z5127440
YIFAN ZHAO
22 May 2018

COMP9331 Assignment Report

Computer Networks and Applications Session 1, 2018

In order to implement a part of the peer-to-peer (P2P) protocol Circular DHT, I used **Python 3.6.4** to achieve 5 features: Initialization, Ping Successors, Requesting a file, Peer departure and Kill a peer.

YouTube link: **https://youtu.be/yMxseQOxj2E**

Step 1: Initialization

To initialize the P2P systems ， TCP and UDP should be set up .So I set up a TCP client and a UDP client to send data to servers, and a TCP server and a UDP server to handle these data.

My data message formats:

```
str = "{0},{1},{2}".format(tag, peer, message)
```

When I need to use the data ,I decode it to a list L .

```
# data encode
def data_encode(self,tag, peer, message):
    str = "{0},{1},{2}".format(tag, peer, message)
    return str.encode(encoding='UTF-8',errors='ignore')

# data decode
def data_decode(self,data):
    s = data.decode(encoding='UTF-8',errors='ignore')
    L = s.split(",")
    return L
```

Step 2: Ping successors

Firstly, a ping_loop function is used to send "ping" data to other peers automatically.

Then , handle data which have a tag as"ping".In this moment , set this peer's preorder peer .And send a new data which contains the times this peer have had received from it's preorder peer.

```
def ping_loop(self,peer):
    times = 1
    while self.isRun:
        for i in self.successor:
            data = self.data_encode("ping", peer, times)
            self.udp_client(data, 50000 + i)
        times = times + 1
        time.sleep(12)
```

The interval between pings is 12s.

Step 3: Requesting a file

```
def find_nearest(self,a, b):
    a = int(a)
    b = int(b)
    min_value = abs(a - b)
    if min_value >= 255 - min_value:
        return 255 - min_value
    else:
        return min_value

# help fuction to find file in or not in this peer
def find_file(self,number, peer):
    if number == peer:
        return True
    else:
        value = self.find_nearest(number, peer)
        value_next1 = self.find_nearest(number, self.successor[0])
        value_last1 = self.find_nearest(number, self.preorder[0])
        if value < value_next1 and value <= value_last1:
            return True
        else:
            return False
```

Handle data which have a tag as"request" and find the file location by two help function .

Step 4: Peer departure

If a peer leave from system , it will informs its preorder peer and successors by TCP message. Then it's successors and it's preorder peers should update.

```
if query == "quit":
    print("Peer {0} will depart from the network.".format(peer_message))
    s_list = message.split("|")
    if int(peer_message) == self.successor[0]:
        self.successor[0] = int(s_list[0])
        self.successor[1] = int(s_list[1])
    else:
        self.successor[1] = int(s_list[0])
    print("My first successor is now peer {0}.".format(self.successor[0]))
    print("My second successor is now peer {0}.".format(self.successor[1]))
    return
```

Step 5: Kill a peer

From step 2, we send the times this peer have had received from it's preorder peer. Compare these two response times . If abs(response_time1-response_time2) is more than 4 , that means a peer is lost in systems. Find the peer and update others.

```
isLost = False
lostPeer = -1
if int(peer_message) == self.successor[0] and int(message) > self.response_from_pre1:
    self.response_from_pre1 = int(message)
    if self.response_from_pre1 - self.response_from_pre2 >= 4 and self.response_from_pre2 > 0:
        isLost = True
        lostPeer = 1
        self.response_from_pre2 = self.response_from_pre1
```

The number of lost packets before assuming a peer is killed is 4.

Conclusion:

In this assignment , although I make several mistakes , I have learn lots of knowledge about TCP , UDP and P2P systems. In my program, there are some points to improve ,such as find nearest file locations and how to find a lost peer.I will study hard in future.Thanks a lot!