Code Progress Document

Commit Hash for Code Check-In

4909fb2283c6de14ed749c7946eccfcf5c40f76f

- **Client can be run via the following command:**
  - `usage: python3 bittorrent_client.py [-h] --torrent=TORRENT [--down_path DOWN_PATH] [--ip_addr IP_ADDR] [--ip_port IP_PORT]`
  - `python3 bittorrent_client.py --torrent=torrents/flatland.torrent`
- **Flatland file will be downloaded to the following path if download path not specified:**
  - `testing/test_write_file`

Overall Progress Update

We have kept up with the deadlines that we listed within our decomposition document. All of the functions within the Tracker, Peer, File System, and Connection Management have been implemented and tested separately, however they have not been integrated and tested together. As of the commit hash stated above, our bittorrent client can retrieve peers from a tracker (both for compact and noncompact versions), connect to the poole peers by sending a handshake and bitfield message, and then download the entire flatland file from the poole peers. However, our client is not able to properly download files other than the flatland file. We are still currently investigating this issue, because if our client is able to download from one peer, it should be able to download from other Bittorrent peers, unless there is a bug in our code.

In addition, we have implemented all of the necessary functions for uploading to other instances of our client, but we have not tested out these parts yet. We also have not completely implemented the top-4 choking strategy yet (we are not keeping track of the top 4 uploaders who are interested in us as of right now), but we have implemented the basic unchoking strategy where we unchoke a peer who is interested in us. In addition, we have also implemented the strict priority policy for our piece selection strategy.

Our plan from here on out is as follows:

- `Dec 6, 2022` : Jonathan and Sadia will continue to implement the File System and Peer components in order to get clients to upload to each other.
- `Dec 7, 2022` : This day will be reserved to fix any bugs within our implementation, connect to non-poole peers, remove any code that is not generic for different torrent files, and ensure that the current implementation and protocol is cohesive before moving onto strategies. Everyone will contribute to this.
- `Dec 8, 2022` : Sadia will finish implementing the Top-4 strategy.

- Dec 9, 2022 : Annie will finish implementing the Optimistic Unchoking strategy for extra credit.
- Dec 10, 2022 : Saar will finish implementing the UDP Tracker for extra credit.
- Dec 11, 2022 : Annie will finish implementing End Game mode and Rarest-First strategies for extra credit.
- Dec 12, 2022 : Everyone will contribute to testing the client to ensure that it's performance is up to the same level as the reference client. The final report will also start being written.
- Dec 13, 2022 : Last minute testing and checking over the final report and client along with final submission.

We plan to implement four extra credit pieces: the UDP-tracker protocol, optimistic unchoking, end-game mode, and rarest-first strategy.

Tracker

Assigned To: Saar Cohen
Progress Summary: All of the functions in the Tracker API in the Decomposition doc are implemented and tested. I did refactor this class a bit, so now there is an extra function called `_get_ip_port_of_tracker` which I will explain later. In addition, I fixed a bug in the compact version of `get_list_of_peers` to iterate through the peers list byte-wise.
API:

- `get_list_of_peers` - was implemented and tested already.
- `_get_len_property` - was implemented and tested already.
- `get_num_pieces` - was implemented and tested already.
- `get_piece_hash_by_idx` - was implemented and tested already, but now it is indeed used in the `check_hash` function in the main file.
- `get_list_of_peers_compact()` - same as `get_list_of_peers()`, but supports the compact mode (the client accepts a compact response).
  - ☑ ~~Implemented~~
  - ☑ ~~Tested independently~~
  - ☑ ~~Tested with Integration~~

  Notes: As listed we had a bug which is now fixed, which was related to the peers section of the dictionary response from the Tracker which is now a robust version.

In addition, I debugged the overall code ın order to make our client succeed in downloading a file (I tested using the flatland file which is now working).
I have added:

- In the `validate_peer_list` function – a section which checks if the number of peers we have in our current list of peers is below a threshold which is set to be 30. If so, it re-connects the Tracker and asks for a new peers list and updates its list.
- A socket timeout when trying to connect to a new peer, so that we can easily detect a socket which we cannot connect to (such as a socket behind a NAT).

- Using Transmission i figured out that even if we successfully connected to a peer, it doesn't mean the handshake is successfully done. Thus, I made the function `send_recv_handshake` to return a boolean – True if the client sent and received back the handshake from the peer, and False otherwise. Hence now we have in our `client_state_list` only connected peers which we handshaked them.
- Fixed a bug in receiving a piece (id==7): calculating the wrong length of piece results our client to mistakenly read the data from peers with an offset (resulting the next piece to contain some data from the last piece, and not beginning with index and begin). This bug is now resolved.
- Added a command line argument for downloaded path to specify a path for downloading file. It's default value is still ./testing/test_write_file as before so it is compatible with older versions.

Peer

Assigned To: Sadia Nourin
Progress Summary: All of the code that was promised was implemented and tested independently (albeit some of the names of the functions did change in the process). Some of the functions were also tested after integration, as our downloading component is successful. There was a bug found within the `convert_list_to_bitfield()` function during manual unit testing that still needs to be fixed. It currently does not crash the client, but it may do so in the future, so this is first on my to-do list to finish.

API:

- ☐ `send_choke_msg(), send_unchoke_msg(), send_interested_msg(), send_not_interested_msg(), send_have_msg(), send_bitfield_msg(), send_request_msg(), send_piece_msg()` - These are all functions that send the peer messages from the Peer class. All of them have been implemented and tested via Poole peers and through inspecting Wireshark messages.
  - ☑ ~~Implemented~~
  - ☑ ~~Tested independently~~
  - ☑ ~~Tested with Integration~~
- ☐ `compare_bitfield_and_request_piece()` - This function compares the current client's bitfield with that of a peer and requests a piece if the peer has a piece that client doesn't have.
  - ☑ ~~Implemented~~
  - ☑ ~~Tested independently~~
  - ☐ Tested with Integration (One of the sub functions have been tested, but a bug was discovered during testing that still needs to be fixed)

  Note: In order to implement this function, three new functions were added in util.py:
  - `find_chunks_of_8()` - This function determines how many bytes are within a bitfield.
    - ☑ ~~Implemented~~

        - ☑ ~~Tested independently~~
        - ☑ ~~Tested with Integration~~
    - `convert_bitfield_to_list()`- This function converts a bitfield into a list of integers
        - ☑ ~~Implemented~~
        - ☑ ~~Tested independently~~
        - ☑ ~~Tested with Integration~~
    - `convert_list_to_bitfield()`- This function converts a list of integers to a bitfield given the number of pieces needed from the torrent file.
        - ☑ ~~Implemented~~
        - ☑ ~~Tested independently~~
        - ☐ Tested with Integration (Not finished with testing due to a bug discovered)
- ☐ `handle_request_piece()` - This function will respond to a request message that it receives from one of its peers. Although this was tested with dummy data, it was not tested with a real request incoming from a peer yet because we have not started uploading tests between two instances of our clients yet.
    - ☑ ~~Implemented~~
    - ☑ ~~Tested independently~~
    - ☐ Tested with Integration

File System

Assigned To: Jonathan Camberos
Progress Summary: All the functionality is currently working except for the `check_hash()` and `reset piece()` which both have been tested independently but not integrated yet. The global variable for storing the blocks of individual pieces was changed from a list of bytes, to a bytearray object which helped with writing to file but led to modification of nearly all functions. I also split up functions, renamed functions, and edited function parameters as our initial picture of how functions were going to work out was very different than in practice.
API:

- ☐ `piece_dictionary` - This global variable will hold the currently received blocks of a piece, for all pieces, in order to recv blocks from the sender. It is a dictionary of byte arrays.
    - ☑ ~~Implemented~~
    - ☑ ~~Tested independently~~
    - ☑ ~~Tested with Integration~~
- ☐ `file` - Global variable we will be writing valid pieces to
    - ☑ ~~Implemented~~
    - ☑ ~~Tested independently~~
    - ☑ ~~Tested with Integration~~

- [ ] `send_from_file()` - This function will be triggered when a peer requests a block of a piece of the file from our client. It will read the requested block for a certain piece and send it to the peer
    - [x] ~~Implemented~~
    - [x] ~~Tested independently~~
    - [ ] Tested with Integration
- [ ] `start_dictionary_and_file()` - This function will initialize the global variable piece_dictionary and file for use
    - [x] ~~Implemented~~
    - [x] ~~Tested independently~~
    - [x] ~~Tested with Integration~~
- [ ] `write_to_file()` - This function will be triggered when the hash of a piece has been verified, in which case we write the piece to our file based on an offset determined by the piece index
    - [x] ~~Implemented~~
    - [x] ~~Tested independently~~
    - [x] ~~Tested with Integration~~
- [ ] `add_bytes_to_dictionary()` - This function will be triggered when a peer sends a ‘piece’ message. The bytes from the message will be copied to the appropriate byte array holder based on the piece index
    - [x] ~~Implemented~~
    - [x] ~~Tested independently~~
    - [x] ~~Tested with Integration~~
- [ ] `check_hash()` - This function will be triggered when a full piece has been received, in which case it will calculate the hash of the received piece and compare it to the hash of the torrent file
    - [x] ~~Implemented~~
    - [x] ~~Tested independently~~
    - [ ] Tested with Integration
- [ ] `check_exists_piece_name()` - This function will check if a certain piece index exists, for edge cases purposes ex: torrent file has 5 pieces but peer sends piece with index of 21 (error)
    - [x] ~~Implemented~~
    - [x] ~~Tested independently~~
    - [x] ~~Tested with Integration~~
- [ ] `reset_piece()` - This function will upon either the bytes length for the piece size has been exceeded or if the hash for a complete piece does not match the torrent hash. In which case we reset the bytearray in the global piece_dictionary and request the piece from scratch
    - [x] ~~Implemented~~
    - [x] ~~Tested independently~~

- [ ] Tested with Integration
- [ ] `check_size()`- This function will for a certain piece index, read from the global piece_dictionary the current length for a byte array and compare it to the expected size (regular piece vs last piece edge case)
  - [x] ~~Implemented~~
  - [x] ~~Tested independently~~
  - [x] ~~Tested with Integration~~

Connection Management

Assigned To: Annie Zhou
Progress Summary: The connection management's core functionality has been implemented. This component has been tested independently on my end through packet-sniffing with WireShark and smaller generic component tests. Therefore, our group can conclusively say that connection works when torrenting the flatland torrent. However, more testing is required for the connection management component when the project is in the next phase of implementing extra-credit features, during the uploading file process, and for other torrent files. In addition, more testing is needed with use cases that are harder to simulate for `validate_peer_list()` (i.e. peer list is below the threshold of peers, dropping a connection, etc…). Aside from the connection management component, I have been helping other team members test their components and have implemented argument parsing functionality for the BitTorrent Client.

API:

- [ ] `recv_handshake_from_initiator()` - This function will handle a peer initiating a connection with the client. The function will also call the `send_bitfield_msg()` function after a successful handshake.
  - [x] ~~Implemented~~
  - [x] ~~Tested independently~~
  - [x] ~~Tested with Integration~~
- [ ] `send_recv_handshake()` - This function will be called to initiate a connection to another peer from the client-side. As a result the client will send a Bittorrent handshake message.
  - [x] ~~Implemented~~
  - [x] ~~Tested independently~~
  - [x] ~~Tested with Integration~~
- [ ] `send_keep_alive_msg()` - This function will periodically send Bittorrent keep-alive messages to peers on the client's peer list to avoid closing the connection.
  - [x] ~~Implemented~~
  - [x] ~~Tested independently~~
  - [x] ~~Tested with Integration~~
- [ ] `validate_peer_list()` - This function will check if the client should be dropped from the select call based on whether a peer has responded in the past two minutes or if it

has closed the connection. If the number of peers is below the threshold, the client will request a new peer list with `get_list_of_peers()` or `get_list_of_peers_compact()`.

- [x] ~~Implemented~~
- [x] ~~Tested independently~~
- [ ] Tested with Integration