COMP3331
CDHT Programming Assignment

Mohammad Ghasembeigi

BRIEFING

The extended version of the CDHT programming assignment was completed.
Python was used to write the application and all code is contained in a single file (**cdht_ex.py**).
A modified version of the setup file is provided in **setup.sh**.

TESTING AND DEVELOPMENT ENVIRONMENT

Developed on: Xubuntu 14.04 using Python 2.7.6.

Tested on: Xubuntu 14.04 using Python 2.7.6.
CSE Lab Computers using Python 2.7.3.

How to run application:
Open **setup.sh** in your favourite text editor, add or remove peers to create a valid circular DHT network scenario and then run the bash script by opening a terminal in the current directory and entering:

```
    $ chmod 755 ./setup.sh
    $ ./setup.sh
```

Alternatively, you can enter the network as an individual peer like so:

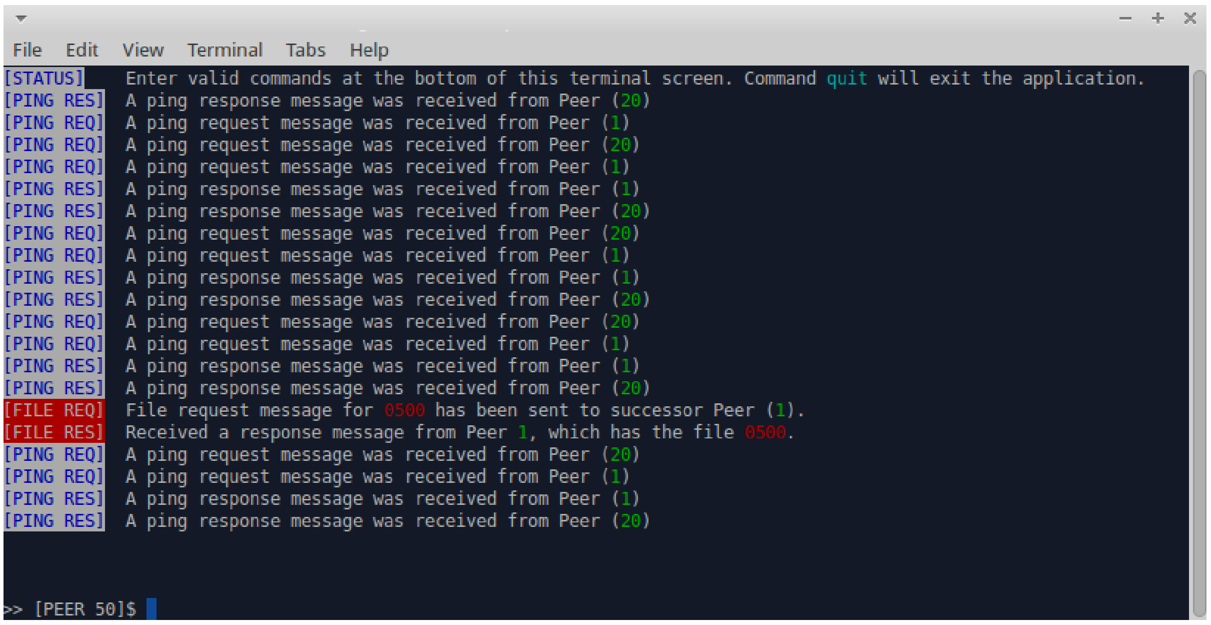
```
    $ python ./cdht_ex.py [peer identifier] [succ1 id] [succ2 id]
```

Note: ensure **python** points to a python 2.7.x build and not a python 3.x build.

PROGRAM DESIGN

Python was used as the development language due to my familiarity with it and because UDP/TCP protocols are very straightforward and easy to use in the language.

Curses Screen
The application itself uses **curses** to display all text as opposed to the terminal (**xterm**). The application prints messages to a curses screen that has 2 columns, one for printing control messages and one for printing information messages. Curses was used in this application as it can format output nicely and allowed me to keep the standard input prompt (that end users use to enter text into stdin) at the bottom of the screen. The standard input prompt allows you to enter commands into the system at any time. You are able to delete characters if you wish using the backspace key. The messages on the screen will automatically 'wrap upwards' once they reach the last few lines of the terminal. The input prompt displays the peer identifier attached to the current screen at a glance.



```
[STATUS]    Enter valid commands at the bottom of this terminal screen. Command quit will exit the application.
[PING RES]  A ping response message was received from Peer (20)
[PING REQ]  A ping request message was received from Peer (1)
[PING REQ]  A ping request message was received from Peer (20)
[PING REQ]  A ping request message was received from Peer (1)
[PING RES]  A ping response message was received from Peer (1)
[PING RES]  A ping response message was received from Peer (20)
[PING REQ]  A ping request message was received from Peer (20)
[PING REQ]  A ping request message was received from Peer (1)
[PING RES]  A ping response message was received from Peer (1)
[PING RES]  A ping response message was received from Peer (20)
[PING REQ]  A ping request message was received from Peer (20)
[PING REQ]  A ping request message was received from Peer (1)
[PING RES]  A ping response message was received from Peer (1)
[PING RES]  A ping response message was received from Peer (20)
[FILE REQ]  File request message for 0500 has been sent to successor Peer (1).
[FILE RES]  Received a response message from Peer 1, which has the file 0500.
[PING REQ]  A ping request message was received from Peer (20)
[PING REQ]  A ping request message was received from Peer (1)
[PING RES]  A ping response message was received from Peer (1)
[PING RES]  A ping response message was received from Peer (20)


>> [PEER 50]$
```

Figure 1: Curses screen for Peer 50 in CDHT network

Colour Highlighting

To make the output easily readable, control signals, file hashes and peer identifiers were highlighted with different colours. This made the screen more visually appealing as well as more readable for clients who are managing multiple screens at once.

Program Design

The program itself if quite simple but elegant. Upon starting an instance of the application, some initialization tasks are performed (such as argument checking and port availability checking) before the peer is entered into the CDHT network.

From this point onwards, three infinitely looping functions are run.
The **main** function infinitely loops to capture user input from the input prompt. If input is found, it is executed and if the command is invalid, an error message is printed to the screen.

Another infinitely looping function (**pingMonitor**) is run via a thread. This thread continuously sends out pings to the current peers successors as well as monitors UDP port $50000 + i$ for incoming ping requests.

Another infinitely looping function (**TCPMonitor**) is run via a thread. This thread handles all TCP communication messages. It continuously monitors TCP port $50000 + i$ for incoming file transfer and peer churn quit/query requests.

MESSAGE DESIGN

There are 6 control messages used when printing information:

[STATUS]	General messages about the application, network and input commands.
[PING REQ]	Informs client that a ping request was received.
[PING RES]	Informs client that a response to a ping request was received.
[FILE REQ]	Informs client that a file request message is being sent or forwarded in CDHT network.
[FILE RES]	Informs client that a requested file is available and is being received /sent.
[PEER CRN]	Informs client that a successor peer has either gracefully or ungracefully exited the CDHT network and thus a peer churn is in progress.

There are 4 CDHT commands that are valid:

quit	Will gracefully exit CDHT network.
request <filehash>	Requests provided file hash using CDHT network.
ping on	Enables ping messages.
ping off	Disables ping messages. **Note:** Pings will continue to be sent/received. This simply suppresses the **PING REQ/PING RES** messages that normally appear on the screen.

The message format for ping is **4 bytes** long and is detailed below:

Ping Type	1 byte	**0x00** for ping request, **0x01** for ping response
Sender Peer ID	1 byte	The sender peer ID between 0 – 255.
Sequence Number	2 bytes	The sequence number attached to this ping (0-65535)

The message format for file transfer is **4 bytes** long and is detailed below:

Message Type	1 byte	**0x00** for file request, **0x01** for file request (forwarded), **0x02** for file response
Sender Peer ID	1 byte	The sender peer ID between 0 – 255.
File Hash	2 bytes	The requested file hash (numeral form). ie. filehash “2012” is sent as the 2-byte integer 2012 (for TCP performance reasons)

The message format for a peer churn is **6 bytes** long and is detailed below:

Message Type	1 byte	**0x04** for peer graceful quit message, **0x05** for query request, **0x06** for query response
Sender Peer ID	1 byte	The sender peer ID between 0 – 255.
Successor 1 Peer ID	2 bytes	The identifier for successor 1. (supports negative values for special peer states) Not used in query request mode (**0x05**)
Successor 2 Peer ID	2 bytes	The identifier for successor 2. (supports negative values for special peer states) Not used in query request mode (**0x05**)

FUTURE IMPROVEMENTS

- There is a small delay before threads are killed when a peer gracefully leaves. A future improvement would be to kill threads in a faster (but still thread safe) manner.
- The screen could be improved to add auto resize and scroll functionality. This would make using the program easier. A log output for each instance could also be created.
- Obviously, the next step functionality wise would be to add actual file transfer (over TCP) and peer joining capabilities to the CDHT network.

BUGS AND TROUBLESHOOTING

Screen output contains many weird/irrelevant characters
This is a curses issue which may occur if you run the application after an application that modifies the terminal characters is run. This shouldn't happen out of debugging but if it does, simply exit the application and run the command **clear** before starting it again. Alternatively, resize the xterm window slightly to force screen to refresh.

Program fails to start because port(s) are already in use
Ensure the normal ports that will be used are not in use. Also ensure no python instance is running (ie a previous instance of the application you halted may still be clogging up a socket).
If this is the case, run **ps** and then **kill -9 [pid]** where pid is the processor id of the python instance.

LIMITATIONS

Program fails to churn peer correctly or auto-detect new successors/predecessors
Please allow 1 ping round trip (~1 second) to complete before you make a peer quit. The application can only handle 1 quit at a time. Exiting more than one instance at a time will most likely lead to unexpected results. You should also wait for 1 ping round trip to complete when first starting an instance (before entering any commands).

Peer churning in 3 peer network
If 3 peers remain in a network and a peer leaves (gracefully or ungracefully) then each of the two remaining peers will have themselves as their second successor. I decided to not create a special case for this scenario and left it "as is" because there is no nice way to handle it. The same limitation is present when churning in a 2 peer network (leaving 1 final peer in the network).

REFERENCES AND ACKNOWLEDGEMENTS

A modified version of the curses screen template (functions **input**, **prompt** and loop in **main**) was created by *James Mills* as a reply to a stack overflow question I asked:
http://stackoverflow.com/a/30259422/1800854

The **terminate_thread** was made by Johan Dahlin and is available here:
http://stackoverflow.com/a/15274929/1800854

The enum definition type was crafted by the Stack Overflow community and is available here:
http://stackoverflow.com/a/1695250/1800854

All other work is my own.