

Term Project – File Synchronization

Circular doubly linked list of nodes (Pseudo Chord)

- Each node has 'pointers' which are just ip addresses of its predecessor and successor nodes
- Any time a node changes data in a file, the dht that is generated every X seconds 'updates' so the next node will know there is an update
- Nodes need to constantly poll their predecessor for a connection

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)
 - If there are no other nodes on our port, the client effectively becomes a network of 1
- It connects to this server node
- 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 (see appendix figure 1)

Connecting

- Every node always has a server thread running
- Every X seconds, the node's server thread connects to the next node's client thread; it forms a connection, which is passed to a new RequestHandler 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
 - content is the hashed contents
- when two nodes connect, the client and server make new hashtables of their files
- the client compares its hashtable to the servers and requests any files that the server has that the client does not or which have different contents (see appendix figure 2)
 - we default to taking the server's copy of files if they are different

Improvements with more time

The network

- we could debug the network so that when the previous and next nodes of a node are the same node, there is no weird socket error
 - perhaps due to us not having main start the server and client threads correctly – that is, they are desync'd
- conversely we could implement Chord the proper way with the finger table and math and that would improve performance ($\log(n)$ for sending messages, updating, and storage)
- we could implement proper waiting to make some threads block until a connection is made instead of relying on `Thread.sleep()` to force waiting
- in `findIpAddresses`, instead of spawning new threads to ping each possible ip address in our subnet, we could make something to properly query an arp table to find nodes for our network
- we would implement a way to handle a node leaving the network and leaving abruptly
- we would implement a menu to join/leave the network and to sync files to handle errors from having searching for/joining a network and syncing files in the same 'thread'

DHT

- we would add in a timestamp and cache a hashed copy of each file to manage merge conflicts better
- on hash table conflict, we could look at each file conflict and compare each file's hashed contents to the cached copies and if they exist, we simply take the newest one and update the old copy, otherwise we cache this new copy and tell the user they need to manage the merge
 - we could organize the hashed value so that we could determine which file versions are parents of which children and use this information to highlight which parts of the file are different to users (like git)

Output (works in reverse too, but depends on which order we start the processes)

client side
port 9000 to establish network

```
run:
9000
Listening for a client
Trying
.0
finally
Devices on subnet 192.168.254:
192.168.254.19
.0 192.168.254.19
Listening for a server
9019
Listening for a client
previous: .0
previous: 192.168.254.19
next: 192.168.254.18
next: 192.168.254.18
Key: asdf.docx
Value: 1619909916029
Key: test.txt
Value: 1619909917082
Key: Untitled2.png
Value: 1619909918410
Key: excel.xlsx
Value: 1619909352968
Key: New OpenDocument Spreadsheet.ods
Value: 1619909353978
Key: test - Shortcut.lnk
Value: 1619909355014
Key: test2(0).txt
Value: 1619909356015
gimme files
asdf.docx
test.txt
Untitled2.png
excel.xlsx
New OpenDocument Spreadsheet.ods
test - Shortcut.lnk
test2(0).txt
Connection Set: /192.168.254.19:49710
C:\cecs327\asdf.docx
12570
File Received
asdf.docx
still want: 7
Received a connection
asdf.docx
excel.xlsx
New OpenDocument Spreadsheet.ods
test - Shortcut.lnk
test.txt
test2(0).txt
Untitled2.png
next: 192.168.254.19
file: asdf.docx
file: excel.xlsx
file: New OpenDocument Spreadsheet.ods
file: test - Shortcut.lnk
file: test.txt
file: test2(0).txt
file: Untitled2.png
hash size: 7
C:\cecs327\test.txt
35
File Received
test.txt
still want: 6
C:\cecs327\Untitled2.png
1120228
File Received
Untitled2.png
still want: 5
C:\cecs327\excel.xlsx
13949
File Received
excel.xlsx
still want: 4
C:\cecs327\New OpenDocument Spreadsheet.ods
6598
File Received
New OpenDocument Spreadsheet.ods
still want: 3
C:\cecs327\test - Shortcut.lnk
907
File Received
test - Shortcut.lnk
still want: 2
C:\cecs327\test2(0).txt
17
File Received
test2(0).txt
still want: 1
192.168.254.19 192.168.254.19
Listening for a server
```

server side
using port 9000 to establish network

```
run:
9000
Listening for a client
Trying
.0
finally
Devices on subnet 192.168.254:
192.168.254.18
.0 192.168.254.18
9018
Listening for a server
Listening for a client
Received a connection
asdf.docx
excel.xlsx
New OpenDocument Spreadsheet.ods
test - Shortcut.lnk
test.txt
test2(0).txt
Untitled2.png
previous: 192.168.254.18
next: 192.168.254.18
file: asdf.docx
file: excel.xlsx
file: New OpenDocument Spreadsheet.ods
file: test - Shortcut.lnk
file: test.txt
file: test2(0).txt
file: Untitled2.png
hash size: 7
Waiting for client
C:\cecs327\asdf.docx
File Sent
previous: 192.168.254.19
previous: 192.168.254.18
next: 192.168.254.18
next: 192.168.254.18
C:\cecs327\test.txt
File Sent
C:\cecs327\Untitled2.png
File Sent
C:\cecs327\excel.xlsx
File Sent
C:\cecs327\New OpenDocument Spreadsheet.ods
File Sent
C:\cecs327\test - Shortcut.lnk
File Sent
C:\cecs327\test2(0).txt
File Sent
Waiting for client
C:\cecs327\asdf.docx
File Sent
C:\cecs327\test.txt
File Sent
C:\cecs327\Untitled2.png
File Sent
C:\cecs327\excel.xlsx
File Sent
C:\cecs327\New OpenDocument Spreadsheet.ods
File Sent
C:\cecs327\test - Shortcut.lnk
File Sent
C:\cecs327\test2(0).txt
File Sent
192.168.254.18 192.168.254.18
Listening for a server
```

hashtable of filename keys and hashed file content values

message to server that it wants these files

received this file

listing files in shared directory

listing files in hashtable

sending files

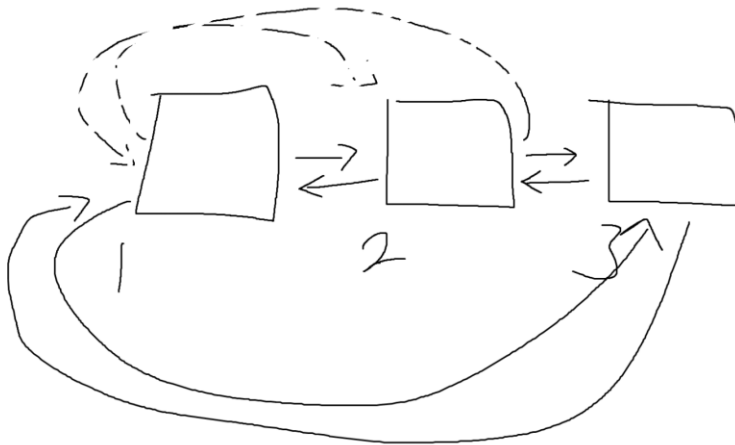
previous and next ip's are set correctly looping to listen for updates

previous and next ip's are correct looping for updates

Appendix

Figure 1

How nodes join the network



when no machines in a lan have our specific port open, there is no network
 in this case, a node, 1 will create its own network and have "" as its next and previous 'pointers'

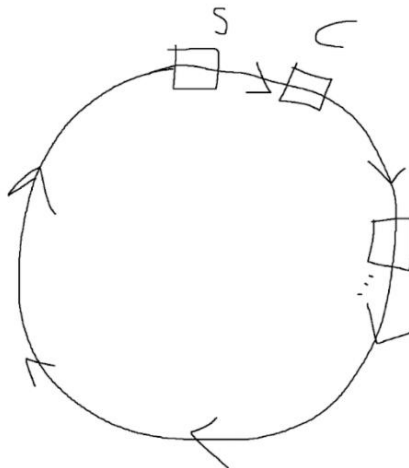
when there is one machine, node 1, in a network and node 2 tries to join, node 2 will set its previous and next to equal node 1's ip address and node 1 will set its previous and next as node 2's

when the network is 2 or more, node 3, the last node will always set its previous to the node it contacts (node 2) and its next to the contacted node's next; node 2 will keep its previous, set its next as node 3 and tell

its old next (node 1) that its new previous is node 3

Figure 2

synchronization



Client is always listening for server trying to connect
 since connection requests timeout after ~10 seconds, we need to have client open its port again every 10 seconds (pulsing updates)

when Server has an update, they try to connect to Client

we assume they connect

dht keys are fileNames, values are hashed fileContent

Server recreates a fresh dht of its local files and sends it to Client

Client recreates a fresh dht of its local files

Client then compares the two dhts.

if the two dhts are equal, do nothing

else

make a temp dht

copy client dht into temp dht

loop over server dht

if a server fileName matches one in temp, compare the hashed fileContents: if equal, nothing, else take the Server one (alternatives are take one with newer timestamp, append file contents)

Directory: CECS327

Server: test.txt
 asdf.txt(has: "a")
 bit.bmp
 x.xls

Client: asdf.txt (contains: "b")
 x.xls

Client: test.txt
 asdf.txt (has: "a")
 bit.bmp
 x.xls

Server DHT:
 key value
 test.txt 000000000
 asdf.txt 000000001
 bit.bmp 000012789
 x.xls 000128938

Client₁ DHT:
 key value
 asdf.txt 000000002
 x.xls 000128938

Client₂ DHT:
 key value
 test.txt 000000000
 asdf.txt 000000002
 bit.bmp 000012789
 x.xls 000128938