COMP9331 Computer Networks and Applications Assignment

Yun Lu
Z5139037

I. Testing and Development Environment

I developed this system using Python 3.6, and tested on CSE Lab Computers using Python 2.7.3.

II. Briefing of the System

This assignment is to develop a part of a peer-to-peer (P2P) protocol Circular DHT network, to stimulate some implements of CDHT.

This system is mainly composed up with 4 classes, including UDPServer, UDPClient, TCPServer and TCPClient. All the classes are created from inheriting from threading.Thread. Thus they are able to run separately in four threads simultaneously.

UDPServer holds the server code to receive request message and send response message by UDP protocol. The server itself is a multithreaded server, which allows it run separately in a thread. It has inner functions for handling incoming messages.

UDPClient holds the client code to send request message and receive response message from server by UDP protocol, and check the content of messages. For example, if a peer send a ping message with a sequence number inside to another peer's UDPServer, and when it get the response message, it will read sequence bit to check if it is the same with the one it sent to the server, to decide if a peer is alive or dead, allowing the peer to update its list of successors.

TCPServer holds the server code that handles the receiving and sending TCP messages. The role of this class is to take an incoming connection, read the data and check the content to trigger the next movement.

TCPClient is designed to receive the sting input from the host, and send a new TCP message to a peer's TCPServer. It can call a function called TCPClient_request(), aimed to establish a TCP connection and send message through TCP protocol. Other class can call the function as well, so that they do not need to stop threads to send TCP messages.

III. How the system work

1. Initialisation

This system start with the global function to get parameters from user input. There are 3 parameters, which are ***peer_id, peer_successor_1, peer_successor_2***.

Then every peer start their treads of UDPServer, UDPClient, TCPServer and TCPClient, and start pinging their successors every 10 seconds.

So far the initialisation is done.

2. Running

After initialisation, the system is running. Every 10 seconds, a peer would send UDP request messages to its successors. Once it receive a request message, it would response immediately.

During the running, users can do some operations on any peers. A peer would send messages to other if the user execute some operations like quit and request file.

3. Shutting down

A peer would stop running if user input ‘quit’ or kill it (by pressing Ctrl + C in the xterm).

In the ‘quit’ operation, the string input would trigger TCP message sent to its predecessors and a is_quit flag to turn all of the peer’s threading off.

If a peer is stopped by killing, its predecessors would check if they can receive response message from it. If they sent more than 5 request message but with no response, they would reckon the peer dead and change their successors.

IV. Message Design

UDP design:

1. Ping request:
   a string type data contains ***sequence number*, *peer_id***, with a space between them.

2. Ping response:
   2 string type data contains ***last received sequence number*, *peer_id***, with a space between them.

TCP design:

1. Request file:
   If the file is not found, the message would be a string type data contains ***filename, peer_id, the original peer***, with spaces among them.
   If the file is found, the message send to the original peer would be a string type data contains the word **‘file’, *filename, peer_id*.**

2. A peer quit gracefully
   a string contains the number ***‘0’, peer_id, peer_successor_1, peer_successor_2*.**
   ‘0’ is a flag to tell other peer this peer is leaving, and *peer_successor_1, peer_successor_2* is its successor so that its predecessor can ping them.

3. A peer leave ungracefully
   When a peer is found dead, a string contains ***peer_id*** of which is leaving will be sent to its predecessor’s the other successor.
   Then it receive a string type message like ***peer_successor_1, peer_successor_2***, containing its two new successors.

V. Design Tradeoffs

Originally I would like to use list containing integers to transfer the message. However as I recall what we learnt from the lecture, TCP messages are all transferred and encoded as strings. Then I changed my data type to strings. It makes my data longer, but by this way, I can write more detailed content in one packet. Then the server can easily recognise which message is for what operation.

Another tradeoff is the time interval of each Ping. If the interval is too long, the peer would hardly found the leaving of others. If it is too short, the system would bring heavy load to the computer. So I decide 10 seconds, between two times of ping. It is reasonable to detect a peer’s leaving and allow operation time.

VI. Youtube Link for Demo

https://youtu.be/WgEKfAxDpoo