying this line as such. It is NOT a filename. Note that this file will be already in the supplied folder when you run your program on Gradescope. More about that later.

3. The P2PClient computes the hash of each of the file it already has, and then sends this information to the P2PTracker++. When sending this information to the P2PTracker++, the P2PClient is basically saying "if other P2PClients need this chunk, I have it. I am reachable for other P2PClients on a specific IP and port number (one specified in the command line parameters by -transfer_port)." Understanding this statement properly is important for this assignment. Each P2PClient maintains a persistent connection with P2PTracker++. It then also creates a *different* TCP connection for other P2PClients to connect with it. When creating this TCP connection, the port number is taken from the command line argument, and this port number will be communicated with the P2PTracker++ so that other P2PClients know which IP+Port to access for that file chunk. The P2PClient, therefore, sends the following entries to the P2PTracker++:

   `LOCAL_CHUNKS,<chunk_index>,<file_hash>,<IP_address>,<Port_number>`

   The P2PClient can send several such lines of text to the P2PTracker++ and all of them will have the same `IP_address` and `Port_number`. The client does not send any additional `LASTCHUNK` entry.

4. After the P2PClient has updated the P2PTracker++ with all the file chunks it has, it asks for where to obtain other file chunks to the P2PTracker++. To keep things simple, we use a simple request-response format for these queries. The P2PClient sends the following request text:

   `WHERE_CHUNK,<chunk_index>`

   It gets back a response from the P2PTracker++ as:

   `GET_CHUNK_FROM,<chunk_index>,<file_hash>,<IP_address1>,<Port_number1>,`
   `<IP_address2>,<Port_number2>,...`

   The P2PClient is expected to randomly pick a P2PClient to connect to, from the given list.

If P2PTracker++ does not have knowledge about that chunk, it will respond with: `CHUNK_LOCATION_UNKNOWN,<chunk_index>`

5. The client is free to then send another request to P2PTracker++, or first get the file from another P2PClient, or even do the two in parallel. We do not specify this behavior. However, please note that you will find it easiest to not perform the two actions in parallel.

6. The client makes a TCP connection with a peer P2PClient specified in the P2PTracker++'s response. It then requests for a specific file chunk using the command:

   `REQUEST_CHUNK,<chunk_index>`

   The peer P2PClient sends the contents of the file in response. The contents are sent as bytes without any further application layer headers. Note that the contents can arrive in more than one segment over the TCP connection. Therefore, keep reading from the TCP socket until the socket is closed by the peer sending the file.

7. Finally, the P2PClient updates the P2PTracker++ with the new chunk it has, by executing the command:

   `LOCAL_CHUNKS,<chunk_index>,<file_hash>,<IP_address>,<Port_number>`

4 P2PTracker++ Responsibilities

The P2PTracker++ of course responds to clients as mentioned above. The additional responsibilities and caveats are listed below.

1. The P2PTracker++ starts on a "well-known" IP address and port number ("localhost" and 5100).

2. It waits for P2PClients to connect to it. The P2PTracker++ is able to connect with a large number of clients at the same time.

3. The P2PTracker++ accepts commands of type `LOCAL_CHUNKS`, `WHERE_CHUNK`, and responds with `GET_CHUNK_FROM` or `CHUNK_LOCATION_UNKNOWN`. It does not respond to `LOCAL_CHUNKS` commands.

4. The P2PTracker++ maintains an internal check_list which contains all information it obtains from `LOCAL_CHUNKS` commands. It also maintains a chunk_list which contains only verified entries which will be returned to P2PClients when they ask for `WHERE_CHUNK`.

5. The P2PTracker++ moves entries from the check_list to the chunk_list only when two or more P2PClients agree on the hash of the file chunk. So, when the first client connects and sends all its `LOCAL_CHUNKS`, the entries are only recorded in the check_list. When a second P2PClient joins and it has some of the same chunks, P2PTracker++ verifies if the hash sent by both the P2PClients match. If it matches, then the entry is

- what happens if two clients agree on a hash, and one client disagrees
- a new client with a disagreeing hash joins, but this hash agrees with the previously only disgreeing hash
- which chunk should be sent to the chunk_list?
----> first two hashes that match becomes the hash taken into account

moved to the chunk_list and both the P2PClients' information is also kept. If the hash values do not match, then an entry is added to the check_list and this entry will only move to chunk_list after two P2PClients agree on the hash and only the information of the agreeing clients will be present in the chunk_list. Only entries from the chunk_list are communicated with the P2PClient when responding to a `WHERE_CHUNK` request.

5 Gradescope

Gradescope will invoke the P2PClient with `p2pclient.py -folder <my-folder-full-path> -transfer_port <transfer-port-num> -name <entity-name>` on the command line.

The folder will already contain the `local_chunks.txt` file.

Gradescope will split a large file randomly and create chunk files which your P2PClient program will use.

Gradescope will make multiple invocations of your P2PClient and also of your P2PTracker++ to test various aspects of the assignments. This includes supplying different chunks to different clients.

6 Gradescope Grading Strategy - Subject to change

Gradescope tests just the P2PTracker++ for the following properties:

1. ([5 points]) Accepts simultaneous connections from more than 2 clients.
2. ([5 points]) Provides a list of P2PClients where the chunk exists when there are at least two P2PClients who have that chunk.
3. ([5 points]) Ignore clients who have the wrong hash. Such clients should not be present in the chunk_list and should not be returned to a P2PClient when asked for the `WHERE_CHUNK`.
4. ([5 points]) Chunk list does not include a particular chunk for which 2 clients do not exist yet.

Gradescope tests just the P2PClient for the following attributes:

1. ([5 points]) Correctly sends its own checklist to the P2PTracker++.
2. ([5 points]) Correctly asks the P2PTracker++ for all other chunks it needs.
3. ([5 points]) Connects to peer P2PClients without disconnecting from the P2PTracker++.
4. ([10 points]) Performs file transfer from the peer clients.
5. ([5 points]) Asks the peers only for the correct chunks.

6. ([5 points]) Disconnects from peers after file transfer is complete.

Gradescope tests the complete system as follows:

1. ([5 points]) More than one client can be invoked.
2. ([10 points]) All P2PClients make progress and eventually have all chunks.
3. ([10 points]) P2PClients finish in reasonable time.
4. ([10 points]) P2PClients can make and receive connections to and from other P2PClients at the same time.
5. ([5 points]) Small chunk sized files succeed.
6. ([5 points]) Large chunk sized files succeed.

7 Addendum

1. **SHA-1** hashing algorithm should be used to calculate the hash of the file. You can refer to https://www.programiz.com/python-programming/examples/hash-file for the implementation details.
2. **Logging**

   The client and server will each need to write actions they take to a common log named **"logs.log"**. By action, we mean any message sent that starts with `LOCAL_CHUNKS`, `WHERE_CHUNK`, `REQUEST_CHUNK`, `GET_CHUNK_FROM`, `CHUNK_LOCATION_UNKNOWN`. Any time a client or tracker sends one of these messages, the message (along with the client name or P2PTracker prepended to it).

   The messages expected in the log file are:

   (a) `<client_name>,LOCAL_CHUNKS,<chunk_index>,<file_hash>,<IP_address>,<Port_number>`

   (b) `<client_name>,WHERE_CHUNK,<chunk_index>`

   (c) `<client_name>,REQUEST_CHUNK,<chunk_index>,<IP_address>,<Port_number>`

   Note: This logline is different from the message you are sending from the client to the other client. The `<IP_address>,<Port_number>` corresponds to the client to which the client is connecting to get the file chunk.

   For instance, if **client_1** with IP address and port number <**localhost, 7000**> is connecting with **client_2** with IP address and port number <**localhost, 7001**> for chunk index **5**, then the log message should be:

   `client_1,REQUEST_CHUNK,5,localhost,7001`

(d) `P2PTracker,GET_CHUNK_FROM,<chunk_index>,<file_hash>,<IP_address1>, <Port_number1>,<IP_address2>,<Port_number2>,...`

(e) `P2PTracker,CHUNK_LOCATION_UNKNOWN,<chunk_index>`

To be clear, a sample log file may look something like the following:

```
client_1,LOCAL_CHUNKS,2,69b5b00d42765a411ea6e72f39e665728f1b7c03,localhost,8957
client_2,LOCAL_CHUNKS,1,69b5b00d42765a411ea6e72f39e665728f1b7c03,localhost,8958
client_3,LOCAL_CHUNKS,1,69b5b00d42765a411ea6e72f39e665728f1b7c03,localhost,8959
client_4,LOCAL_CHUNKS,2,69b5b00d42765a411ea6e72f39e665728f1b7c03,localhost,8960
P2PTracker,CHUNK_LOCATION_UNKNOWN,1
client_1,WHERE_CHUNK,4
P2PTracker,CHUNK_LOCATION_UNKNOWN,4
P2PTracker,GET_CHUNK_FROM,1,69b5b00d42765a411ea6e72f39e665728f1b7c03,localhost,8959,localhost,8958
client_1,REQUEST_CHUNK,1,localhost,8959
client_1,LOCAL_CHUNKS,1,69b5b00d42765a411ea6e72f39e665728f1b7c03,localhost,8957
client_2,WHERE_CHUNK,2
P2PTracker,GET_CHUNK_FROM,2,69b5b00d42765a411ea6e72f39e665728f1b7c03,localhost,8960,localhost,8957
client_2,REQUEST_CHUNK,2,localhost,8960
client_2,LOCAL_CHUNKS,2,69b5b00d42765a411ea6e72f39e665728f1b7c03,localhost,8958
client_3,WHERE_CHUNK,2
P2PTracker,GET_CHUNK_FROM,2,69b5b00d42765a411ea6e72f39e665728f1b7c03,localhost,8960,localhost,8957
client_3,REQUEST_CHUNK,2,localhost,8960
client_3,LOCAL_CHUNKS,2,69b5b00d42765a411ea6e72f39e665728f1b7c03,localhost,8959
```

Note: Note: This is just the sample snippet of the log file to elaborate on the structure of the messages expected. Do not focus on the values! Also note that there is not one correct log file we are looking for. Everyone's log will look slightly different depending on how they implemented it.

Additional notes regarding logging:

(a) Note that each line follows the format with **NO whitespace**: `entity_name,COMMAND,....`

(b) For the **tracker**, use **P2PTracker** as the entity name. For **clients**, use the value passed in through the command line arg "-**name**".

(c) You **WILL NOT** need to submit the log file with your submission; the autograder will have your submission generate the log file.

(d) You may use any logging library or method of file writing you wish, as long as the messages in **logs.log** fit the format described above. You can refer to the following snippet for python's traditional logging implementation.

```python
import logging
logging.basicConfig(filename="logs.log", format="%(message)s", filemode="a")
logger = logging.getLogger()
logger.setLevel(logging.DEBUG)
logger.info("Your Message")
```

(e) You may log additional messages in the log file if you wish for debugging purposes. As long as the required commands are present in the file, additional lines are completely fine.