# Report for Project P2P

## 1. Software & Environment

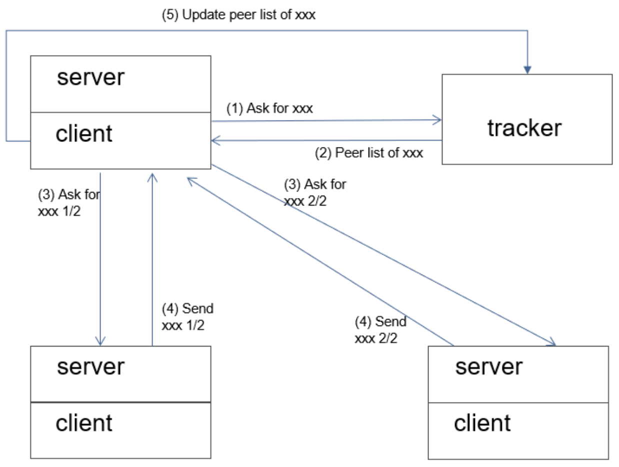
This project is written in Python 3.6 using IDE PyCharm. The libraries being used includes NumPy, socket, os, threading.

The test environment was: Windows 10, 2.3GHz*4.

## 2. Test scenarios and test results

We conducted our experiment in two scenarios: ① communicate within one computer and ② communicate between multiple computers.

In both cases, we had one tracker and 3 clients running, for each client there is a folder containing all the files that it's ready to serve, and all the files it downloaded from others are directly saved to this folder. The process is as shown in the following figure:



We successfully conducted the experiment in both scenarios, client1 gets the file it asked from the other two peers and calls the tracker to update the peer list.

The test result is shown in a separate file (testResult.pdf) and the videos.

## 3. Code Design

### Tracker:

Tracker is implemented in **tracker.py** file, we stored the addresses of clients in the network and peer list (list of clients who have the requested file) in the beginning field of it:

```
addr1 = ('127.0.0.1', 1000)
addr2 = ('127.0.0.1', 1100)
addr3 = ('127.0.0.1', 1200)
dic = {"net.txt": [addr1, addr2], "network.txt": [addr1, addr3], "seeds.txt": [addr2, addr3]}
```

Since tracker can only receive two types of messages: ① query for peer list and ② update the peer list, we designed a header field “query_update” for the messages and defined a method to split the data in tracker.py:

```
def parse(data):
    query_update = data[0]  # 0: this message is a query, 1: this message is an update
    message = data[1]
    return query_update, message
```

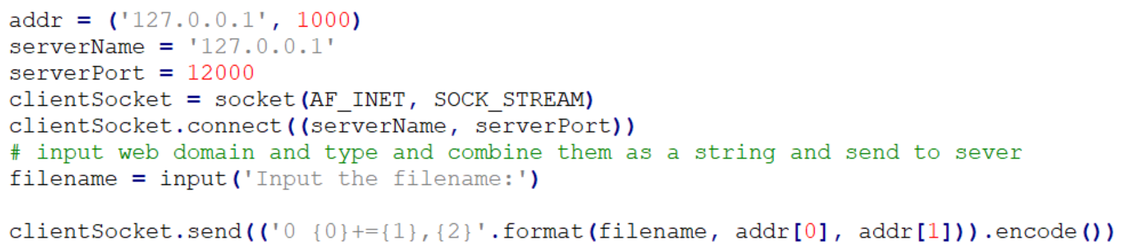
We implemented tracker with threading so it can handle multiple request simultaneously. In each process, tracker handle requests as the following code defines, it's worth mentioning that if a client queries for a file that it already has according to the peer list, tracker remove it from the dictionary and server as if the client doesn’t have this file; and if a query ask for some file that is not recorded, tracker returns a “404”:

```
modifiedMessage = self.conn.recv(1024).decode()
query_update, message = parse(modifiedMessage.split())
filename, addr = message.split('+=')
name, port = addr.split(',')
try:
    if (name, int(port)) in dic[filename]:
        dic[filename].remove((name, int(port)))
    if int(query_update) == 0:
        a = dic[filename]
        resp = ""
        for i in range(len(a)):
            resp += str(a[i][0])+" "+str(a[i][1])+" "
        self.conn.send(resp.encode())
    elif int(query_update) == 1:
        dic[filename].append((name, int(port)))
        print('Update!')
        print(dic)
except KeyError:
    self.conn.send('404'.encode())
self.conn.close()
```

**Client:**

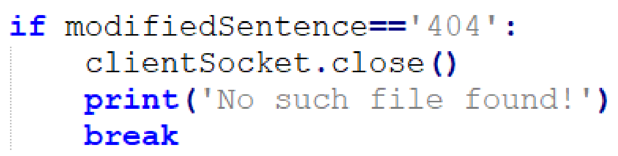
One client is implemented in two files, one to send queries and updates and receive the responses (node1.py, node2.py, node3.py), another to sever files in the folder (node1server.py, node2server.py, node2server.py).

In node<i>.py, in the beginning field of the file, the address of its server and the address of tracker are stored:



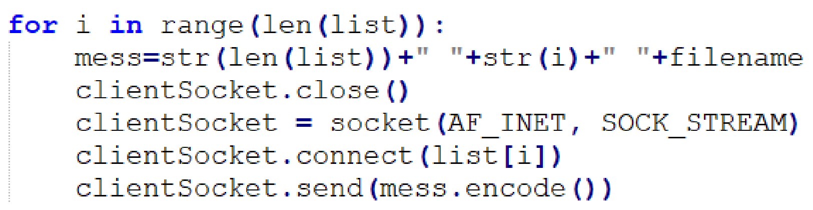
```
addr = ('127.0.0.1', 1000)
serverName = '127.0.0.1'
serverPort = 12000
clientSocket = socket(AF_INET, SOCK_STREAM)
clientSocket.connect((serverName, serverPort))
# input web domain and type and combine them as a string and send to sever
filename = input('Input the filename:')

clientSocket.send(('0 {0}+={1},{2}'.format(filename, addr[0], addr[1])).encode())
```

Each time when we run the node<i>.py, we need to input the file name and it'll first send a query to tracker, there may be two kinds of response: “404” or the peer list of this file:



```
if modifiedSentence=='404':
    clientSocket.close()
    print('No such file found!')
    break
```

If a peer list is returned, the client then send request to all the addresses in the list and ask for part of the file:



```
for i in range(len(list)):
    mess=str(len(list))+" "+str(i)+" "+filename
    clientSocket.close()
    clientSocket = socket(AF_INET, SOCK_STREAM)
    clientSocket.connect(list[i])
    clientSocket.send(mess.encode())
```

When it has received all the parts of the file, it combines them and write it into a new file with the same name as the name it requested, and put it into the folder of its. Finally, it sends the update message to the tracker, and that’s one whole process:

```
f = open(os.getcwd()+"\\node1\\"+filename, 'w')
f.write(ans.decode())
f.close()
clientSocket.close()
clientSocket = socket(AF_INET, SOCK_STREAM)
clientSocket.connect((serverName, serverPort))
clientSocket.send(('1 {0}+={1},{2}'.format(filename, addr[0], addr[1])).encode())
```

In node<i>server.py, the logic is quite simple, the server receives request for the file and send back the specified part of the file, then wait for next request:

```
serverPort = 1000
serverSocket = socket(AF_INET, SOCK_STREAM)
serverSocket.bind(('127.0.0.1', serverPort))
serverSocket.listen(10)
print('The server is ready to receive')
while 1:
    connectionSocket, addr = serverSocket.accept()
    modifiedMessage = connectionSocket.recv(1024).decode().split()
    totalnum = int(modifiedMessage[0])
    neednum = int(modifiedMessage[1])
    filename = modifiedMessage[2]
    str = open(os.getcwd()+"\\node1\\"+filename, 'rb').read()
    str = str[int(len(str)/totalnum*neednum):int(len(str)/totalnum*(neednum+1))]
    connectionSocket.send(str)
    print('Uploading {0}({1}/{2})'.format(filename, neednum+1, totalnum))
```

## 4. Summary of Experience

In the process of this project, we improved our architecture step by step. At the beginning, we only implement some simple procedures of sending and receiving messages, after that, we gradually added multithreading, header fields, and many kinds of exception handling. We started with communication within on computer and at the end we realized communication between different computers.

We learned a lot from this project and had deeper understanding of P2P networks. The experience of doing this this project improved our project cooperation ability, and it would hopefully benefit our further study.

## 5. Personal Summary

Luo: All the big projects started with some simple and plain codes, and it’s improved over and over to become big and robust. There is no need to design a perfect architecture at beginning, and that’s not possible as well. So, when starting a project, it’s important that you take the first step and don’t get overwhelmed, and of course it’s better to have good coding habits so you won’t change too much of your codes during the improvement.

Lin Xiao: Sometimes it’s hard to find some small mistakes in your own codes. Therefore, partners are playing important roles while debugging one’s program. Making connections between computers is sometimes difficult, especially when you have few knowledge about how the firewalls works. Only within the process of learning, a person could acknowledge the skills to build up a p2p system.

Yukai Zhao: Actually, the method to implement our project is easy. But, none of us knew how to start at the beginning. We thought the project was too hard. Later, we begin with implement a very simple tcp connection between two port of a computer and continuously improve it. Now, I know that when I encounter a big problem, it’s important to depart it into many easy little problems recursively and solve them one by one.

Ji Jie: We had many problems when beginning with the project, but when we came out with the idea of tcp connection, everything became clear. We build tracker, node clients, node servers and tried to send txt file between them. Then we improved the tracker to update with time. Finally, we made the system run on different computers. Each step is not easy and all of us four people need to discuss and cooperate with each other. Teamwork is very important in a big project.