327 Term Assignment 1 Report

Node is pretty basic. Has an IP which is determined by the host's IP address. Its port is defaulted to 1111 because why not. It has a list of peers which is a list of sockets. It also has a list of records. Each record is a list of two elements, a file's name and a file's last time modified (LTM).

After initializing its attributes, it will start a **listen** thread. This thread just waits for anyone to request a connection and accepts them. After establishing the connection, Node keeps track of the socket created between them into Node's list of peers.

Then Node runs **findOpenPorts**. First, we grab the first three octets of the host's IP address. So if the IP address is 192.168.0.0, it'll get 192.168.0. Then it gets the ARP table and gets all the IP addresses in it that start with those first three octets. I did this because I assumed that those are actual users on my network and because going through all the networks would take too long. After we have our list of IP addresses, we try to establish a connection to it on our given port. If you establish a connection, they are now connected and peers. If not, nothing happens. I was thinking about making this a thread so it'd keep looking for other unconnected devices in a case where they missed each other somehow.

The records are mainly used for the synchronizing . There is a thread that will constantly check Node.py's folder, **checkForSelfUpdates**, seeing if any of them have been recently modified. It compares the file's LTM against the record's LTM of the same file name. If an item inside the folder does not appear in the records, this means that it's a new item. Add this item into the records. All of these updated and new items' file names and file sizes are also logged into an array, which is to be passed to the **send** function. If the new items array is empty, do nothing.

**send** does exactly that. It loops through all of Node's peers and sends all the updated and new files to them. The file size is to make sure all the bytes were sent across. I have a problem with **send**, though. Whenever a client sends their files to the server, the server acts appropriately and downloads them. Although, after updating this item, the server then thinks wants to update this item and runs **send** again (which is actually appropriate to how I coded it.

That part specifically is a problem with the way I did it). But once the server tries to send the "updated" file back to the client, it would send a file of 0 bytes. I think it has something to do with the reading/writing is still processing, when it sends the update. So the file being sent during the update is the 0 bytes one, and then the file finishes writing afterwards.

**receive** is very similar to **send**. It makes sure it has all the bytes and writes it all at once.