

Socket Programming

计算机网络 CS339

李子龙 518070910095

2021 年 10 月 7 日

目录

1	C/S 模型	2
1.1	C/S 模型要求	2
1.2	C/S 模型测试	2
1.3	C/S 模型 Python 实现	6
1.4	C/S 模型 C 实现	8
2	P2P 模型	12
2.1	P2P 模型要求	12

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.

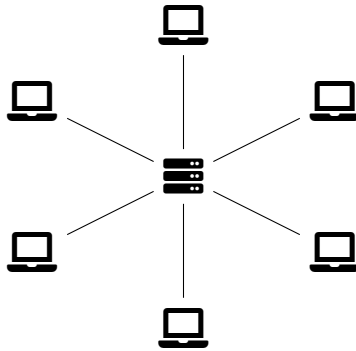
1 C/S 模型

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。

```
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 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`

```
1 # !/bin/bash
2
3 rm -rf file*
4 rm -rf result_py.dat
5 dd if=/dev/zero of=file.txt bs=1024 count=10240
6 python server.py &
7 sleep 1
8 python client.py 10485760
9 xxd file.txt > file.hex
10 xxd file_receive.txt > file_receive.hex
11 # diff file.txt file_send.txt
12 diff file.hex file_receive.hex
13 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。

```
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# 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: 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。

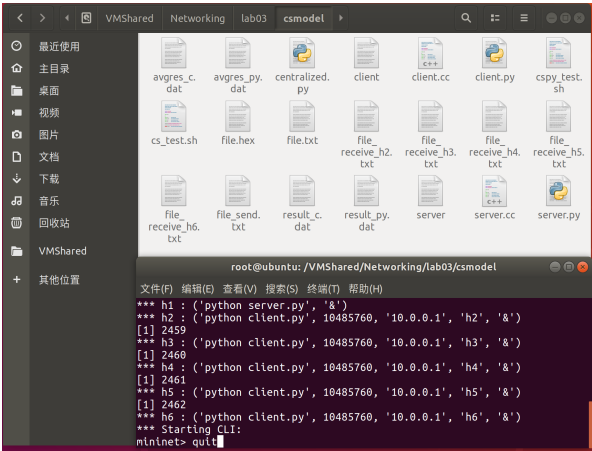


图 3: 运行多机测试

由图 4 可见，随着客户机的增长，传输时间也会显著变长。

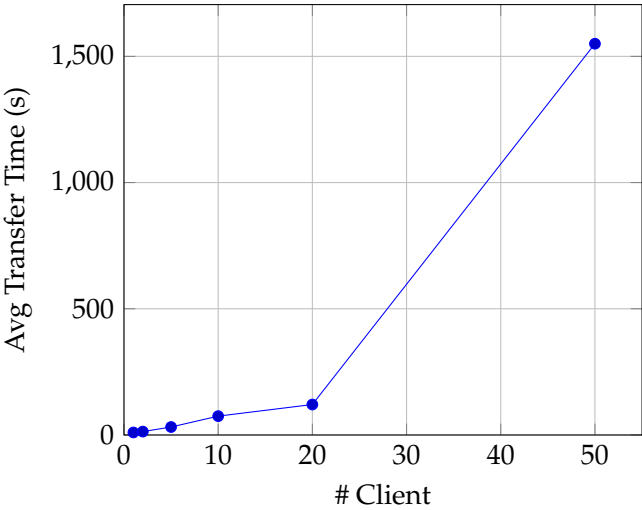


图 4: 不同的客户机数量下平均传输时间

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 2: `csmodel/centralized.py`

```

1  # To construct a centralized structure
2  # for C/S model.
3
4  import os, glob
5  from sys import argv
6  from time import sleep
7  from mininet.link import TCLink
8  from mininet.topo import LinearTopo, Topo
9  from mininet.net import Mininet
10 from mininet.log import lg, info
11 from mininet.util import dumpNodeConnections, irange, run
12 from mininet.cli import CLI
13
14 class CentralizedTopo(Topo):
15     "A centralized topo."
16
17     def build(self, hostnumber=2, **params):
18         # Central switch
19         switch = self.addSwitch('s1')
20         # Star hosts
21         hosts = [self.addHost('h%s' % h) for h in irange(1, hostnumber)]
22
23         # Star structure
24         for host in hosts:
25             self.addLink(switch, host)
26
27 def FileTransfer(hostnumber=2):
28     "Make file transfer between switch and hosts simultaneously."
29
30     servercmd = "python server.py"
31     clientcmd = "python client.py"
32     resultfile = "result_py.dat"
33     avgresfile = "avgres_py.dat"
34     if len(argv) >= 2:
35         if argv[1] == "c":
36             servercmd = "./server"
37             clientcmd = "./client"
38             resultfile = "result_c.dat"
39             avgresfile = "avgres_c.dat"
40
41     dirty = False
42     if len(argv) >= 3:
43         if argv[2] == "--dirty":
44             dirty = True
45
46     topo = CentralizedTopo(hostnumber)
47     # topo = LinearTopo(1, hostnumber) # The same.
48     net = Mininet(topo=topo, link=TCLink, autoStaticArp=False)
49     net.start()
50     # Test connectivity.
51     if not dirty:
52         net.pingAll()
53
54     # clean the files.
55     for file_receive in glob.glob("file_receive*"):

```

```

56         os.remove(file_receive)
57
58         fileSize = os.path.getsize("file.txt")
59
60         dumpNodeConnections(net.hosts)
61
62         # Place the server on h1.
63         print(net.hosts[0].cmdPrint(servercmd, "&"))
64
65         sleep(2)
66
67         # All other host request the file from h1.
68         for i in range(1, hostnumber):
69             if i == hostnumber - 1:
70                 print(net.hosts[i].cmdPrint(clientcmd, fileSize, net.hosts[0].IP(),
71 net.hosts[i].name))
72             else:
73                 net.hosts[i].cmdPrint(clientcmd, fileSize, net.hosts[0].IP(), net.
74 hosts[i].name, "&")
75
76         CLI(net)
77
78         # check the difference.
79         if not dirty:
80             for i in range(1, hostnumber):
81                 run("diff file.txt file_receive_"+net.hosts[i].name+".txt")
82
83         results = []
84         with open(resultfile, "r") as rf:
85             resultlines = rf.read().splitlines()
86             for i in range(1, len(resultlines)):
87                 results.append(float(resultlines[i].split('\t')[1]))
88         avg = sum(results)/len(results)
89         print("Average: "+str(avg))
90         print("Avaliable: " + str(len(results)) + "/" + str(hostnumber-1) + "("
91 + str(int(len(results)*100/(hostnumber-1))) + "%)")
92         with open(avgresfile, "a") as af:
93             af.write(str(hostnumber-1)+"\t"+str(avg)+"\n")
94
95         net.stop()
96
97     if __name__=="__main__":
98         lg.setLevel( 'info' )
99         hostnumber = input("Please input hostnumber:")
100         FileTransfer(int(hostnumber))

```

1.3 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 3: `csmodel/server.py`

```

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

```

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")

```

1.4 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 5: 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
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)
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 6: 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 5: [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
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 6: `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 }

```

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.