COMP9331 Assignment

Kuan-Chun Hwang z5175539

YouTube Demo: https://youtu.be/rjvxU5fpuuQ

Python version: 3.5 was used to design the program

Program Design

The steps I took to implement the program was to first identify which type of messages are required by TCP and UDP and then created a sender and receiver for TCP and UDP that would handle the relevant communication required by each transport protocol. Below describes the role of each functions in the program.

TCP and UDP servers

The global variables, 'udp' and 'tcp', are used to hold instances of tcp and udp servers. These are set up when the code first runs and are bound to the corresponding port of the peers.

TCPreceiver

This function will handle the reception of TCP messages such as quit, get successor, successor/predecessor update and file request/respond. Its role is to take incoming data and process it accordingly.

TCPsender

The function sendTCP is used to send TCP messages. It takes the destination port and the message to be sent as inputs. It has its own instance so that the receiver socket can continue receiving data while this instance is working.

UDPreceiver

This function will handle the reception of UDP messages, which are ping messages. It will handle ping request and ping responses appropriately while sorting out the ping sequence number tracking and predecessors.

UDPsender

The function sendUDP is used to send UDP messages, which is mainly ping related. Similar to TCP sender, it will take message and destination port as inputs.

PingCheck

The function pingCheck is used to check the ping sequences tracker to determine if any peer has peer has ungracefully quit the system by being killed by the user. It will check for number of consecutive unanswered pings and run the ungracefulQuit function if required.

UngracefulQuit

This function runs when a peer that was killed is detected. It opens a new TCP instance and sends the get successor message to the appropriate successor that was determined by PingCheck.

LocateFile

This function is used to determine if the file is located at a peer. Since file is located based on the closest successor of its hash value, there are three cases that need to be considered. The first is the normal case where the hash is between the predecessor and peer ID (including peer ID). The second is when a peer is the first peer such that predecessor is larger than the peer ID (highest peer ID) and the hash is larger than the predecessor. The last case is similar the second case that when the peer's predecessor has the highest peer ID, and the hash value is smaller than the predecessor and the peer's ID.

Working of the System

Initialisation

The setup phase of the system which sets up the TCP and UDP receivers port. After setup, the system will start listening for command line inputs and also start processing ping request and responses. The setup, taking command line input, reading from TCP and UDP receiver, sending ping request and performing ping sequence check to detect if any peer is no longer alive is all performed by the main function.

Ping successors
Each peer will start pinging by sending ping request messages to its two successors to check if they are still alive. This part is done by the main function and a ping is sent out every 10 seconds. This choice of time will be explained in a later section. The sending of ping request messages is done by the UDP sender described above. These ping request messages are then processed by the receiving peer's UDP receiver which will react with ping response messages using UDP.

Requesting a file
To give file requests, user will input 'request XXXX' into the command line, where XXXX is the filename. Any filename that is not got 4 characters and are not a number will be rejected so format for filename less than 1000 should use the format described in the assignment specification (e.g. 0159). The hash of the file is evaluated and then is locatefile function described in the previous section is used to determine if a peer has the file. If a peer has a file, it sends file respond message to the requesting peer using TCP. If it doesn't, it will forward the file request message to its first successor using TCP as well.

Peer departure
Peer departure is detected when user input 'quit' into command line of a pair. This input will trigger the main function to send successor and predecessor update messages to appropriate peers using TCP. After all update has finished, the confirmed quit message will then be sent over TCP to tell the leaving peer that it can safely exit.

Kill a peer
When a peer is killed, ping messages are used to detect this. Ping check detects if there are 3 or more consecutive pings unanswered by keeping track and looking at the ping sequences. Once detected, it waits for 5 seconds before declaring that a peer is no longer alive. In summary, the three parameters are: number of consecutive ping request messages that a peer fails to respond to before that node is declared to be no longer alive = 3, frequency of ping messages = every 10s and timeout = 5s. When a dead peer is detected, the appropriate action will be made by the predecessors of the dead peer to update its successor.

Message Design
UDP message design
UDP is used for ping request and responses and their message format are:

Ping request: pReq,[seqNo],[sender],[predecessor number]

- pReq to indicate that it is a ping request, seqNo for sequence number, sender to indicate who the request is from and predecessor number is used to indicate the current relationship between the receiver and the sender. Predecessor number is either 1 or 2 to indicate if the receiving peer is first successor or second successor.

Ping response: pRes,[seqNo],[responder]

- pRes to indicate ping response and seqNo for sequence number. This sequence number will be the sequence number of the ping request for confirmation purposes. Responder will be the peer ID of the peer that is responding to the ping request.

The UDP messages type encode, 'pRes' and 'pReq', are used because they are reasonably short and has a clear meaning as well. Therefore, the size of the message sent is kept minimum while still being readable by human. Shorter message type encoding can be used, but it may become more difficult to understand. The size ping request message is longer than the ping response by 1 byte as it requires an additional field of predecessor number which is a number encoded as byte string. The UDP sender sends both ping request and response by encoding the messages above as byte strings.

TCP message design
TCP is used to send the following messages:

File request: fReq,[filename],[requester]

- fReq to indicate that it is a file request message. Filename for the file being requested. Requester for the peer ID of the requester. These messages are send when the user type in the appropriate command line input, which triggers the locatefile function to send file requests to its successors until it reaches a peer that has the file. Since filename will always have 4 digits and the requester ID range from 0 to 255, the maximum size of file request message is 11 bytes.

File response: fRes,[filename],[responder]

- fRes to indicate file response. The file response message follows a similar style to file request so it also has a maximum size of 11 byte.

Get Successor: gSuc

- Upon receiving gSuc as the message, the TCP receiver will send the first successor number encoded as byte string as a reply. gSuc is sent when an ungraceful quit peer is detected.

Successor/Predecessor update: sUPD/pUDP,[leaving peer][new successor1/predecessor1][new successor2/predecessor2]

- These messages are used to tell a peer to update its successor or predecessor. Maximum 13 bytes.

Graceful quit: q

- Upon graceful quit command is entered by the user and after successors and predecessors are appropriately updated, the main function will send ‘q’ using TCP to confirm that the peer can safely exit. This message is kept as short as possible by using only 1 character.

From above, the successor/predecessor update messages have the largest possible size. The size of the messages can be reduced further by reducing the message type encoding (e.g. fReq, fRes…). However, it may become difficult to understand if a reduced message type encoding was used and may make development harder.

Design choices and trade-offs

Ping intervals: 10s is used here because it is an appropriate time interval that make the command line interface still useable such that it is not over flooded with ping response and request messages. However, a trade-off is that it made a killed peer detection a lot longer (30s).

Number of lost packet and timeout before death declaration: The program allows 3 consecutive unanswered ping and has a timeout of 5 seconds. When the third consecutive ping is unanswered, the program will wait 5 seconds before death declaration. I felt like three was a good number because the ping interval was set to 10 seconds. Therefore, only a total of 30s is required to detect killed peers. If ping interval was decreased, this number can be increased. Since ping is set to 10s, timeout of 5s was reasonable.

Fail Cases

In general, the program will work in all cases. However, there are cases when command line input will overflow. This occurs when a file request is made between the period after a peer is ungracefully killed and before it is declared dead by other peers. If a file request is made during this period, and the killed peer has the file or is required to pass on the file request, the command line will be over flooded with file forward request messages as the system is trying to find the killed peer which hasn’t being declared dead yet. This issue can be solved by reducing the ping interval and the trade-off of this was discussed before.

When two successors of a peer is ungracefully killed one after another, before they are declared dead because the predecessor peer uses one of the successor normally to find its new successor and if both of its successors are dead, it won’t be able to find new successors and the system will break.

Possible improvements and extensions

One of the possible improvements will be to modify the set up so that it becomes a circular DHT with short cuts. This will require an additional variable to be setup in the program to keep track of its short cut peer and the setup script will need to be modified to accommodate this change. This can reduce the number of messages required for file requests. However, additional ping will also need to be sent to the short cut peer to maintain the short cut path.

Future extension could be that the program will be sending actual files. When a peer has a file that satisfies a file request, it can send the file through TCP for the requester to use.