CS 118 – Project 1 Part 2

Team:

Soomin Jeong – UID: 304-116-854
Jeffrey Wang – UID: 104-148-158
James Wu – UID: 204-135-990

Overview:
Our overall design involves connecting to the tracker, then calling various functions to set up handshakes and send/receive bitfields from all the peers. This then allows us to follow through with the message flow for each peer we decide to connect to. We have separate functions for each item sent in the message flow.

Setting up handshakes – we set up handshakes by creating a message for the given peer, encoding it, then sending it. When we have received a handshake, we keep track of all the peers and their socket infos in a map. We identify peers using a C++ pair for their IP and port because PEERID is insufficient for distinguishing peers.

Bitfields – We handle bitfields by mapping each peer's info to their respective bitfields. This way we can use the map to find the bitfield of any given peer.

Payloads – All messages are sent and receive in the form of a payload. We have send and receive payloads are modular functions that any message sender/handle can call. Both of these functions require socket and peer information. Send encodes the message payload and sends it. Whenever we receive a payload, we have a function to parse the message received.

Parsing messages – Whenever we parse the message received, we check the 5th bit of its header to determine what kind of message type it is. We move on to sending the next item based on which case of message type we receive. We found that using the 5th bit of the header to check what message type it is was the most efficient way to do this because each message is responsible for keeping track of what type it is, which is a much safer and more robust implementation than using a finite state machine (our original plan).

Handling pieces – We perform several bitshifting operations to check the bitfield of the peer with our own to determine which pieces we have and don't have. We use bitshifting in a reverse way so that there will be no trailing zeros.

Uploading – We upload files to a peer when a peer requests a connection to us. We multithread the downloading and uploading so that we always are listening for a new connection. Once we accept the connection, we parse all the messages, except with the opposite end, replying to a handshake with a handshake and replying to a bitfield with a bitfield.

This crux of this project is to understand how BitTorrent works. Our source code succeeds in passing all criteria that all of the test cases ask for (test downloading, uploading, and error verification). The main issue is having it run in the allotted time (~30 seconds for 23-grading.sh). If we change these time constraints, we pass all of the test cases. The **only** part our program struggles with is multi-threading – we download/upload too slow, but we are sure that everything works. We understand that this is an important part of the assignment.