Program Design

This code consists of 4 classes that are sub classes of the thread class from the python threading module. They are: wait_for_intput, ping_request, ping_response, TCP_listen. Two functions are used to send messages via TCP and UDP, they are named send_UDP_message, and send_TCP_message respectively. Global variables are used to keep track of the peer's ID, two successors and two predecessors.

Waiting for input

The function of this class is wait for any input from the user. Only 2 commands are accepted: "quit", which will kill the peer, and then "request <file name>" which will find the peer which has the file and send a message to the requesting peer.

Ping request

This class controls how often a peer sends ping requests to it's successors, and also if a peer has missed over a number of consecutive ping acknowledgement messages. The latest ping ack message is stored and passed via globals. If over 5 in a row are missed (7 for second successors) the peer will deem it's successor dead and find it's new successors. The reason for having different sequences missed for successor and second successors is because the code for retrieving a new second successor depends on getting a new successor first, hence the small delay. To minimise the possibility of peers receiving multiple ping requests at the same time, a peer will wait 10 seconds to send 1 ping request to it's successor, then it will wait another 10 seconds before it sends a ping request to it's second successor.

Ping response

This class waits for any incoming UDP messages. It then starts a new thread with the target function response_UDP to determine what steps to take based on the message received. Initially I did not open a separate thread for each UDP message received, which would cause severe packets lost since multiple messages would arrive at the same time, but the thread was still processing the previous messages. With multi-threading implemented, packet loss was successfully reduced.

TCP_listen

This class has the same function as ping_response, except it listens for incoming TCP messages. It then opens a separate thread calling the function response_TCP which much like response_UDP determines what steps to take based on the message received.

Messages design

For all messages regardless of TCP or UDP, the first bit of it is always the message type. The subsequent parts of the message are always separated by one space.

Pings

The body of ping messages look like this: "PING peer_id {succ or sec_succ} sequence_number". The peer_id is the id of the sending peer, and the successor part is to let the receiver know if the sender is it's predecessor or second predecessor. Finally the sequence number is to check if a peer has died.

Ping ACKs

Ping ACKs look like this: "PING_ACK peer_id predecessor {succ or sec_succ} sequence_number" The predecessor part is used to let receiver know it is the sender's predecessor. The successor part is to determine if the peer received an ACK from it's successor or second successor.

Request File message

This message looks like this: "REQUEST_FILE file_name original_requester found_flag cycle_flag".

The found flag is used to determine if a file has been found. The cycle flag is for when the hashed filename is smaller than the original requester, meaning the file has to be send through the entire DHT circle to find the correct file.

Response message
The response message looks like this: “RESPONSE peer_id name”. Here peer ID is the ID of the current peer, which is where the file was found. The field name contains the original file name.

Quit message
The quit message looks like this: “QUIT {PRE or PRE_PRE} peer_id successor sec_successor”. The PRE field is used to notify if a peer lost it’s successor or second successor. Peer id contains the ID of the peer that is leaving, and successor and second successor contain the IDs of it’s new successors.

Ungraceful depart messages
These messages look like this: “{NEW_SUCC or NEW_SEC_SUCC} successor SUC_SECC”. They are used to look up the new successors upon discovery of one of it’s successor’s ungraceful departure.

Possible Improvements
Improvements could be done by good documentation. While writing the code, there were numerous times where I could not find a certain function or forgot what a block of code was supposed to do. Secondly, passing values with global variables made the readability a lot worse. This could be avoided by passing the global variables as lists, which are passed by reference in Python. Lastly, some part of the code could be better modularised, using more functions instead of putting all the functions into a giant block of code.

Appendix
For this implementation, Python 3.6.2 was used.
Therefore the setup script had to be changed to the following:
Xterm -hold -title “Peer 1” -e “python3 cdht.py 1 3 4”
Note that the only difference is the added “3” to indicate which version of Python to use.
The link to the demo video is: https://youtu.be/xjWtYCml0Zg