z5161678

COMP33331 Assignment Report

Briefing

The Circular DHT programming assignment for COMP3331 was completed.

Python3 was used to write the application and all of the code is contained in a file (**cdht.py**)

The Youtube Link to view the video is: https://youtu.be/c1VioAIV0VI

Testing and Development Environment

Developed on: CSE Lab Computers using Debian GNU/Linux 6.0.10 running Python 3.7.2

Tested on: CSE Lab Computers using Debian GNU/Linux 6.0.10 running Python 3.7.2

Running the program: The program can be run using a format similar to the one given below. One can add and remove lines based on the number of peers they want.

```
xterm -hold -title "Peer 1" -e "python3 cdht.py 1 3 4 300 0.3" &
xterm -hold -title "Peer 3" -e "python3 cdht.py  3 4 5 300 0.3" &
xterm -hold -title "Peer 4" -e "python3 cdht.py  4 5 8 300 0.3" &
xterm -hold -title "Peer 5" -e "python3 cdht.py  5 8 10 300 0.3" &
xterm -hold -title "Peer 8" -e "python3 cdht.py  8 10 12 300 0.3" &
xterm -hold -title "Peer 10" -e "python3 cdht.py  10 12 15 300 0.3" &
xterm -hold -title "Peer 12" -e "python3 cdht.py  12 15 1 300 0.3" &
xterm -hold -title "Peer 15" -e "python3 cdht.py  15 1 3 300 0.3" &
```

Program Design

Language Choice: Python was used as the development language because I find it easier to read and understand as compared to the Java and C. Furthermore, the UDP/TCP protocols are very straightforward and easy to use in the language

Display: The application prints messages to the terminal (**xterm**). At the beginning of the program the user is prompted by an 'Enter command: ' message that allows them to enter a command based on the assignment specifications. If a wrong message is typed, then the screen states 'Invalid command' to let the user know.

Design: Upon starting the program, for each instance of a peer, there are 4 threads that are run. These are:

1. A thread that sends UDP request messages to the first successive peer

2. A thread that sends UDP request messaged to the second successive peer

3. A thread that listens to all UDP messages on port *50000+i* and returns a request or response message based on what the peer received.
    a. This thread is also responsible for determining whether the successor peers are alive or not. If a response message is not heard for two 'rounds' of messages, then the peer that is considered to be dead and is removed from the DHT. The other peers then update their successor and predecessor peers.

4. A thread that listens to all TCP messages on port *50000+i* and returns response messages based on what the peer received.

a. This thread solely is responsible for file related messages. If a file related operation such as 'send this file' or 'receive this file' is called, then it starts a thread that deals with sending and receiving file related data

File Manager Thread: This thread sends and receives UDP messages for file transmission. It implements a stop-and-wait message format and also writes relevant messages into the *responding* and *requesting* log files.

There is also an infinitely looping function in main that listens to the commands of the users and executes commands based on it.

All threads are *daemons* so that they are killed once the program ends or is terminated

There are various helper functions that help achieve the tasks described above.

Message Design

UDP Transmission Messages:

Message	Purpose	Printed statement
REQUEST:portNum:msgNum:tag	A request message to successive peers. *portNum*: Port number sent from *msgNum*: Numerical count of message sent *tag*: Tag for whether it's sent to first or second successive peer	A ping request message was received from Peer *x*
REPONSE:msgNum	A response message with the appropriate *msgNum*	Ping response message was received from Peer *x*

UDP File Messages

FILE:seqNum:ackNum:data *Note: The header(FILE:seqNum:ackNum) was modified to be 25 characters	Message containing file data. *seqNum*: Sequence Number *ackNum*: Acknowledgement Number data: File data to be sent Sent by peer sending file data
ACK:ackNum	Acknowledgement message for file data received *ackNum*: Appropriate acknowledgement number Sent by peer receiving file data

TCP File Messages

F_REQUEST:requestor:KEEP:filename	Sent to next peer to tell them that file is stored with them *requestor: peer that requested file*
F_REQUEST:requestor:CHECK:filename	Tells successive peer to check whether they have the file
F_REQUEST:requestor:FOUND:filename	Tells current peer they have the file

Future Improvements

- Screen could have options for scroll functionality. A log output for each peer could also be created
- Curses and colour highlighting could be employed to make printed messages more readable
- Add file transfer over TCP network and allow peers to join the CDHT network

Troubleshooting and Bugs

- **Program fails to start because ports are in use** – Run *ps* and use *kill -9 [pid]* where *pid* is the processor id for the python instance. Make sure that ports don't need to be used
- After a peer gracefully leaves, it accepts an extra UDP request message.
- Sometimes it takes a round (10s) for a peer to display request and response messages just after receiving a file.

Limitations

- **Peer churning in 3 peer network** – If there are 3 peers and one leaves, then the remaining peers have themselves as their second successor.
- **Fails to deal with more than 1 peer churn in 1 round trip** – If more than one peer leaves within 1 second then the program fails to detect that both peers have left till after 2 round trips and then won't be able to detect their next successors.

References and Acknowledgements

All code used in the program was mine.

The report format was adopted based on Mohammad Ghasembeigi's report on GitHub:
https://github.com/mohammadg/Circular-DHT-Network/blob/master/doc/report.pdf