# Assignment Report

## Design Methodology

The circular distributed hash table was designed in python 3.6, and as such the first few design steps were in determining how to create modularity in the program using classes. Approaching the problem, it was clear that the main module of the program will need to use a total of 7 classes:

* A UDP Listener class that contains the information for formulating and sending UDP ping messages to its peers. It is required to send messages and receive messages concurrently
* A TCP Listener class that contains the information for sending and receiving commands such as requesting a file, reacting to quit commands and informing peers of a dead peer.
* A Sender class that is called by the TCP Listener class when a command for requesting a file is sent through the network. This class is primarily used for file transfers and contains all code that is used to serialise the Packet information. The requesting and responding peers will create this class to initiate the file transfer
* A peer relationship class that is responsible for maintaining the peer’s successors and predecessors. This class is also used to periodically update the peers successors and predecessors and fix the relationships in their correct positions when a peer quits or is killed.
* A ping message class that is used by the UDP, TCP and Sender classes to formulate simple strings that contain a keyword that is interpreted as an instruction, followed (usually) by the current peers identity and other important keywords that are used by the listener classes.
* A Packet class which keeps track of the Packets that are to be sent over the network. This class is used to divide the file into a sequence of objects so that they may be sent over the network as serialised pickle packets
* An Acknowledgement class that is used to keep track of the packets received by the requester and create specific acknowledgement messages that is interpreted by the responders Sender class.

Following with this modular design as a base plan, the program was easily divided into parts that could be tested independently of each other. As the program is loaded up, each peer will instantiate all of the above classes and will create a global variable of a struct-like data structure that will maintain the current peers successors.

## Design Construction

The CDHT design was based largely off of Labs 2 and 3. The program was to use the base of Lab 2 in the construction of the UDP Listener class and uses elements of Lab 3 in the TCP Listener classes and is heavily modified to include recurrent ping messaging and connectioned messaging via TCP.

### Ping Message Frequency

The ping messages are sent via UDP every 1 second to each peers successor, and the UDP socket that is used by the peer is flushed every 6 seconds to ensure that if a peer needs to make a new connection, the backlog of messages in the socket are removed so as to make sure that a peer can respond to any ping messages it receives from a different and unexpected peer, which is the case when a peer decides to either quit or is killed.

### Peer Killed Response

The design of this circular distributed hash table does not use ping messages that contain an ACK or sequence number; they simply contain messages of strings that state who is sending the ping. This impacted the design from an early stage, as the ping messages were designed long before considering the case where a peer is killed. To provide a workaround for this, a queue data structure was used to maintain all the incoming ping request and response messages and provides a checking mechanism to determine if the peer is killed. The queue for the ping responses will append all ping responses into it and when it reaches 20 messages, it will pop all of the messages and see whether any of these match the current peers successor; if it is a match then the queue will keep the current successor, if these 20 messages are popped and no ping response is matched to any of the current peers successor, it determines that the successor of the current peer is no longer there, and subsequently determines that the successor of the current peer has been killed.

### Message Format

The design of the messages to be transmitted were relatively simplistic in relation to the UDP and TCP Listener classes. The ping message class is a class that is used by both TCP and UDP Listeners, and its design was of simply wrapping up the current peers identity along with a select number of keywords that can be interpreted by the listeners.

The packets that contain the file data are created using the Packet class, and they created by calling the *divide\_file()* function in the Sender class. This function uses the current MSS and sequence number to create small chunks of the file and creates a Packet object out of the data. Using python, the object is serialised by using Pickle, and is sent as a bytes-like object over the network via UDP. These Packet objects contain a method inside that can be called to write the objects contents into an opened file, and provide the corresponding Acknowledgement objects with their correct sequence numbers.

### File Responsibilities

A separate function is provided that provides the logic behind determining if the current peer is responsible for the requested file. For each file request message that is received, the current peer will format the file name into an understandable form and scan the folder the program is located in; if the file name matches the current peers peer identity in its logic it will return true, and false otherwise.

### Socket Flushing

This was by far the most overlooked aspect of the program design. Initially the sockets buffer size was set so high that upon a peer quitting or dropping, the new successors of the current peer would not respond to a ping message as the buffer size was far larger than the actual ping message size, and so a large backlog of messages would sit in the socket and would only be processed every time a new ping request or response message was received. The workaround for this was to provide a global flusher counter that would increment in tandem with the UDP ping request messenger and would fire the global *flush()* method whenever it reaches a count of 6, which was chosen as the same amount of messages in the queue data structure that determines a peers status.

### Python Version

Version 3.6.5

## Youtube Link

Disclaimer: I do not have access to CSE machines as I live and work far away from campus, therefore I have screen capped the program using alternate software and have run my program through a PuTTY SSH on the CSE Machines.

The link can be found at:

<https://youtu.be/cc94V52OLlg>