ICCS 323 / ICCS 223 Data Communication Networks

Siriporn Auemanarom 5780464
Stefano Cassio 5480954

Break down of the application	
Handle	The main class in the program is Handle. This is the class that deals with the logic of the application and takes care of everything. First, we set a few variables to hold certain variables such as the name of the torrent file, port. This class use to detect whether you are client or server. Then distribute the job of each roles.
get_master_ip	Is used to store the computer's current IP and later when the master has been discovered and M is added to the IP of the master. This is called at towards the end of the handle class where every machine is able to determine which IP is the master IP. Once it enters the main class it will start up 2 threads. - UDPregister( ) - UDPserv( )
UDPregister	used to register the IP Inside the UDPregister thread it gets all the network interface of the current machine and will continue to loop as long as there are more interfaces to go through. - Send a packet that includes (Message, Message Length, 255.255.255.255, Port) - While interface.hasMoreElements() we keep looping over and first check to see if the networkinterface is in loopback mode or if the networkInterface is down. - From here we use the NetworkInterface.getInterfaceAddresses to retrieve all the interfaces of the computer (utun0, awdl0, en0, lo0). We

	also check to see if there are any null addresses and handle it. - We create another variable of type DatagramPacket that now that broadcast's the the subnet. 192.xxx.x.255
UDPserve	used to broadcast to the network and we also used it to add the symbol "M" to the end of the IP
UDPregisterM	used to register master computer so we could identify which computer has the job to distribute the file
Seed	In this function, we keep track of how much the client have downloaded from the server. We also created a progress bar for client to show how much they have downloaded by keeping it in a variable called progress. While the clients are downloading the file, it also a seeder by giving the part that they already finished download to other clients. When the client have finished downloaded all the file, it will turn completely to be a seeder.
SimpleFileClient	We calculate the file size and open the socket to accept the clients.
SimpleFileServer	In this class, we acknowledge all the clients and make connection with clients. We let's the clients know that the connection is successfully accepted by sending a string 'Accepted connection' along with the file size.
To_torrent	Library from github: https://github.com/mpetazzoni/ttorrent

Problems /Bugs?

- Find master -> add M
- Mac vs. window
- Python -> java

Discovery:
One of the biggest problems that we faced was in the discovery part, when we were trying to figure out how to let all the computers know which machine was the master. We were given a suggestion by Nook to add a symbol to the front/back of the master IP so when we scan each IP and find an "M" inside every computer on the network will be able to determine the "Master". This brought other issues because we later found it quite challenging to try and add our "M". We managed to do it by changing the datatype from String to ConcurrentSkipList and adding the "M"+IP.

Windows-Mac
Running the application on windows caused some problems that we did not plan for such as, directories are different, looping over network interface would give unwanted interfaces such as (utun0, awdl0, en0, lo0) and case sensitivity. All these problems took a lot longer than we had expected to fix and was only able to solve these issues through debugging for almost a day. There were also multiple errors "human errors" that were made such as naming a file .txt.txt instead of it being just .txt. This had set us back many many hours.

Python -> Server was too slow
We initially had the idea to distribute the .torrent file through a webserver that was written in python but upon testing we found that there was a couple issues. Firstly it was not able to handle more than 4 users at the same time and as when it was written and tested it was not asynchronous there was also the issue of performance as it would wait for one file to finish downloading before starting the next *get* request.

Some of the problems/functions that we still need to address are

1. Speed, not as fast as we wanted.
2. Writing more robust code as there are still some bugs that we were not able to fix such as handling certain Exceptions such as Connection refused
3. Make it more autonomous as we still need to type in certain commands to make the application run properly.