CDHT Report

Z5141448 Ruofei HUANG

Runtime Envriomeent

Python3 (subversion 3.6.7)

Demo Video Link

https://youtu.be/Pjk3ots-fWI

Implement Steps

1. Simple Ping Client and Server
2. Central state management
3. Command line arguements handling
4. Ping Client and Ping Server with loss rate
5. File sender and modify Ping Server to receive file
    a. Design protocal and handle the value
    b. Passing buffer
    c. Stop and wait
    d. Log files
6. Main Controller and Input Worker (handle input after the programe is running)
    a. Testing callbacks
    b. Transfer the state management into main controller
7. Peer management
    a. Predecessor successor
    b. Whether file is exist in local
    c. Loss of predecessor or successor
8. TCP Information handling
    a. Peer Loss
    b. File request and response
    c. Callback hell
9. Final testing & Troubleshoot

Design Choice

Ping Detail

1. Ping Interval: 15 sec
2. We are not using ping number to decide:

We use live probability, when the live probability < 0.001, we will decide the peer is lost. Then we kill the ping thread and make a callback to main thread. Each ping drop will have the live_probability *= loss_rate. Each time a ping is answer, we make sure the peer is there, hence the live rate is 1. If the peer is uncertain of it's healthy, we immediately re-ping the peer.

Protocal Design

Each message passed within TCP or UDP has two part, one is header and another is body. Header will be the enum of information type such as:

```
15  # Const for request type
16  PING = 1
17  RECV_PING= 2
18  FILE = 3
19  FILE_ACK = 4
20
```

and event value. Each value will take up 8 bytes and send as an big-endian unsigned integer.
The body in file transfer it will be bytes array. In TCP message, we will have a 8 bytes message in some required information. This 8 bytes message will also be an unsigned integer.

Predecessor and Peer Logic

We only record the the latest two predecessor ping request, so if there is only one predecessor and have two ping request, the client will only think it has one predecessor. As for peer logic, we will sort then with the peer is less than current client, we will increment or decrement by 256 base on predecessor sort or successor sort. Then the main can get them by index. At that time, we will reverse the process to get the correct id.

File Transfer Design

We use the ack by bytes instead of ack by sequence number. Because I want to improve the sending performance by having some window size. But we required to use stop and wait. It's same logic, but we don't have sequence number to ack.

Overall Structure

1. cdht.py
    a. Main controller
        i. All the callback is write in here
        ii. Manage the worker threads
        iii. Manage the information store
    b. Input worker

2. event.py
    a. Logging logic
3. headers.py
    a. Protocal Logic
    b. Helper function to translate between byte and integer
4. info.py
    a. TCP Server & Worker
        i. Pre-process the informations
        ii. Dispatch the information to correct
    b. TCP Client
        i. Wrap the information to message
        ii. Send to correct server
        iii. Callback to correct function if there's any reply message
5. peer.py
    a. Predecessor and successor management
    b. File location management
6. ping.py
    a. Ping Client
        i. Decide whether peer is alive
        ii. Ping it
    b. Ping Server
        i. Receive and reply ping
        ii. Receive file and reply ack
            1. Record ack
            2. Write to file
    c. File Transfer Client
        i. Get file by Max Segment Size
        ii. Transfer or retransmit if ack is loss
7. store.py
    a. Signle instance of information store