Prompt

– How does the client discover other clients on the network?

We iterate through all possible ip addresses on our subnet on our designated port, 9000, and attempt to connect; successful connections are nodes

search for other nodes with one main thread, split off new child threads to make requests to each potential node then return and populate main thread's list of nodes as child thread returns trickle in

– How does the client deal with files of the same name but different contents?

files of same name different contents - make a hashtable with the filename as the key and hashed contents as the value; we can use timestamps to determine which is newest, but merge conflicts must still be handled

different timestamps - older files used as base, newer files' changes of same items replace; something like git for merging

– How does the client determine the order of syncing with regards to the files of other clients?

send out a message from the syncing computer that the file has changed

Implementation

Circular doubly linked list of nodes (Pseudo Chord)

- Each node has its own data as well as 'pointers' which are just ip addresses of its predecessor and successor nodes
- Any time a node changes data in a file, it will update its dht and inform its successor of the changes
- Nodes need to constantly poll their predecessor for a connection (a successful connection indicates there is an update)
- Use circular doubly linked list of nodes to maintain network - to keep the dht synchronized; use the dht to send the files when there is an update

Joining the network

- Client node joins a network by using FindIpAddresses and getNodes(). It then chooses a random server node from this list (probably the 0th element since this should be one of the fastest)
- It connects to this server node on the port used for connecting
- It sends a message that it wants to join the network
- The server node sends a message to its successor that a new node wants to join the network
- The client node becomes the server node's new successor and the server node becomes the client nodes new predecessor
- The server nodes' old successor becomes the client nodes' new successor and the client node becomes the server node's old successor's predecessor

Connecting

- Every node always has a server thread running
- When node has a file update, the node's server thread connects to the next node's client thread; it forms a connection, which is passed to a new RequestHandler thread that sends the server's dht; the client compares the dht to its own and the hashed file contents to determine which files it wants from the server; it then requests the files and receives them from requesthandler

DHT

- Keys are fileNames

    1. Content has timestamp and a list of the ip addresses with that file
    2. We can also have it so that content is a list of pairs of timestamps and ip addresses
- For: 1., once a node changes a file, it searches its local hashtable for the filename, updates the timestamp to current time, broadcasts to all ip addresses in the list that a change occurred and sends to them the filename, timestamp, and file contents (or just changes), it sends just the updated hashmap to all other nodes in the network (or we send hashmaps when any two nodes communicate)
- For: 2. (probably less efficient), once node changes a file, it searches the hashtable for the fileName, updates the timestamp for its ip address in the list, and sends the filename, timestamp, and file contents to all ip addresses with old timestamps; each receiving node will have to update its file contents and hashmap with the new timestamp then send their updated hashmap to all nodes in the network
- To resolve merge conflicts in DHTs or files, we can:
    - Simply overwrite the older content with the newer
    - Iterate through all content, if any data does not match, highlight differences, inform user, give the file a null timestamp indicating to resolve merge conflict
    - We can keep track of all changes and if there are no conflicting changes, we can keep all changes, conflicts can be resolved as above