Socket Programming

计算机网络 CS339

李子龙 518070910095

2021 年 10 月 9 日

目录

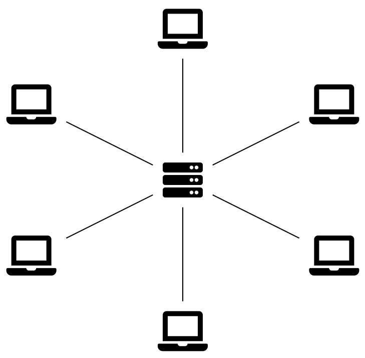
You will implement a simple file share application using TCP Socket APIs.

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

1.2.1 批量测试

```
cd csmodel
./cstest.sh
```

这将会生成 10 MB 的随机文件 `file.txt`，运行 Python 🐍 版本的脚本进行模拟。
由图 1 可见，随着客户机的增长，传输时间也会线性变长。

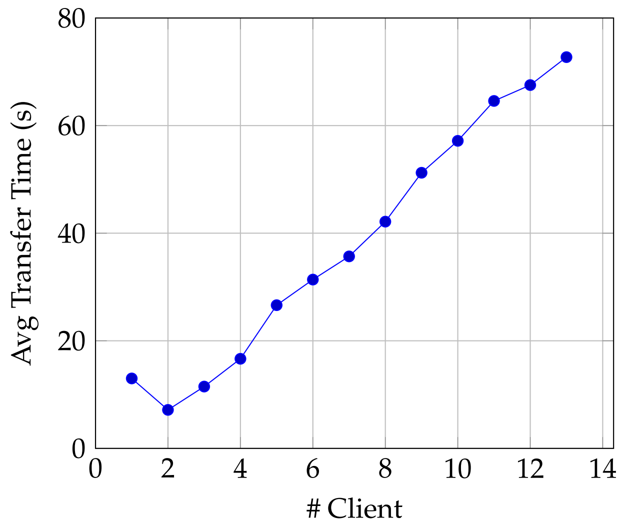


图 1: 不同的客户机数量下平均传输时间

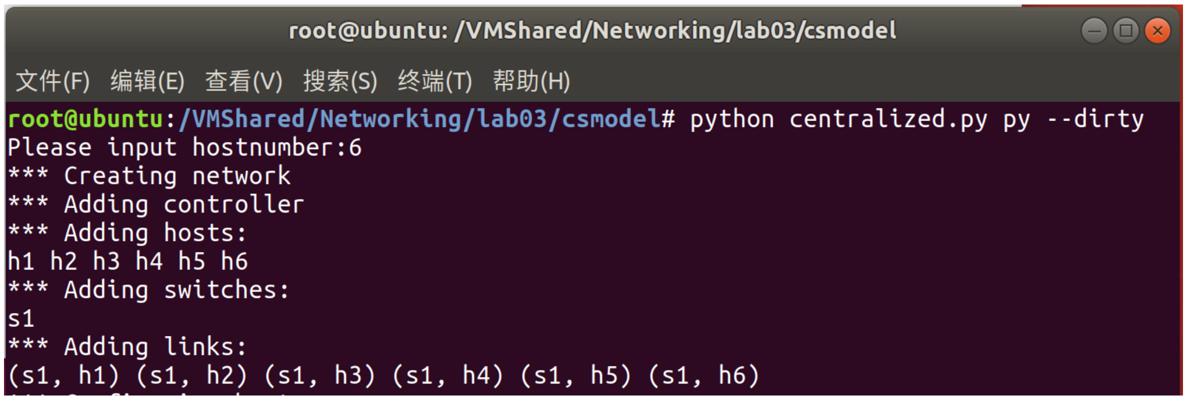
Listing 1: csmodel/cstest.sh

```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 py --dirty
done
exit
```

1.2.2 单次测试

```
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
```

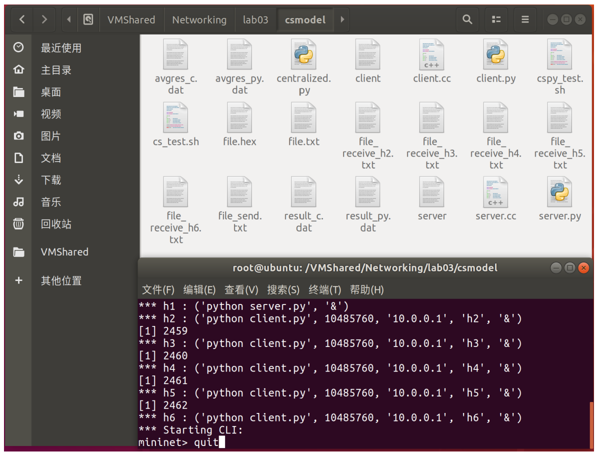
使用 python 运行 csmodel/centralized.py 脚本即可开始测试星状结构的网络。如果没有给定主机数量，运行脚本后就会提示输入。如果脚本后面添加参数 py --dirty 则可以跳过一些检查，这对于客户机数量多的情况比较有用。



```
root@ubuntu:/VMShared/Networking/lab03/csmodel# python centralized.py py --dirty
Please input hostnumber:6
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2 h3 h4 h5 h6
*** Adding switches:
s1
*** Adding links:
(s1, h1) (s1, h2) (s1, h3) (s1, h4) (s1, h5) (s1, h6)
```

图 2: C/S Mininet 测试

C/S 模型架构实现代码见附录 A。由图 3 所示，所有的客户机会同时从服务器获取上述的 10 MB 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`。由于最后一个主机进程会阻塞 Mininet 测试主线程，所以最后一个结束后，等待片刻，就会对结果进行统计，并给出平均时间。



```
*** h1 : ('python server.py', '&')
*** h2 : ('python client.py', 10485760, '10.0.0.1', 'h2', '&')
[1] 2459
*** h3 : ('python client.py', 10485760, '10.0.0.1', 'h3', '&')
[1] 2460
*** h4 : ('python client.py', 10485760, '10.0.0.1', 'h4', '&')
[1] 2461
*** h5 : ('python client.py', 10485760, '10.0.0.1', 'h5', '&')
[1] 2462
*** h6 : ('python client.py', 10485760, '10.0.0.1', 'h6', '&')
*** Starting CLI:
mininet> quit
```

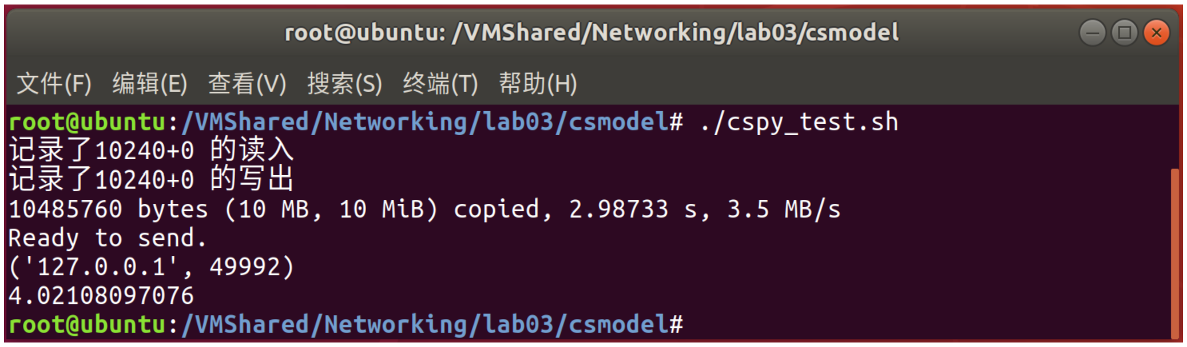
图 3: 运行多机测试

1.2.3 单机测试

```
./cspy_test.sh
```

使用 `csmodel` 文件夹下的 `cspy_test.sh` 脚本即可测试 Python 版本的单服务器—单客

户端模型，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。


root@ubuntu: /VMShared/Networking/lab03/csmodel
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)

```
root@ubuntu:/VMShared/Networking/lab03/csmodel# ./cspy_test.sh
记录了10240+0 的读入
记录了10240+0 的写出
10485760 bytes (10 MB, 10 MiB) copied, 2.98733 s, 3.5 MB/s
Ready to send.
('127.0.0.1', 49992)
4.02108097076
root@ubuntu:/VMShared/Networking/lab03/csmodel#
```

图 4: 单机测试

Listing 2: csmodel/cspy_test.sh

```
# !/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
```

C/S 模型的服务器和客户端 Python 实现见附录 B。

当然也参考着示例代码实现了 C 语言版本，详见附录 C。C 语言实现版本的速度快很多，基本上不会超过 0.5s 就可以完成上面 Python 的操作，这样会导致数据差别小，就不给出 C 语言测试版本的运行时间测试了，注意 C 语言版本使用了 2680 端口。

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

2.2.1 批量测试

```
cd p2pmodel
./p2ptest.sh
```

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`。图 7 显示了对等方的数目多少不会导致传输时间的太大波动，至少不是 C/S 模型的线性增长。

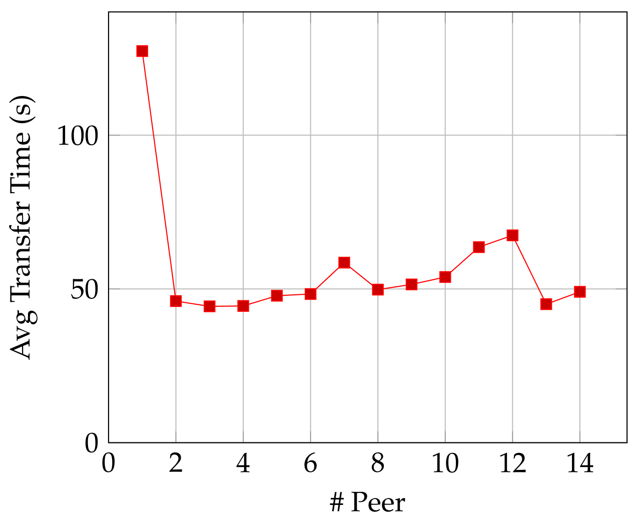


图 5: 不同的对等方数量下平均传输时间

Listing 3: p2pmodel/p2ptest.sh

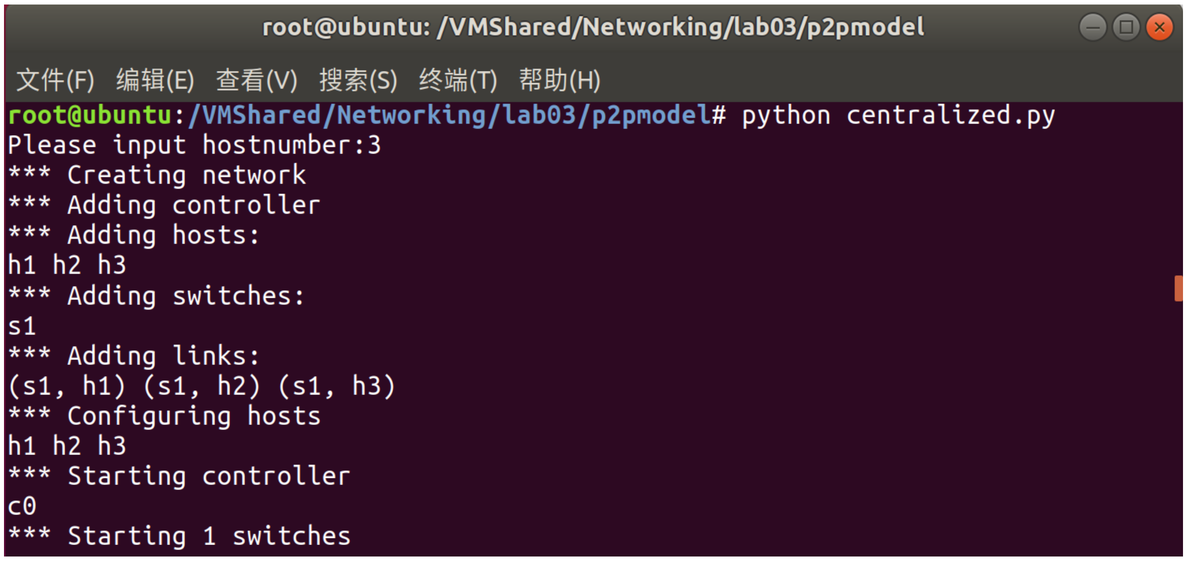
```
# !/bin/bash

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
```

2.2.2 单次测试

当然也可以直接运行 python 脚本进行测试：

```
python centralized.py 5
```

最后一个参数用于指定主机数量。请注意，应当先生成文件再进行单次测试。


root@ubuntu: /VMShared/Networking/lab03/p2pmodel
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)

```
root@ubuntu:/VMShared/Networking/lab03/p2pmodel# python centralized.py
Please input hostnumber:3
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2 h3
*** Adding switches:
s1
*** Adding links:
(s1, h1) (s1, h2) (s1, h3)
*** Configuring hosts
h1 h2 h3
*** Starting controller
c0
*** Starting 1 switches
```

图 6: P2P Mininet 测试

附录 D 显示了 P2P 模型的 Mininet 架构实现。制定了主机数量后，会生成对应的 `tracker.dat` 文件，和题设一样的格式。

附录 E 显示了 P2P 模型的服务端和对等端实现。关于对等方，重要的一点是每个对等方也会新建一个服务器线程，以向其他对等方发送数据，如果本地没有文件，就会先阻塞地向服务器获取数据。对等端内部会使用一个数据结构存储已经得到的文件块信息，最后会等待所有向其他对等方的传输线程终止，才会准备将数据写入文件。由于实现起来较为复杂，就不提供 C 语言实现版本。

3 结论

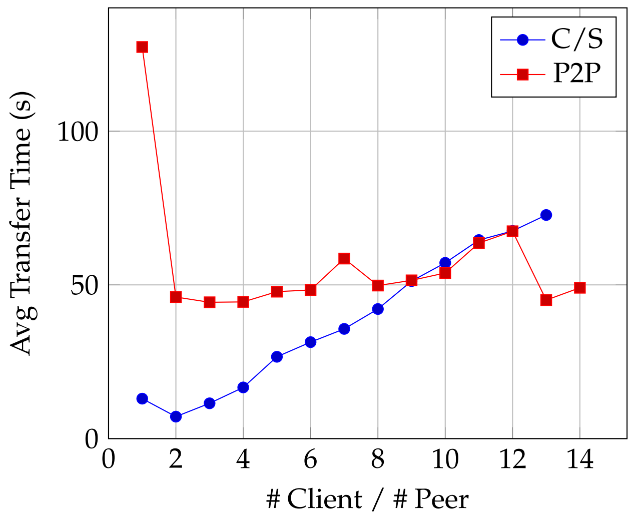


图 7: 平均传输时间比较

将统计数据放在一张图上，会发现后面 C/S 模型仍然在线性增长，而 P2P 模型并不怎么增长。而 P2P 在创建线程（增加连接数）上的开销要比 C/S 模型大很多，使用本地虚拟机的 Mininet 模拟会在这个方面遭受性能问题，对等方数目再多甚至会报错。

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。`dumpNodeConnection` 是重要的，这样才可以防止 IP 地址被重复使用。

Listing 4: `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
    if len(argv)>=4:
        if(argv[3]=="--dirty"):
            dirty = True

    topo = CentralizedTopo(hostnumber)
    # topo = LinearTopo(1,hostnumber) # The same.
    net = Mininet(topo=topo, link=TCLink, autoStaticArp=False)
    net.start()
    # Test connectivity.
    if not dirty:
        net.pingAll()

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

    fileSize = os.path.getsize("file.txt")

    dumpNodeConnections(net.hosts)

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

    # To make sure the client will contact the server.
    sleep(2)

    # All other host request the file from h1.
    for i in range(1,hostnumber):
        net.hosts[i].cmdPrint(clientcmd,fileSize,net.hosts[0].IP(),net.hosts[
    i].name,"" if i == hostnumber - 1 else "&")

    # CLI(net)

    # check the difference.
    if not dirty:
        for i in range(1,hostnumber):
            run("diff file.txt file_receive_"+net.hosts[i].name+".txt")

    # Wait for all hosts complete.
    complete = 0
    while not complete == hostnumber:
        results = []
        with open(resultfile,"r") as rf:
            resultlines = rf.read().splitlines()
            complete = len(resultlines)
            for i in range(1,len(resultlines)):
                results.append(float(resultlines[i].split('\t')[1]))
        sleep(10)

    avg = sum(results)/len(results)
    print("Average: "+str(avg))
    print("Avaliable: " + str(len(results)) +  "/" + str(hostnumber-1) + "("
    + str(int(len(results)*100/(hostnumber-1))) + "%)")
    with open(avgresfile,"a") as af:
        af.write(str(hostnumber-1)+"\t"+str(avg)+"\n")

    net.stop()

if __name__=="__main__":
    lg.setLogLevel( 'info' )
    if len(argv)>=2:
        hostnumber = int(argv[1])
    else:
        hostnumber = int(input("Please input hostnumber:"))
    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 5: csmodel/server.py

```python
# python server.py [&]

from socket import *
import threading

serverPort = 2683
serverSocket = socket(AF_INET, SOCK_STREAM)
serverSocket.bind(('', serverPort))
serverSocket.listen(20)
serverSocket.settimeout(20)

with open("file.txt","rb") as f:
    content = f.read()

class SendThread(threading.Thread):
    def __init__(self, connectionSocket):
        threading.Thread.__init__(self)
        self.connectionSocket = connectionSocket
    def run(self):
        sendout(self.connectionSocket)

def sendout(connectionSocket):
    connectionSocket.send(content.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('Server ready to send.')

while True:
    try:
        connectionSocket, addr = serverSocket.accept()
    except timeout:
        print("Server closed.")
        break
    print(addr)
    sth = SendThread(connectionSocket)
    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。

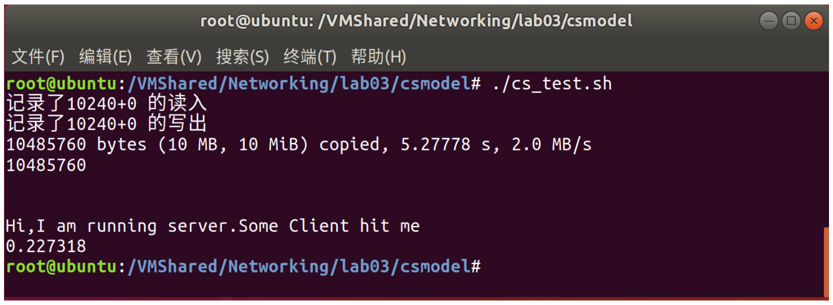
Listing 6: csmodel/client.py

```python
from socket import *
from sys import argv
import time
import math

MAXLINE = 1024
```

```python
serverName = "127.0.0.1"
filename = "file_receive.txt"
hostname = ""

if len(argv)>=2:
    MAXLINE = int(argv[1])
if len(argv)>=3:
    serverName = argv[2]
if len(argv)>=4:
    hostname = argv[3]
    filename = "file_receive_" + hostname + ".txt"

START_TIME = time.time()

serverPort = 2683
clientSocket = socket(AF_INET, SOCK_STREAM)
clientSocket.connect((serverName, serverPort))

# To speed up, skip the process of saving it to the memory first.
with open(filename,"wb") as f:
    for i in range(int(math.ceil(MAXLINE/1024))):
        f.write(clientSocket.recv(1024).decode())

END_TIME = time.time()

clientSocket.close()

elapsed_time = END_TIME-START_TIME
print(elapsed_time)

with open("result_py.dat","a") as rf:
    rf.write(hostname + "\t" + str(elapsed_time) + "\n")
```

C C/S 模型 C 实现 C

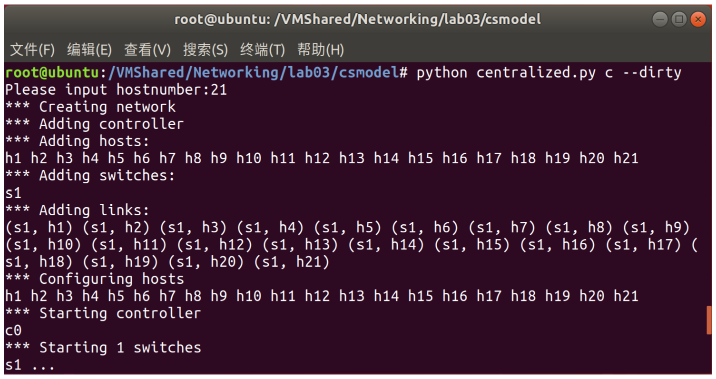
如果想运行 C 语言版本，进行单机测试：


root@ubuntu: /VMShared/Networking/lab03/csmodel
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)

```
root@ubuntu:/VMShared/Networking/lab03/csmodel# ./cs_test.sh
记录了10240+0 的读入
记录了10240+0 的写出
10485760 bytes (10 MB, 10 MiB) copied, 5.27778 s, 2.0 MB/s
10485760


Hi,I am running server.Some Client hit me
0.227318
root@ubuntu:/VMShared/Networking/lab03/csmodel#
```

图 8: C 语言单机测试

进行 Mininet 测试需要使用参数 `c --dirty`。


root@ubuntu: /VMShared/Networking/lab03/csmodel

```
root@ubuntu:/VMShared/Networking/lab03/csmodel# python centralized.py c --dirty
Please input hostnumber:21
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13 h14 h15 h16 h17 h18 h19 h20 h21
*** Adding switches:
s1
*** Adding links:
(s1, h1) (s1, h2) (s1, h3) (s1, h4) (s1, h5) (s1, h6) (s1, h7) (s1, h8) (s1, h9)
(s1, h10) (s1, h11) (s1, h12) (s1, h13) (s1, h14) (s1, h15) (s1, h16) (s1, h17) (
s1, h18) (s1, h19) (s1, h20) (s1, h21)
*** Configuring hosts
h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13 h14 h15 h16 h17 h18 h19 h20 h21
*** Starting controller
c0
*** Starting 1 switches
s1 ...
```

图 9: 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 7: csmodel/server.cc

```c
#include <stdio.h> // standard input and output library
#include <stdlib.h> // this includes functions regarding memory allocation
#include <string.h> // contains string functions
#include <errno.h> //It defines macros for reporting and retrieving error
    conditions through error codes
#include <unistd.h> //contains various constants
#include <sys/types.h> //contains a number of basic derived types that should
     be used whenever appropriate
#include <arpa/inet.h> // defines in_addr structure
#include <sys/socket.h> // for socket creation
#include <netinet/in.h> //contains constants and structures needed for
    internet domain addresses
#include <pthread.h>

char* dataSending; // Actually this is called packet in Network Communication
    , which contain data and send through.
int fsize;

void* send_thread(void* conn_void_ptr){
    int clintConnt = *((int*)conn_void_ptr);
    pthread_detach(pthread_self());
    free(conn_void_ptr);
    write(clintConnt, dataSending, sizeof(char)*fsize);
    close(clintConnt);
}

int main()
{
    int clintListn = 0;
    struct sockaddr_in ipOfServer;
    clintListn = socket(AF_INET, SOCK_STREAM, 0); // creating socket
    memset(&ipOfServer, '0', sizeof(ipOfServer));
    ipOfServer.sin_family = AF_INET;
    ipOfServer.sin_addr.s_addr = htonl(INADDR_ANY);
    ipOfServer.sin_port = htons(2680);      // this is the port number of
running server
    bind(clintListn, (struct sockaddr*)&ipOfServer , sizeof(ipOfServer));
    listen(clintListn , 20);

    int* clintConnt;
    pthread_t th;

    FILE* f;
    if((f = fopen("file.txt","rb"))==NULL){
        fprintf(stderr, "Can't open file file.txt.\n");
        return -1;
    }
    fseek(f,0,SEEK_END);
    fsize = ftell(f);
    fseek(f,0,SEEK_SET);
    dataSending = (char*)malloc(sizeof(char)*fsize);
    printf("%d\n",fsize);
    fread(dataSending, sizeof(char), fsize, f);
    fclose(f);

    f = fopen("file_send.txt","wb");
    fwrite(dataSending,sizeof(char),fsize,f);
    fclose(f);

    f = fopen("result_c.dat","w");
    fprintf(f,"Host\tTime\n");
    fclose(f);

    while(1)
    {
        printf("\n\nHi,I am running server.Some Client hit me\n"); //
whenever a request from client came. It will be processed here.

        clintConnt = (int*)malloc(sizeof(int));
        *clintConnt = accept(clintListn, (struct sockaddr*)NULL, NULL);

        pthread_create(&th, NULL, send_thread, clintConnt);
    }

    return 0;
}
```

服务器会首先读取文件数据。与一个客户机建立连接后，就会开辟一个新的线程，但是这个线程会与主线程显式脱离，以传输数据。

Listing 8: csmodel/client.cc

```
#include <sys/socket.h>
#include <sys/types.h>
#include <netinet/in.h>
#include <netdb.h>
```

```c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>
#include <arpa/inet.h>
#include <time.h> //contains various functions for manipulating date and time

int main(int argc, char* argv[])
{
    struct sockaddr_in ipOfServer;
    ipOfServer.sin_family = AF_INET;
    ipOfServer.sin_port = htons(2680);
    ipOfServer.sin_addr.s_addr = inet_addr("127.0.0.1");

    char filename[80];
    sprintf(filename, "file_receive.txt");

    int MAXLINE = 1024;

    switch (argc)
    {
    case 1:
        printf("./client [file_size] [target_ip] [host_number]\n");
        return 0;
    case 4:
        sprintf(filename, "file_receive_%s.txt",argv[3]);
    case 3:
        ipOfServer.sin_addr.s_addr = inet_addr(argv[2]);
    case 2:
        MAXLINE = atoi(argv[1]);
    default:
        break;
    }

    clock_t start,finish;
    start = clock();

    int CreateSocket = 0,n = 0;
    char* dataReceived = (char*)malloc(MAXLINE*sizeof(char));
    memset(dataReceived, 1 ,sizeof(dataReceived));

    if((CreateSocket = socket(AF_INET, SOCK_STREAM, 0))< 0)
    {
        printf("Socket not created \n");
        return 1;
    }

    if(connect(CreateSocket, (struct sockaddr *)&ipOfServer, sizeof(
    ipOfServer))<0)
    {
        printf("Connection failed due to port and ip problems\n");
        return 1;
    }

    for(int i = 0; i < MAXLINE/1024; ++i)
```

```c
        read(CreateSocket, dataReceived + (i*1024) , 1024);

    FILE* f = fopen(filename,"wb");
    fwrite(dataReceived,sizeof(char),MAXLINE,f);
    fclose(f);

    finish = clock();

    FILE* rf = fopen("result_c.dat","a");
    double elapsed_time = (double)(finish-start)/CLOCKS_PER_SEC;
    if (argc==4)
        fprintf(rf,"%s\t%f\n",argv[3],elapsed_time);
    else fprintf(rf,"\t%f\n",elapsed_time);
    fclose(rf);
    fprintf(stdout, "%f\n", elapsed_time);

    if( n < 0)
    {
        printf("Standard input error \n");
    }

    return 0;
}
```

客户端会首先将数据存储在内存中，再写入文件。

D P2P 模型架构实现

Listing 9: p2pmodel/centralized.py

```python
# python centralized.py [hostnumber]

# To construct a centralized structure
# for P2P model.

import os, glob
from sys import argv
from time import sleep
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):
        net.hosts[i].cmdPrint(clientcmd,chunkSize,net.hosts[0].IP(),net.hosts
    [i].IP(),"" if i == hostnumber - 1 else "&")

    # CLI(net)

    results = []
    with open(resultfile,"r") as rf:
        resultlines = rf.read().splitlines()
        for i in range(1,len(resultlines)):
            results.append(float(resultlines[i].split('\t')[1]))
    avg = sum(results)/len(results) if not len(results) == 0 else -1
    print("Average: "+str(avg))
    print("Avaliable: " + str(len(results)) +  "/" + str(hostnumber-1) + "("
    + str(int(len(results)*100/(hostnumber-1))) + "%)")
    with open(avgresfile,"a") as af:
        af.write(str(hostnumber-1)+"\t"+str(avg)+"\n")

    net.stop()

if __name__=="__main__":
    lg.setLogLevel( 'info' )
    if len(argv)>=2:
        hostnumber = int(argv[1])
    else:
        hostnumber = int(input("Please input hostnumber:"))
    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 10: p2pmodel/server.py

```
# python server.py [chunkSize]

from socket import *
from sys import argv
import threading

serverPort = 2684
serverSocket = socket(AF_INET, SOCK_STREAM)
serverSocket.bind(('', serverPort))
serverSocket.listen(20)

with open("file.txt","rb") as f:
    content = f.read()

chunkSize = 1024
if len(argv)==2:
    chunkSize = int(argv[1])

class SendThread(threading.Thread):
    def __init__(self, connectionSocket, chunkNumber):
        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。主要有两种线程：ServiceThread 用于向其他对等方提供服务（或者会向服务器获取数据，如果最后没有完整传输，会强制从服务器获取对应的数据）；另一方面，FetchingThread 用于从其他对等方获取数据，并在最后合并到主线程。

由于数据上并没有线程间依赖，所以可以不加线程锁进行同步，编程上在这方面也有着对应的考虑。有时候对等方的服务线程可能还没有开启，会有失联的情况，这时会尝试一次重连，基本上都可以正确传输（对等方数目适当的情况下）。

Listing 11: p2pmodel/peer.py

```
# 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]

content = {}

def DownContent(chunkNumber):
    # Check if this peer has the chunk first.
    if not trackers[chunkNumber][2]:
        # Server use 2684 for communication.
```

```python
        clientPort = 2684
        clientSocket = socket(AF_INET, SOCK_STREAM)
        clientSocket.connect((serverName, clientPort))
        # send out the chunkNumber
        clientSocket.send(str(chunkNumber).encode())

        # expect to receive the chunk from the server.
        fileChunk = ""
        for i in range(chunkSize/1024):
            fileChunk = fileChunk + clientSocket.recv(1024).decode()
        # threadLock.acquire()
        content[chunkNumber] = fileChunk           # write to self content
        # threadLock.release()
        trackers[chunkNumber][2] = True            # declare to be useable
        print("Received Size from " + serverName + " :" + str(len(fileChunk))
    )

        clientSocket.close()
    else:
        fileChunk = content[chunkNumber]
    return fileChunk

class SendThread(threading.Thread):
    def __init__(self, connectionSocket, chunkNumber):
        threading.Thread.__init__(self)
        self.chunkNumber = chunkNumber
        self.connectionSocket = connectionSocket
    def run(self):
        # Detect the downloading first.
        downFileChunk = DownContent(self.chunkNumber)

        # Then send it to the peer
        self.connectionSocket.send(downFileChunk.encode())
        self.connectionSocket.close()

class ServiceThread(threading.Thread):
    def __init__(self):
        threading.Thread.__init__(self)
    def run(self):
        # peer use 2685 for communication.
        receivePeerPort = 2685
        receivePeerSocket = socket(AF_INET, SOCK_STREAM)
        receivePeerSocket.bind(('', receivePeerPort))
        receivePeerSocket.listen(20)
        receivePeerSocket.settimeout(10)

        while True:
            try:
                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()

# Start servicing.
serth = ServiceThread()
serth.start()

# Start fetching
fetth = FetchingThread()
fetth.start()
```

```python

# Wait for merge
fetth.join()

# Now, force to download the own chunk if it is not downloaded.
for tracker in trackers:
    if tracker[1] == hostName and tracker[2] == False:
        DownContent(tracker[0])

# should be done with all file chunks, unless error occurred.
with open(filename,"wb") as f:
    for i in range(len(content)):
        if i in content.keys():
            f.write(content[i])
        else:
            print("- " + hostname + " lost chunk " + str(i))

END_TIME = time.time()

elapsed_time = END_TIME-START_TIME
print(elapsed_time)

with open("result_py.dat","a") as rf:
    rf.write(hostname + "\t" + str(elapsed_time) + "\n")
```