CDHT ASSIGNMENT

CAMERON MURRAY Z3417671

https://youtu.be/PcruAEHSMMw

INTRODUCTION

The CDHT assignment was attempted using Java on Intellij IDEA in a file cdht.java. The java project has 4 parts, a main function and 3 methods that are created upon compilation. These all utilize their own threads. The file can be compiled and run with a shell setup script **setup.sh**

TEST & DEVLOPMENT

Developed on: CSE Lab Computers JRE 8
Windows 7 using JRE 8

Tested on: CSE Lab Computers JRE 8 (VLAB).

Running the program:

After augmenting the peer list in the setup script to create an acceptable Distributed Hash Table Network scaffolding, one can run process with the following command (provided they are in the cur. Directory is)

```
$ ./setup.sh
```

If needed, joining DHTN as a single instance can be achieved as follows e.g, where the integer arguments are the peerID, the first successor and the second successor, respectively:

```
$ java cdht 4 5 8
```

If the file is packaged, you must provide a suitable CLASSPATH identifier instead.

PROG. DESIGN

The chosen language was Java due to its relative simplicity compared to C. Both Java & Python has very easily implemented socket and protocol implementation, however due to my unfamiliarity with Python, Java was selected

Program Design

The project is relatively straightforward; as soon as the program is instantiated, the java implementation initially traverses a couple initialization responsibilities to determine the accessibility of a user defined port, maximum threading capacity and argument scaffolding to govern if a peer can be successfully injected into the network

The Java implementation, with a main() method, Ping Method, UDP Server Method and TCP Server Method runs all 3 methods in their own respective infinitely looped threads. The duty of the main method in the project was to define the static var's that hold the node descriptions. If an input not found, then an Error/Exit warning is then forwarded to the printscreen
The 3 three other threads can be abstracted as follows

Ping Thread	instantiate/initialise then P I N G's 2 succPeers, deals with un-graceful departure
UDP Thread	Waits and responds to TCP Threading P I N G's, i.e. waits for responses
	- Thread infinitely loops to broadcast pings to the succ's of existing peers - Watches UDP prt:5 0 0 0 0 + P E E RID for any available pingRequests
TCP Thread	Waits for either a file requisition, successful file transmission response, a succPeer requisition or a (graceful) peerQuit
	- Thread infinitely loops to handle TCP comms. It continuously ----- - Watches TCP prt:5 0 0 0 0 + P E E RID for; fileTransfer requests or churn requests.

MESSAGE DESIGN

UDP Transport:

Message	**Description**
PING <seq>	Send ping with sequence number <sequence>.
REPLY <seq>	Reply to ping with incoming sequence number <sequence>.

TCP Transport:

Message	**Description**
REQUEST<aaa><send><req>	Peer <sender> is forwarding a request message for file with name <wxyz> to be sent to peer <requester>.
FILE<wxyz><id>	Inform recipient that peer <id> has file <wxyz>.
QUIT<1st><2nd><Identification>	Peer <id> with successors or predecessors <first> and <second> is leaving the network.
QUIT_RECEIVED	Acknowledge a QUIT message.
LIST	Request recipient to reply with its list of successors.
PEERS <first> <second>	Sender has successors <first> and <second>.

IMPROVMENTS

- Being able to efficiently end threads in a way that doesn't hurt the underlying schematics of the project. This could fix any delays that occur after graceful peer departure
- Could use a more friendly GUI for displaying the responses/requests, as well as any extra functionalities to make traversing the programs functions easier.
- A output stream for each time a peer joins the DHT, could be written to and archived in a log
- Utilising real TCP file transmission would be a way to increase the working functionality of the system
- Extending a real time peer joining capability by contacting any node.
- There could be other improvements, like 2-directional traffic, File requisitions to be sent to 2nd successor, more ungraceful peer dying situations (multiple dying peers), or more efficient ungraceful departure handling (e.g. contacting successor and predecessor).

CONSIDERATIONS

The Ping Thread iterates every X seconds that is user defined in the static variables list. This is guided by a defined TIMER that initializes the pinging after 1s. The chosen TIMER used for demonstration purposes was 10000ms, however choosing a greater value around the 25000 mark could utilize a compromise between reserving system resources (from excessive pings) and checking succ status. A departing node can inform not only its pred, but also its succ to update the state of the pred in a more efficient manner. This is because a in the program, a node that has undergone a graceful departure has been made to inform both the succ and the pred, even though only the pred are needed for the specification. If only the pred was informed, the nodes could take up till the following ping to augment their preds.

The timeout before a ping is considered lost in a real space could be approximated by the transmission control protocol timeout-interval: EstimatedRTT+4*DevRTT, as ERTT & DRTT are the moving means. For the demonstration, this interval was user defined at 1 second. The reason this can be set relatively low is due to the small RTT of different ports branching from a single node.

Prior to a succ being regarded as having left ungracefully (left without warning), the user defined amount of unreturned ping responses is set to 4. If the number of unreturned pings is set too great, there would be unused resources whilst waiting for initiative to be taken to restructure the CDHT. The max difference in seqNums is 20000000000000000. This could be more efficient by utilizing a smaller sequence range since we are only looking for 4 missing acks to determine a peer has died.