Socket Programming

计算机网络 CS339

李子龙 518070910095

2021 年 10 月 8 日

目录

You will implement a simple file share application using TCP Socket APIs.

You can use any programming language. You need to submit your code together with a report. The report should contain how to run your program and some evaluation results.

1 C/S 模型

1.1 C/S 模型要求

1. Implement C/S model:

   (a) Server listens to a given port (>1024, e.g. 2680)

   (b) Multiple clients request the same file from the server

   (c) Each client save the file to its local directory.

2. Use Mininet to compare the overall file downloading time. Study how the number of downloading time changes with respect to the number of peers. You need to create the following star topology in Mininet. You can use one host as a server, and the other hosts as peers requesting files.
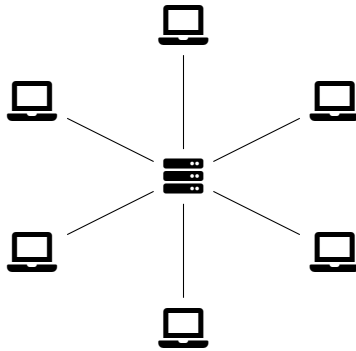
1.2 C/S 模型测试

在 root 模式下，使用 csmodel 文件夹下的 cspy_test.sh 脚本即可测试。

图 1: 单机测试

本地测试将会在服务器与客户端之间，在 **2683** 端口上传输 10 MB 的文件，传输时长大概在 4s 左右。如果最后没有比较上的异常，就说明被完整地传输了。

Listing 1: csmodel/cspy_test.sh

```bash
# !/bin/bash

rm -rf file*
rm -rf result_py.dat
dd if=/dev/zero of=file.txt bs=1024 count=10240
python server.py &
sleep 1
python client.py 10485760
xxd file.txt > file.hex
xxd file_receive.txt > file_receive.hex
# diff file.txt file_send.txt
diff file.hex file_receive.hex
exit
```

而对于 Mininet 模拟来说，使用 `python` 运行 `csmodel/centralized.py` 脚本即可开始测试星状结构的网络。运行脚本后需要输入测试的主机数量。如果脚本后面添加参数 `py --dirty` 则可以跳过一些检查，这对于客户机数量多的情况比较有用。

图 2: C/S Mininet 测试

由图 3 所示，所有的客户机会同时从服务器获取上述的 10MB 文件，每个客户机会在完成传输后各自存储为对应的 `file_receive_h*.txt` 文件，并将计时结果写入文件 `result_py.dat`，当 Mininet 退出 CLI 的时候（`quit`指令），就会对结果进行统计，并给出平均时间。
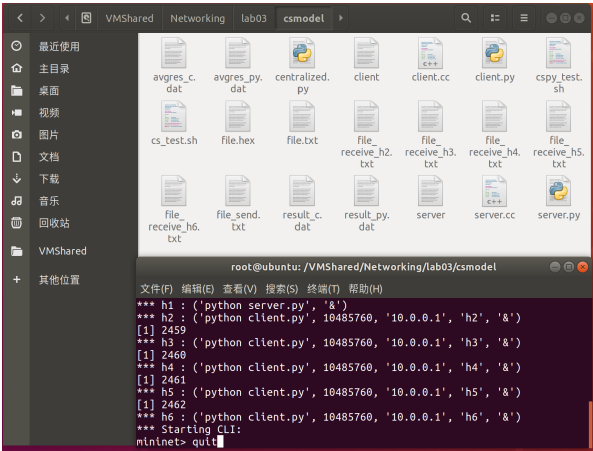

图 3: 运行多机测试

由图 4 可见，随着客户机的增长，传输时间也会显著变长。
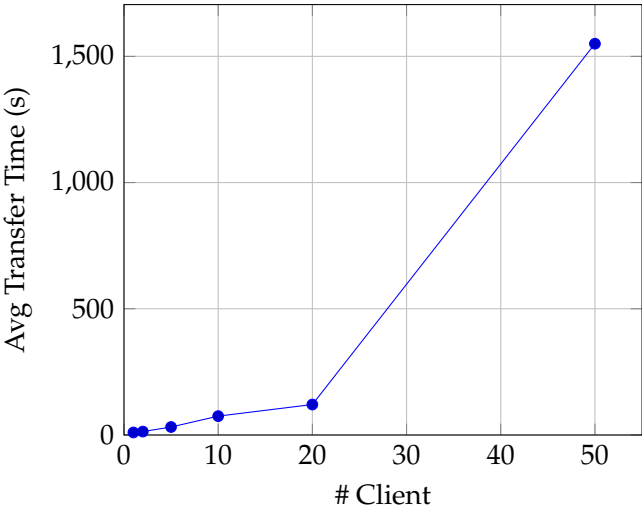
图 4: 不同的客户机数量下平均传输时间

2 P2P 模型

2.1 P2P 模型要求

1. Implement P2P model: Each peer downloads part of the file from the server, and then distributes it to all the other peers.

2. Use Mininet to compare the overall file downloading time under P2P model.

3. For peer-to-peer mode, you may want to design a tracker file first. The tracker file contains the following information:

file chunk id	peer ip
1	10.0.0.1
2	10.0.0.2
3	10.0.0.3
4	10.0.0.4

Then, if a client wants piece 2, it will contact 10.0.0.2. If the peer with ip=10.0.0.2 does not have the piece yet, it will contact the server to download this piece first.

2.2 P2P 模型测试

使用 p2pmodel/p2ptest.sh 可以进行批量测试，这一步可以创建一个 10MB 大小的文件 file.txt。当然也可以直接运行 python 脚本进行测试：

```
sudo python p2pmodel/centralized.py 5
```

最后一个参数用于指定主机数量。

图 5: P2P Mininet 测试

Listing 2: p2pmodel/p2ptest.sh

```bash
# !/bin/bash

rm -rf file*
rm -rf result_py.dat
dd if=/dev/zero of=file.txt bs=1024 count=10240
for hostnumber in 2 3 4 5 6 7 8 9 10 11 12 13 14
do
    python centralized.py $hostnumber
done
exit
```


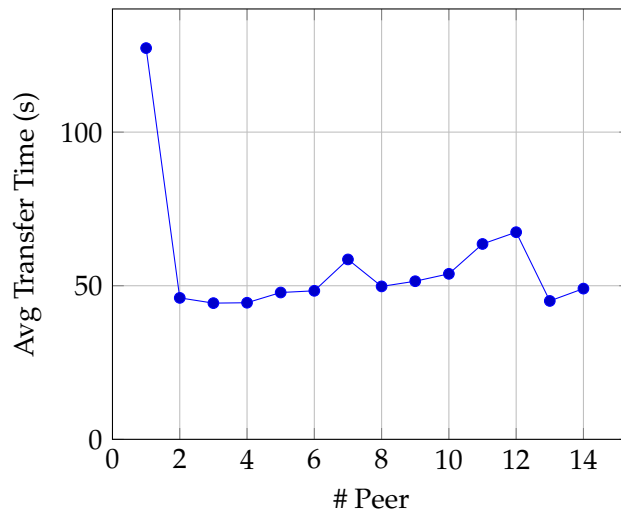图 6: 不同的对等方数量下平均传输时间

A C/S 模型架构实现

下面给出了 Mininet 配置的具体代码。实际上星状网络可以采用 `LinearTopo` 代替，此处为了更好地控制源码，就采用了手写的 `CentralizedTopo` 网络结构。

Listing 3: `csmodel/centralized.py`

```python
# python centralized.py [hostnumber] [py|c] [--dirty]

# [hostnumber] is the number of hosts in the structure (including the server)
# py will launch python script, while c will launch c version.
# --dirty will pass all the checking

# To construct a centralized structure
# for C/S model.

import os, glob
from sys import argv
from time import sleep
from mininet.link import TCLink
from mininet.topo import LinearTopo, Topo
from mininet.net import Mininet
from mininet.log import lg, info
from mininet.util import dumpNodeConnections,irange, run
from mininet.cli import CLI

class CentralizedTopo(Topo):
    "A centralized topo."

    def build(self, hostnumber=2, **params):
        # Central switch
        switch = self.addSwitch('s1')
        # Star hosts
        hosts = [self.addHost('h%s' % h) for h in irange(1,hostnumber)]

        # Star structure
        for host in hosts:
            self.addLink(switch, host)

def FileTransfer(hostnumber=2):
    "Make file transfer between switch and hosts simutanously."

    servercmd = "python server.py"
    clientcmd = "python client.py"
    resultfile = "result_py.dat"
    avgresfile = "avgres_py.dat"

    if len(argv)>=3:
        if(argv[2]=="c"):
            servercmd = "./server"
            clientcmd = "./client"
            resultfile = "result_c.dat"
            avgresfile = "avgres_c.dat"

    dirty = False
```

```python
49        if len(argv)>=4:
50            if(argv[3]=="--dirty"):
51                dirty = True
52
53        topo = CentralizedTopo(hostnumber)
54        # topo = LinearTopo(1,hostnumber) # The same.
55        net = Mininet(topo=topo, link=TCLink, autoStaticArp=False)
56        net.start()
57        # Test connectivity.
58        if not dirty:
59            net.pingAll()
60
61        # clean the files.
62        for file_receive in glob.glob("file_receive*"):
63            os.remove(file_receive)
64
65        fileSize = os.path.getsize("file.txt")
66
67        dumpNodeConnections(net.hosts)
68
69        # Place the server on h1.
70        print(net.hosts[0].cmdPrint(servercmd,"&"))
71
72        # To make sure the client will contact the server.
73        sleep(2)
74
75        # All other host request the file from h1.
76        for i in range(1,hostnumber):
77            net.hosts[i].cmdPrint(clientcmd,fileSize,net.hosts[0].IP(),net.hosts[
   i].name,"" if i == hostnumber - 1 else "&")
78
79        # CLI(net)
80
81        # check the difference.
82        if not dirty:
83            for i in range(1,hostnumber):
84                run("diff file.txt file_receive_"+net.hosts[i].name+".txt")
85
86        # Wait for all hosts complete.
87        complete = 0
88        while not complete == hostnumber:
89            results = []
90            with open(resultfile,"r") as rf:
91                resultlines = rf.read().splitlines()
92                complete = len(resultlines)
93                for i in range(1,len(resultlines)):
94                    results.append(float(resultlines[i].split('\t')[1]))
95            sleep(10)
96
97        avg = sum(results)/len(results)
98        print("Average: "+str(avg))
99        print("Avaliable: " + str(len(results)) +  "/" + str(hostnumber-1) + "("
   + str(int(len(results)*100/(hostnumber-1))) + "%)")
100       with open(avgresfile,"a") as af:
101           af.write(str(hostnumber-1)+"\t"+str(avg)+"\n")
102
```

```
103        net.stop()
104
105    if __name__=="__main__":
106        lg.setLogLevel( 'info' )
107        if len(argv)>=2:
108            hostnumber = int(argv[1])
109        else:
110            hostnumber = int(input("Please input hostnumber:"))
111        FileTransfer(hostnumber)
```

B C/S 模型 Python 实现 🐍

为了方便理解 TCP 网络模型，先跟着书用 Python 实现了一遍多线程 C/S 模型。

下面是服务器端的代码，在主循环中由 `accept()` 阻塞接收，一旦有客户机进入就会开辟新的线程进行文件传输。

Listing 4: csmodel/server.py

```python
1    # python server.py [&]
2
3    from socket import *
4    import threading
5
6    serverPort = 2683
7    serverSocket = socket(AF_INET, SOCK_STREAM)
8    serverSocket.bind(('', serverPort))
9    serverSocket.listen(20)
10   serverSocket.settimeout(20)
11
12   with open("file.txt","rb") as f:
13       content = f.read()
14
15   class SendThread(threading.Thread):
16       def __init__(self, connectionSocket):
17           threading.Thread.__init__(self)
18           self.connectionSocket = connectionSocket
19       def run(self):
20           sendout(self.connectionSocket)
21
22   def sendout(connectionSocket):
23       connectionSocket.send(content.encode())
24       connectionSocket.close()
25
26   with open("file_send.txt","wb") as f:
27       f.write(content)
28   with open("result_py.dat","w") as rf:
29       rf.write("Host\tTime\n")
30   print('Server ready to send.')
31
32   while True:
33       try:
34           connectionSocket, addr = serverSocket.accept()
```

```python
35      except timeout:
36          print("Server closed.")
37          break
38      print(addr)
39      sth = SendThread(connectionSocket)
40      sth.start()
```

下面是客户端代码，由于 TCP 连接接收大小有一定的限制，超过限制可能就会产生长期接收不到的情况，所以每次只会接收 1024 字节，分为多次进行接收。由于 Python 内存访问比较慢，这里直接就写入了文件，没有首先将内容进入内存。

Listing 5: csmodel/client.py

```python
1   from socket import *
2   from sys import argv
3   import time
4
5   MAXLINE = 1024
6   serverName = "127.0.0.1"
7   filename = "file_receive.txt"
8   hostname = ""
9
10  if len(argv)>=2:
11      MAXLINE = int(argv[1])
12  if len(argv)>=3:
13      serverName = argv[2]
14  if len(argv)>=4:
15      hostname = argv[3]
16      filename = "file_receive_" + hostname + ".txt"
17
18  START_TIME = time.time()
19
20  serverPort = 2683
21  clientSocket = socket(AF_INET, SOCK_STREAM)
22  clientSocket.connect((serverName, serverPort))
23
24  # To speed up, skip the process of saving it to the memory first.
25  with open(filename,"wb") as f:
26      for i in range(MAXLINE/1024):
27          f.write(clientSocket.recv(1024).decode())
28
29  END_TIME = time.time()
30
31  clientSocket.close()
32
33  elapsed_time = END_TIME-START_TIME
34  print(elapsed_time)
35
36  with open("result_py.dat","a") as rf:
37      rf.write(hostname + "\t" + str(elapsed_time) + "\n")
```

C C/S 模型 C 实现 C

如果想运行 C 语言版本，进行单机测试：

图 7: C 语言单机测试

进行 Mininet 测试需要使用参数 `c --dirty`。

图 8: C 语言多机测试

可见使用 C 语言效率更高，1s 不到就可以完成，这也导致如果需要对其评测需要使用很大的文件，否则数据差距并不明显，这里就跳过对 C 语言实现版本的测评部分。下面的代码主要是对示例代码的一些改进而得到的。

Listing 6: csmodel/server.cc

```
1  #include <stdio.h> // standard input and output library
2  #include <stdlib.h> // this includes functions regarding memory allocation
3  #include <string.h> // contains string functions
4  #include <errno.h> //It defines macros for reporting and retrieving error
        conditions through error codes
5  #include <unistd.h> //contains various constants
6  #include <sys/types.h> //contains a number of basic derived types that should
        be used whenever appropriate
```

```c
7  #include <arpa/inet.h> // defines in_addr structure
8  #include <sys/socket.h> // for socket creation
9  #include <netinet/in.h> //contains constants and structures needed for
       internet domain addresses
10 #include <pthread.h>
11
12 char* dataSending; // Actually this is called packet in Network Communication
       , which contain data and send through.
13 int fsize;
14
15 void* send_thread(void* conn_void_ptr){
16     int clintConnt = *((int*)conn_void_ptr);
17     pthread_detach(pthread_self());
18     free(conn_void_ptr);
19     write(clintConnt, dataSending, sizeof(char)*fsize);
20     close(clintConnt);
21 }
22
23 int main()
24 {
25     int clintListn = 0;
26     struct sockaddr_in ipOfServer;
27     clintListn = socket(AF_INET, SOCK_STREAM, 0); // creating socket
28     memset(&ipOfServer, '0', sizeof(ipOfServer));
29     ipOfServer.sin_family = AF_INET;
30     ipOfServer.sin_addr.s_addr = htonl(INADDR_ANY);
31     ipOfServer.sin_port = htons(2680);      // this is the port number of
       running server
32     bind(clintListn, (struct sockaddr*)&ipOfServer , sizeof(ipOfServer));
33     listen(clintListn , 20);
34
35     int* clintConnt;
36     pthread_t th;
37
38     FILE* f;
39     if((f = fopen("file.txt","rb"))==NULL){
40         fprintf(stderr, "Can't open file file.txt.\n");
41         return -1;
42     }
43     fseek(f,0,SEEK_END);
44     fsize = ftell(f);
45     fseek(f,0,SEEK_SET);
46     dataSending = (char*)malloc(sizeof(char)*fsize);
47     printf("%d\n",fsize);
48     fread(dataSending, sizeof(char), fsize, f);
49     fclose(f);
50
51     f = fopen("file_send.txt","wb");
52     fwrite(dataSending,sizeof(char),fsize,f);
53     fclose(f);
54
55     f = fopen("result_c.dat","w");
56     fprintf(f,"Host\tTime\n");
57     fclose(f);
58
59     while(1)
```

```
60      {
61          printf("\n\nHi,I am running server.Some Client hit me\n"); //
        whenever a request from client came. It will be processed here.
62
63          clintConnt = (int*)malloc(sizeof(int));
64          *clintConnt = accept(clintListn, (struct sockaddr*)NULL, NULL);
65
66          pthread_create(&th, NULL, send_thread, clintConnt);
67      }
68
69      return 0;
70  }
```

Listing 7: csmodel/client.cc

```
1   #include <sys/socket.h>
2   #include <sys/types.h>
3   #include <netinet/in.h>
4   #include <netdb.h>
5   #include <stdio.h>
6   #include <string.h>
7   #include <stdlib.h>
8   #include <unistd.h>
9   #include <errno.h>
10  #include <arpa/inet.h>
11  #include <time.h> //contains various functions for manipulating date and time
12
13  int main(int argc, char* argv[])
14  {
15      struct sockaddr_in ipOfServer;
16      ipOfServer.sin_family = AF_INET;
17      ipOfServer.sin_port = htons(2680);
18      ipOfServer.sin_addr.s_addr = inet_addr("127.0.0.1");
19
20      char filename[80];
21      sprintf(filename, "file_receive.txt");
22
23      int MAXLINE = 1024;
24
25      switch (argc)
26      {
27      case 1:
28          printf("./client [file_size] [target_ip] [host_number]\n");
29          return 0;
30      case 4:
31          sprintf(filename, "file_receive_%s.txt",argv[3]);
32      case 3:
33          ipOfServer.sin_addr.s_addr = inet_addr(argv[2]);
34      case 2:
35          MAXLINE = atoi(argv[1]);
36      default:
37          break;
38      }
39
40      clock_t start,finish;
41      start = clock();
```

```
42
43      int CreateSocket = 0,n = 0;
44      char* dataReceived = (char*)malloc(MAXLINE*sizeof(char));
45      memset(dataReceived, 1 ,sizeof(dataReceived));
46
47      if((CreateSocket = socket(AF_INET, SOCK_STREAM, 0))< 0)
48      {
49          printf("Socket not created \n");
50          return 1;
51      }
52
53      if(connect(CreateSocket, (struct sockaddr *)&ipOfServer, sizeof(
        ipOfServer))<0)
54      {
55          printf("Connection failed due to port and ip problems\n");
56          return 1;
57      }
58
59      for(int i = 0; i < MAXLINE/1024; ++i)
60          read(CreateSocket, dataReceived + (i*1024) , 1024);
61
62      FILE* f = fopen(filename,"wb");
63      fwrite(dataReceived,sizeof(char),MAXLINE,f);
64      fclose(f);
65
66      finish = clock();
67
68      FILE* rf = fopen("result_c.dat","a");
69      double elapsed_time = (double)(finish-start)/CLOCKS_PER_SEC;
70      if (argc==4)
71          fprintf(rf,"%s\t%f\n",argv[3],elapsed_time);
72      else fprintf(rf,"\t%f\n",elapsed_time);
73      fclose(rf);
74      fprintf(stdout, "%f\n", elapsed_time);
75
76      if( n < 0)
77      {
78          printf("Standard input error \n");
79      }
80
81      return 0;
82  }
```

D P2P 模型架构实现

Listing 8: p2pmodel/centralized.py

```
1  # To construct a centralized structure
2  # for P2P model.
3
4  import os, glob
5  from sys import argv
6  from time import sleep
```

```python
import math
from mininet.link import TCLink
from mininet.topo import LinearTopo, Topo
from mininet.net import Mininet
from mininet.log import lg, info
from mininet.util import dumpNodeConnections,irange, quietRun, run
from mininet.cli import CLI

class CentralizedTopo(Topo):
    "A centralized topo."

    def build(self, hostnumber=2, **params):
        # Central switch
        switch = self.addSwitch('s1')
        # Star hosts
        hosts = [self.addHost('h%s' % h) for h in irange(1,hostnumber)]

        # Star structure
        for host in hosts:
            self.addLink(switch, host)

def FileTransfer(hostnumber=2):
    "Make file transfer between switch and hosts simutanously."

    servercmd = "python server.py"
    clientcmd = "python peer.py"
    resultfile = "result_py.dat"
    avgresfile = "avgres_py.dat"

    topo = CentralizedTopo(hostnumber)
    net = Mininet(topo=topo, link=TCLink, autoStaticArp=False)
    net.start()

    # clean the files.
    for file_receive in glob.glob("file_receive*"):
        os.remove(file_receive)

    # Generate tracker file
    fileSize = os.path.getsize("file.txt")
    chunkSize = int(math.ceil(fileSize/(hostnumber-1)))
    with open("tracker.dat","w") as tf:
        for i in range(hostnumber-1):
            tf.write(str(i)+"\t"+net.hosts[i+1].IP()+"\n")

    # Try to dump
    dumpNodeConnections(net.hosts)

    # Place the server on h1.
    net.hosts[0].cmdPrint(servercmd,chunkSize,"&")

    sleep(2)

    # All other host request files.
    # The last host will be monitored.
    for i in range(1,hostnumber):
```

```python
62          net.hosts[i].cmdPrint(clientcmd,chunkSize,net.hosts[0].IP(),net.hosts
     [i].IP(),"" if i == hostnumber - 1 else "&")
63
64      # CLI(net)
65
66      results = []
67      with open(resultfile,"r") as rf:
68          resultlines = rf.read().splitlines()
69          for i in range(1,len(resultlines)):
70              results.append(float(resultlines[i].split('\t')[1]))
71      avg = sum(results)/len(results) if not len(results) == 0 else -1
72      print("Average: "+str(avg))
73      print("Avaliable: " + str(len(results)) +  "/" + str(hostnumber-1) + "("
     + str(int(len(results)*100/(hostnumber-1))) + "%)")
74      with open(avgresfile,"a") as af:
75          af.write(str(hostnumber-1)+"\t"+str(avg)+"\n")
76
77      net.stop()
78
79  if __name__=="__main__":
80      lg.setLogLevel( 'info' )
81      if len(argv)>=2:
82          hostnumber = int(argv[1])
83      else:
84          hostnumber = int(input("Please input hostnumber:"))
85      FileTransfer(hostnumber)
```

E P2P 模型 PYTHON 实现 🐍

服务器与 C/S 模型的差距不大，主要可以接收一个文件块的参数。

Listing 9: p2pmodel/server.py

```python
1   # python server.py [chunkSize]
2
3   from socket import *
4   from sys import argv
5   import threading
6
7   serverPort = 2684
8   serverSocket = socket(AF_INET, SOCK_STREAM)
9   serverSocket.bind(('', serverPort))
10  serverSocket.listen(20)
11
12  with open("file.txt","rb") as f:
13      content = f.read()
14
15  chunkSize = 1024
16  if len(argv)==2:
17      chunkSize = int(argv[1])
18
19  class SendThread(threading.Thread):
20      def __init__(self, connectionSocket, chunkNumber):
```

```python
            threading.Thread.__init__(self)
            self.connectionSocket = connectionSocket
            self.chunkNumber = chunkNumber
    def run(self):
        sendout(self.connectionSocket, chunkNumber)

def sendout(connectionSocket, cnum):
    connectionSocket.send(content[cnum*chunkSize:(cnum+1)*chunkSize].encode()
    )
    connectionSocket.close()

with open("file_send.txt","wb") as f:
    f.write(content)
with open("result_py.dat","w") as rf:
    rf.write("Host\tTime\n")
print('Ready to send.')

while True:
    connectionSocket, addr = serverSocket.accept()
    chunkNumber = int(connectionSocket.recv(1024).decode())
    print(addr, chunkNumber)
    sth = SendThread(connectionSocket, chunkNumber)
    sth.start()
```

对等方的实现相对复杂。

Listing 10: p2pmodel/peer.py

```python
# python peer.py [chunkSize] [serverIP] [hostIP]

from socket import *
from sys import argv
import threading
import time

chunkSize = 1024
serverName = "127.0.0.1"
filename = "file_receive.txt"
hostName = "127.0.0.1"
hostname = ""

if len(argv)>=2:
    chunkSize = int(argv[1])
if len(argv)>=3:
    serverName = argv[2]
if len(argv)>=4:
    hostName = argv[3]
    hostname = "h" + hostName.split(".")[-1]
    filename = "file_receive_" + hostname + ".txt"

START_TIME = time.time()

with open("tracker.dat","r") as tf:
    tls = tf.read().splitlines()
    # chunkID \t requestIP
    trackers = [[int(tl.split("\t")[0]),tl.split("\t")[1],False] for tl in
```

```
      tls]
29
30  content = {}
31
32  def DownContent(chunkNumber):
33      # Check if this peer has the chunk first.
34      if not trackers[chunkNumber][2]:
35          # Server use 2684 for communication.
36          clientPort = 2684
37          clientSocket = socket(AF_INET, SOCK_STREAM)
38          clientSocket.connect((serverName, clientPort))
39          # send out the chunkNumber
40          clientSocket.send(str(chunkNumber).encode())
41
42          # expect to receive the chunk from the server.
43          fileChunk = ""
44          for i in range(chunkSize/1024):
45              fileChunk = fileChunk + clientSocket.recv(1024).decode()
46          # threadLock.acquire()
47          content[chunkNumber] = fileChunk          # write to self content
48          # threadLock.release()
49          trackers[chunkNumber][2] = True           # declare to be useable
50          print("Received Size from " + serverName + " :" + str(len(fileChunk))
      )
51
52          clientSocket.close()
53      else:
54          fileChunk = content[chunkNumber]
55      return fileChunk
56
57  class SendThread(threading.Thread):
58      def __init__(self, connectionSocket, chunkNumber):
59          threading.Thread.__init__(self)
60          self.chunkNumber = chunkNumber
61          self.connectionSocket = connectionSocket
62      def run(self):
63          # Detect the downloading first.
64          downFileChunk = DownContent(self.chunkNumber)
65
66          # Then send it to the peer
67          self.connectionSocket.send(downFileChunk.encode())
68          self.connectionSocket.close()
69
70  class ServiceThread(threading.Thread):
71      def __init__(self):
72          threading.Thread.__init__(self)
73      def run(self):
74          # peer use 2685 for communication.
75          receivePeerPort = 2685
76          receivePeerSocket = socket(AF_INET, SOCK_STREAM)
77          receivePeerSocket.bind(('', receivePeerPort))
78          receivePeerSocket.listen(20)
79          receivePeerSocket.settimeout(10)
80
81          while True:
82              try:
```

```python
                    connectionSocket, addr = receivePeerSocket.accept()
                except timeout:
                    if len(content)==len(trackers):
                        # The transmission has been complete.
                        break
                    else:
                        continue
                chunkNumber = int(connectionSocket.recv(1024).decode())
                print(addr, chunkNumber)
                sth = SendThread(connectionSocket,chunkNumber)
                sth.start()

def getContent(tracker):
    sendPeerPort = 2685
    sendPeerSocket = socket(AF_INET, SOCK_STREAM)

    try:
        time.sleep(0.5)
        sendPeerSocket.connect((tracker[1],sendPeerPort))
    except:
        print("- Retry connect to "+ tracker[1] + " after 3 sec.")
        time.sleep(3)
        sendPeerSocket.connect((tracker[1],sendPeerPort))
        print("+ Connect to "+ tracker[1] + " success.")

    sendPeerSocket.send(str(tracker[0]).encode())
    # expect to receive the chunk from the server.
    receivedFileChunk = ""
    for i in range(chunkSize/1024):
        receivedFileChunk = receivedFileChunk + sendPeerSocket.recv(1024).
    decode()
    print("Received Size from " + tracker[1] + " :" + str(len(
    receivedFileChunk)))
    content[tracker[0]] = receivedFileChunk          # write to self content
    trackers[tracker[0]][2] = True          # declare to be useable
    sendPeerSocket.close()

class ReceiveThread(threading.Thread):
    def __init__(self, tracker):
        threading.Thread.__init__(self)
        self.tracker = tracker
    def run(self):
        getContent(self.tracker)
class FetchingThread(threading.Thread):
    def __init__(self):
        threading.Thread.__init__(self)
    def run(self):
        rths = []
        for tracker in trackers:
            # To avoid conflict, will be called by other peer.
            if not tracker[1] == hostName:
                rth = ReceiveThread(tracker)
                rth.start()
                rths.append(rth)
        for rth in rths:
            rth.join()
```

```python
137
138  # Start servicing.
139  serth = ServiceThread()
140  serth.start()
141
142  # Start fetching
143  fetth = FetchingThread()
144  fetth.start()
145
146  # Wait for merge
147  fetth.join()
148
149  # Now, force to download the own chunk if it is not downloaded.
150  for tracker in trackers:
151      if tracker[1] == hostName and tracker[2] == False:
152          DownContent(tracker[0])
153
154  # should be done with all file chunks, unless error occurred.
155  with open(filename,"wb") as f:
156      for i in range(len(content)):
157          if i in content.keys():
158              f.write(content[i])
159          else:
160              print("- " + hostname + " lost chunk " + str(i))
161
162  END_TIME = time.time()
163
164  elapsed_time = END_TIME-START_TIME
165  print(elapsed_time)
166
167  with open("result_py.dat","a") as rf:
168      rf.write(hostname + "\t" + str(elapsed_time) + "\n")
```