Assignment Report

COMP9331 18S1

P2P DHT Simulation

Jiongtian Li, z5099187

Date: 20/May/2018

1. Assignment brief description

Python version: 3.6
Demo link: https://youtu.be/f0zWb_lsO-U
This assignment aims to simulate part of peer-to-peer(P2P) protocol Circular DHT network and implementation of socket programing for both UDP and TCP transport protocols, including following functions:

- Initialization and Ping successors
- Requesting a file
- Peer departure
- Kill a peer

2. Initialization and Ping successors

This program cdht.py is designed to take 3 arguments including peer number, first successor and second successor. And once the program runs, it will initialize UDP and TCP sockets which bind to the localhost address and (5000+ peer number) port. Each peer has two successors, and it will continuously ping its two successors every 3 seconds to check if its successors live or not. As a result, each peer will also receive 'ping' request from its predecessor and it will send back a 'pong' response. When each peer receive ping or receive 'pong' response, it will print "A ping request message was received from Peer X" and "A ping response message was received from Peer X." Both sending ping request and response ping request are implemented in UDP protocol, therefore, it may occur lost due to the unreliability of UDP. Sending and replying ping request are implemented in two different function: ping(ping_port) and response_ping(). These two functions will be called in separated threads every 3 seconds.

3. Requesting a file

Once user type 'request X' which X is the 4 digits number file name, the requesting peer will send file request to its successors until it found the location of the file, and the responding peer will send TCP message directly back to the requesting peer. The file is stored in the peer which is the closest to the hash, which is the remainder when file number divided by 256. For example, the hash of file 2561 is 1 and it will be stored in peer 1 if exist. This function will be triggered by the detection of user input, and it will send TCP file request message which including the file number, hash number, requesting peer number and sending from port number, to its first successor immediately. When the peer receives the file request message, it will try to determine if it has the file or not by checking if the hash number is bigger than this TCP request sending port peer(predecessor) and smaller than or equal to itself peer number. If it is, it will send a TCP response message directly to the requesting port. Otherwise, it will forward this TCP message to its first successor. However, different situation need to be handled if the current peer is the first node, which has two larger predecessors peer number. If the peer is first node, it needs to check if the hash number is bigger than its predecessor or hash number is smaller than itself peer number. If one of the situations satisfy, the file is stored in the first node. Both file request message and file response message are implemented in TCP protocol.

4. Peer departure

As the peer will continuously ping its two successors every 3 seconds, and we need to also consider the unreliability of UDP, if the successors continue no response ping messages to its predecessor 30 times, the peer will determine its successor is dead. As a result, peer will take about 90 seconds (ping every 3 second times 30 times lost) to detect its successor is no longer live. Before setting the interval to every 3 seconds and 30 times lost, I tried to use every 10 second interval and 7 times lost, and it did work fine on my local machine. However, I need to increase the frequencies and lost time when I use SSH connect to the UNSW lab machine. And finally, every 3 seconds and 30 times lost are good intervals to determine peer departure in lab machine.

5. Kill a peer

This function is very similar with those of file request by detecting user input ‘quit’ and send/receive TCP messages. So once the program detects user input ‘quit’, it will send its first successor and second successor to its two predecessor which will continuously updated when peer receives ping requests. As a result, the peers receive TCP message from leaving peer then it will update its successors, at the end send back another TCP message ‘received quit’ to the leaving peer. Once the leaving peer receive that response, it will quit the program.

6. Conclusion

In this program, I implement several functions and call those functions in different thread so multiple functions can run at the same time. Besides, one UDP connection is good enough to handle send/receive/response ping requests. However, TCP protocol is quite complex compare to those of UDP, so I implement one TCP server connection which binds to the 50000 + peer number port to listen the TCP messages. Whenever, it need to response or send TCP messages, it will build a separate TCP client connection to send TCP messages which include the information of sending port number, therefore the receiving port will know which port is sending the TCP messages, because the TCP client doesn’t bind to any specific port.