# COMP3331/9331 Computer Networks and Applications

## Assignment Report for Term 1, 2020

Nan Wang Z5238059

## ✧ Description of how my system works

Firstly, I initialed all system arguments by a class, and used the conditional structure to distinguish argument 'init' and 'join'.

```
    def initDHT(self, peer, first_successor, second_successor,
ping_interval):
    self.peer = peer
    self.first_successor = first_successor
    self.second_successor = second_successor
    self.ping_interval = ping_interval
    self.port = self.peer + 1200
```

**Step 2 - Ping Successors:**

In order to realize the UDPping function, I used a UDPclient and a UDPserver, also considering that this is a p2p network, which means it needs to support multiple peers to work at the same time, so I used two threads:

```
Thred1 = threading.Thread(target=p2p.UDPServer, args=(host,))
Thred2 = threading.Thread(target=p2p.UDPClient, args=(host,))
```

In UDPclient, if a peer is alive, it will send the message to its two successors and also send a request message containing specific content to the UDPserver. Once the UDPserver received the message, it will return the response information to the UDPclient. Finally, close their connections.

**Step 3 - Peer Joining:**

Because implementation of this function needs using TCP, there is a new thread:

```
    Thred3 = threading.Thread(target=p2p.TCPserver, args=(host,))
```

In this step, firstly it needs a function to learn which peers are the new peer's successors. I created a TCP connection of know peer to send a message including join peer to the TCPserver, and then find the correct location of join peer. If the current peer didn't find the correct location, which means the join peer should not join between it and its first successor, it will forward this message to the next peer until finding the right location.

```
def forward(self,host,message,dest):
    #print(message, '------', dest)
    serverName = host
    serverPort = dest + 12000
    clientSocket = socket(AF_INET, SOCK_STREAM)
    clientSocket.connect((serverName, serverPort))
    clientSocket.send(message)
    clientSocket.close()
```

Once finding the location of join peer in TCPserver, firstly return to the first function as mentioned above to get the two successors of the join peer and secondly change the current peer's successors(it should be the first the predecessor of the join peer ) and then send the message to the peer which is the second predecessor of the join peer to change its second successor to the join peer.
Finally, the join peer can be initialized and completely join the p2p network.

**Step 4 - Peer Departure (Graceful):**

After finished the function of Join Peer, the remaining functions are simple because we already have the TCPserver and forward function. However, it should be noted that "Quit" will be entered on xterm terminal, so I used a CommandInput function to get the new arguments:

```
def CommandInput(self,host):
    while True:
        command=sys.stdin.readline()
        if command.startswith("Quit"):
            self.isAlive = False
            message = pickle.dumps(
                {"type": "Quit", "Quit_Peer": self.peer, "FS":
self.first_successor, "SC": self.second_successor})
            #print("first_pre", self.first_predecessor)
            #print("second_pre", self.second_predecessor)
            self.forward(host, message, self.first_predecessor)
            self.forward(host, message, self.second_predecessor)
```

Similarly, using the forward function to send a message including the quit peer to TCPserver.

**Step 5 - Peer Departure (Abrupt):**

Using Ping that implementation in UDPclient to make sure if the peer is not alive anymore. I choose the method that every message sent to UDPserver form UDPclient including count in Step 2, so if the ping was sent multiple times to a peer and the count number is more than 2, it means this peer is no longer alive. And then sending the request successor message by TCPserver to get the successors of the dead peer (notice: there are two conditions, first is the dead peer is the current peer's first successor and the second one is the second successor). Finally using forward function to make the request successor message to the response successor message in TCPserver for updating the successors of two predecessors of the dead peer.

**Step 6 - Data Insertion:**

As same as Step 4,
Firstly get arguments and then using TCPserver.

```
    if command.startswith("Store"):
    command=command.split()
    #print(command)
    filename=int(command[1])
    des_peer=self.Hashmod(filename)
    if des_peer==self.peer:
```

```
        print (f"Store {filename} request accepted")
    else:
        print(f"Store {filename} request forwarded to my successor")
        message=pickle.dumps({"type":"Store_file",
"des_peer":des_peer,"file_name":filename})
        self.forward(host,message,self.first_successor)
```

**Step 7 - Data Retrieval:**

After getting the argument form xterm, forwarding to the first successor by TCP to find the location which store the file. When a peer find the file stored there, it should send the file found response message to the requesting peer by forwarding function and use a new function to transfer the file by TCP

```
def file_transfer(self,host,peer,filename,dest):
    tcp_Port=dest+12000
    tcp_server_socket=socket(AF_INET,SOCK_STREAM)
    tcp_server_socket.connect((host,tcp_Port))
    file=open(str(filename)+".pdf","rb")
    data=file.read()
    file.close()
    message=pickle.dumps({"type":"File_transfer","filename":filena
me,"data":data})
    tcp_server_socket.sendto(message,(host,tcp_Port))
    print(f"The file has been sent")
    tcp_server_socket.close()
```

However, one thing to note is that when downloading the file, there may be a case where the completion cannot be accepted, so I used this to make sure pickle.loads() is always correct.

```
    while True:
    connectionSocket, addr=serverSocket.accept()
    data=b""
    while True:
        packet=connectionSocket.recv(2048)
        if not packet: break
        data+=packet
        command = pickle.loads(data)
```

## ✧ Possible improvements part:

There should be stricter limits on the loss of pings, not just through the number of transmissions.

When transferring files, maybe the relevant parameters should be recorded in the log

## ✧ Borrowed code part:

https://stackoverflow.com/questions/44637809/python-3-6-socket-pickle-data-was-truncated