



# Laboratory Report of Computer Networks

## Lab I. Signal Expressing and Sampling Theorem

Name: Yutong LI 厉宇桐

No.: 517261910017

Date: 2020/10/16

Score: \_\_\_\_\_

SHANGHAI JIAOTONG UNIVERSITY  
SJTU-ParisTech Elite Institute of Technology

# Contents

<b>1</b>	<b>Introduction</b>	<b>3</b>
<b>2</b>	<b>Build instructions</b>	<b>3</b>
2.1	Requirement . . . . .	3
2.2	Build CXX executables . . . . .	3
2.3	Usage of CXX executables . . . . .	4
2.4	Usage of Python script . . . . .	5
<b>3</b>	<b>Client/Sever model</b>	<b>6</b>
<b>4</b>	<b>P2P model</b>	<b>7</b>
<b>5</b>	<b>Experiment</b>	<b>7</b>
5.1	Iperf test . . . . .	8
5.2	C/S model (simutaneously) . . . . .	8
5.3	P2P model (simutaneously) . . . . .	9
5.4	P2P model (in order) . . . . .	9
<b>6</b>	<b>Analyze of results</b>	<b>9</b>
6.1	Simutaneous launch . . . . .	9
6.2	Sequence launch . . . . .	10
6.3	The upload status of Node1 . . . . .	11
<b>7</b>	<b>Summary</b>	<b>12</b>

# 1 Introduction

In this experiment we implemented 2 simple file share applications using TCP Socket APIs. The applications are designed under the instruction of the following 2 models:

- C/S model: 1) Server listens to a given port; 2) Multiple clients request the same file from the server; 3) Each client save the file to its local directory.
- P2P model: 1) Each peer downloads part of the file from the server; 2) Peers distribute it to all the other peers. [1]

The programming language of applications is C++, and **Mininet** was used to run simulations.

# 2 Build instructions

The contents of submission is as follows:

```
| - SUBMISSION_ROOT
|   | - cxx
|   |   | - src
|   |       | - common # Definition of classes, tool functions
|   |       | - tcpClient # TCP client implementation (C/S)
|   |       | - tcpNode # TCP node implementation (P2P)
|   |       | - tcpServer # TCP server implementation (C/S)
|   |       | - tcpTracker # TCP tracker implementation (P2P)
|   |   - CMakeLists.txt
|   - mininet_run.py # Minet experiment
```

## 2.1 Requirement

- Linux Operating System (Ubuntu 18.04 LTS tested)
- CMake
- Clang that support C++11 standard
- Python3
- Mininet

## 2.2 Build CXX executables

Run the following script to build CXX executables:

```
1 cd $SUBMISSION_ROOT
2 mkdir build
3 cd build && cmake ../cxx && make
```

The executables are generated in the **build** directory

## 2.3 Usage of CXX executables

### 2.3.1 Usage of tcpServer and tcpClient

The **tcpServer** is launched this way:

```
1 ./tcpServer [Root path]
```

The **tcpClient** is launched this way:

```
1 ./tcpClient [Host IP] [Host Port] [FILEPATH (Remote)] [FILEPATH (Local)]
```

or this way:

```
1 ./tcpClient
```

**FILEPATH(Remote)** refers to the desired file's path on the server side. The path is relative to the server's **Rootpath**. **FILEPATH(Local)** refers to the local path where the file will be saved (must exist).

If no argument were passed to **tcpClient**, a command line interface will be started. The program will wait for input. It expect the user to provide the IP and port where a **tcpServer** is running. Then it connects to the server and list its directories. User can use following commands to interact with it:

- cd [Dir] # Change directory
- ls # List directory
- dl [Filename] # Download file

### 2.3.2 Usage of tcpServer and tcpClient

The **tcpTracker** is launched this way:

```
1 ./tcpTracker
```

The tracker listen to peers requests. It must be launched before **tcpNode**

The **tcpNode** is launched this way:

```
1 ./tcpNode [PORTDelta] [TRACKERHOSTNAME] [TRACKERPORT] [ROOTPATH] [FILE1] [FILE2]
```

`PORTDelta` represent the offset of Node's listen port from `NODEPORT` defined in `macro.h`, this is designed to better debug the program on single host. `TRACKERHOSTNAME` and `RACKERPORT` are IP and port where `tcpTracker` listen. `ROOTPATH` is the the root path of Node. `FILE*` are filenames separated by spaces, referring to the files the user want to sync

The `tcpNodes` can be launched at any sequence, but if none of the `tcpNodes` have desired file, nothing will be transmitted.

## 2.4 Usage of Python script

To run the experiment, simply execute

```
1 sudo python3 mininet_run.py
```

or

```
1 sudo python3 mininet_run.py --max_client 5 --min_client 2 --fs 10 --delay 3 --bw 10
```

The default value of max/min number of client is 5 and 1. The default value of the file size is 10MiB.

There are 7 stages:

- 1) The script build CXX executable.
- 2) The script delete possible old experiment output
- 3) The script prepare test folders, generate random download/sync target.
- 4) The script create Mininet topology.
- 5) The script run tests, redirect standard output to log files and wait enough time.
- 6) The script checks file integrity by running `md5sum`
- 7) The script read log files and plot results

A formula was used to estimate how long the script should wait before all client/node have finished their work:

$$t = \frac{3 \times 8 \times F \times (N + 1)}{B}$$

where  $F$  refers to file size in MiB,  $N$  refers to number of clients and  $B$  refers to Bandwidth.

Sometimes, the C/S model will fail to deliver file even under this time limit. In this case, a `md5sum` hash will detect this issue.

After the experiment, the file structure looks like:

```
|- SUBMISSION_ROOT
|   |- build
|   |   - tcpClient
|   |   - tcpNode
```

```

|   - tcpServer
|   - tcpTracker
|   - ...
|- cxx
|- Files
|   - File_XXM
|- Dst
|   - CS
|       |- Downloads
|           - Client1_FileXXM
|           - Client2_FileXXM
|           - ...
|       - Client1.log
|       - Client2.log
|       - ...
|- P2P
|   |- Node1
|       - File_XXM
|   |- Node2
|   |- ...
|       - Node1.log
|       - Node2.log
|       - ...
|- mininet_run.py

```

Details are explained in script comments.

### 3 Client/Sever model

The structure of Client/Server model is shown in Figure 1

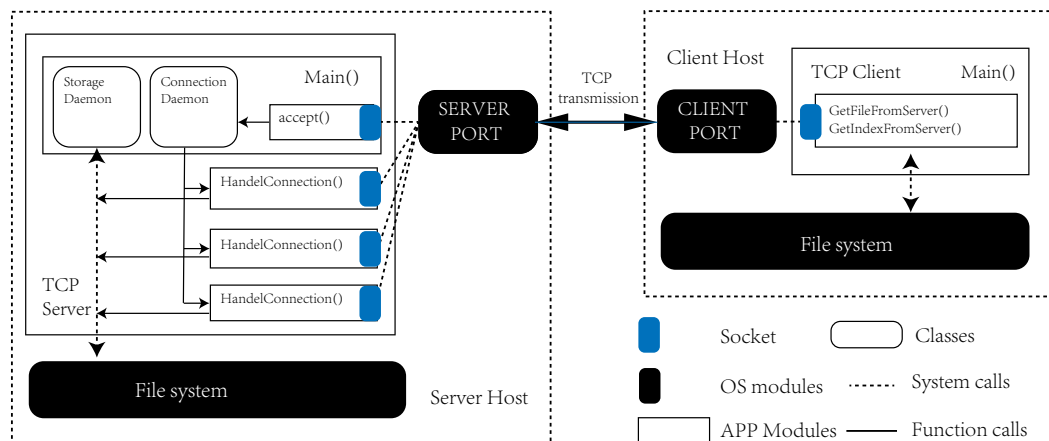


Figure 1 A structure design of C/S model

The server opens multiple threads for incoming clients. When sending a file, the server first send the size of the file, then send the file content in binary. The clients are single threaded. They talk to server by sending fixed-length strings (control messages). The length of these strings are defined in `macros.h`.

## 4 P2P model

The structure of P2P model is shown in Figure 2

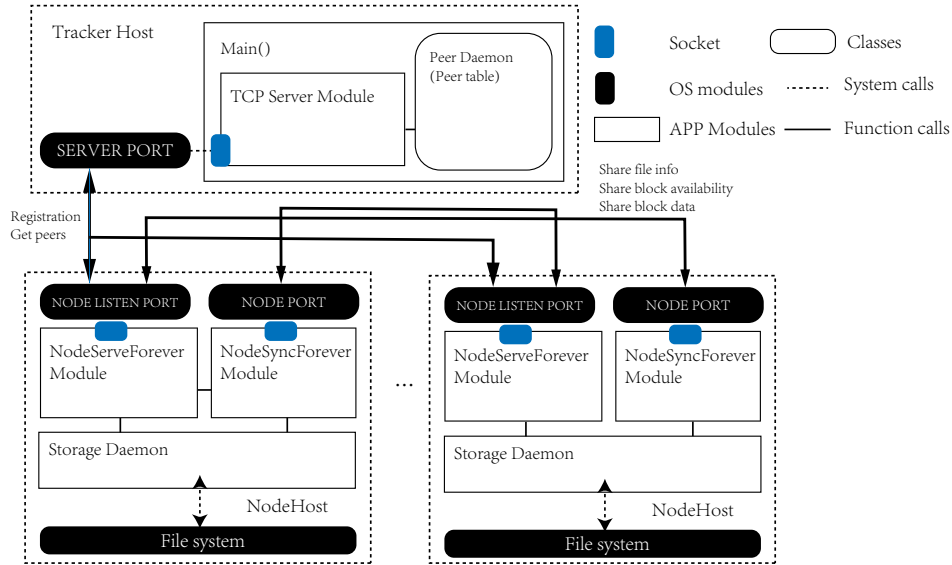


Figure 2 A structure design of P2P model

In our experiment, each node opens multiple threads for receiving and sending. The sending threads are managed in the same way as the `tcpServer`. The receiving threads are synced by mutex.

The client get peer information from tracker. It also register itself to the tracker by multiplexing its listen port.

## 5 Experiment

We run the experiment. An example of standard output is shown in Figure 3

Figure 3 Screenshot of standard output Number Client  $\in 2, 3, 4, 5$ 

## 5.1 Iperf test

We tested the connection speed between Mininet virtual hosts. The results are shown in Figure 3

## 5.2 C/S model (simultaneously)

We launch the server, then launch a certain number of clients.

```
1 hosts[0].cmd(' ./build/tcpServer ./Files/ > ./Dst/CS/Server.log&')
```



```

2     for i in range(1, num_client + 1):
3         hosts[i].cmd('./build/tcpClient 10.10.0.1 18889 ./File_{M} ...
           ./Dst/CS/Downloads/Client_{M}_File_{M} > ...
           ./Dst/CS/Client_{M}.log&'.format(file_size, i, file_size, i))

```

### 5.3 P2P model (simutaneously)

We launch the tracker, then launch a certain number of nodes immediately and simutaneously. The first node has the file.

```

1     hosts[0].cmd('./build/tcpTracker > ./Dst/P2P/Tracker.log&')
2     for i in range(1, num_client + 2):
3         cmd_string = './build/tcpNode 0 10.10.0.1 18888 ./Dst/P2P/Node{M}/ ...
           File_{M} > ./Dst/P2P/Node{M}.log&'.format(i, file_size, i)
4         hosts[i].cmd(cmd_string)

```

### 5.4 P2P model (in order)

We launch the tracker, then launch a certain number of nodes in order. After each lauch, the script wait for 3 seconds.

```

1     hosts[0].cmd('./build/tcpTracker > ./Dst/P2P/Tracker.log&')
2     for i in range(1, num_client + 2):
3         cmd_string = './build/tcpNode 0 10.10.0.1 18888 ./Dst/P2P/Node{M}/ ...
           File_{M} > ./Dst/P2P/Node{M}.log&'.format(i, file_size, i)
4         hosts[i].cmd(cmd_string)
5         time.sleep(time_delay)

```

## 6 Analyze of results

### 6.1 Simutaneous launch

We carried out a set of experiment with number of peers ranging from 1 to 15. The result is shown in the Figure 4

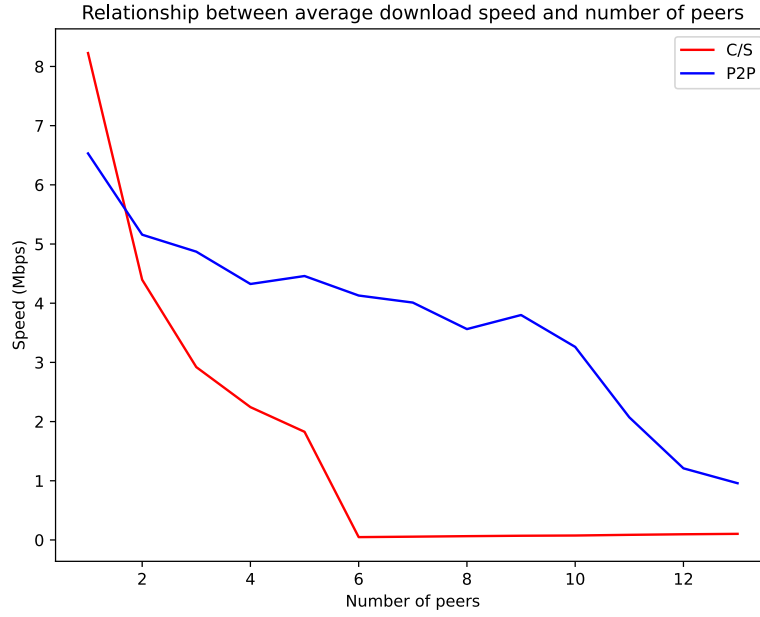


Figure 4 Relationship between average download speed and number of peers, file size = 10MiB

With a number of clients of 1, the P2P model is slightly slower than the C/S model because the client always tries to get more Peers from the tracker, and there are frequent exchanges between Peer and Peer about the latest state of the file, a transfer efficiency that could be improved if it had a more complex and refined control algorithm.

The average transmission rate of the network under the C/S model decreases substantially as the number of clients increases. According to the model, the decrease should be inversely proportional to the number of clients. The experimental data are roughly consistent with it. On the other hand, the average bandwidth of the network under the P2P model is significantly higher than the average speed under the C/S model.

As the number of clients continues to increase, the average speed of the network continues to decrease. A performance analysis revealed that the host CPU usage was close to 100%. We speculate that the network cannot increase the transmission rate indefinitely due to the processing performance of the host. In addition, nodes under the P2P model require additional computation and communication, and have more disk read overhead, all of which result in a decrease in the average speed of the P2P network.

## 6.2 Sequence launch

In the P2P model, if clients are launched simultaneously, clients will try to receive blocks of file from their peers. At the beginning of the network, only one peer has the complete file. As a result, the peers are forced to wait.

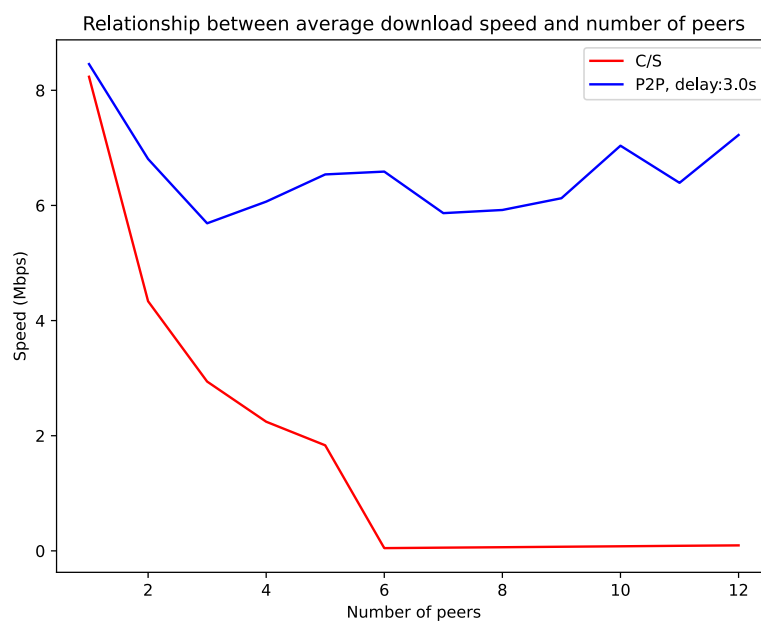
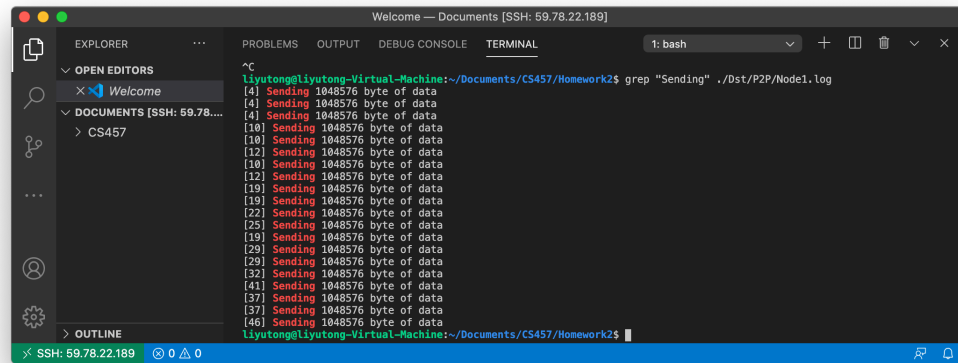


Figure 5 Relationship between average download speed and number of peers, file size = 10MiB, delay=3s

A solution is launching Nodes in a sequence. When we start the Nodes one by one, we can see that the average synchronization time across the network (the time it takes for a Node to complete synchronization) is reduced, resulting in an increase in the average download rate per node as shown in Figure 5. This is because later node gets the file block from its Peer immediately, which is in line with the P2P network transfer rules.

### 6.3 The upload status of Node1

In our experiment, Node1 is the peer who possess the file initially. We are interested in its log



```

Welcome — Documents [SSH: 59.78.22.189]
EXPLORER
  OPEN EDITORS
    Welcome
  DOCUMENTS [SSH: 59.78.22.189]
    CS457
  OUTLINE
    SSH: 59.78.22.189
  PROBLEMS
  OUTPUT
  DEBUG CONSOLE
  TERMINAL
    1: bash
    Liyutong@Liyutong-Virtual-Machine: ~/Documents/CS457/Homework2$ grep "Sending" ./Dst/P2P/Node1.log
    [4] Sending 1048576 byte of data
    [4] Sending 1048576 byte of data
    [4] Sending 1048576 byte of data
    [10] Sending 1048576 byte of data
    [10] Sending 1048576 byte of data
    [10] Sending 1048576 byte of data
    [12] Sending 1048576 byte of data
    [12] Sending 1048576 byte of data
    [19] Sending 1048576 byte of data
    [19] Sending 1048576 byte of data
    [22] Sending 1048576 byte of data
    [22] Sending 1048576 byte of data
    [25] Sending 1048576 byte of data
    [19] Sending 1048576 byte of data
    [29] Sending 1048576 byte of data
    [29] Sending 1048576 byte of data
    [32] Sending 1048576 byte of data
    [41] Sending 1048576 byte of data
    [37] Sending 1048576 byte of data
    [37] Sending 1048576 byte of data
    [46] Sending 1048576 byte of data
    Liyutong@Liyutong-Virtual-Machine: ~/Documents/CS457/Homework2$
  
```

Figure 6 Node1 upload detail, Number Client = 5

The Node1 only uploaded 20MiB of file. Its upload overhead is greatly reduced.

## 7 Summary

In a more congested network, P2P transfers can reduce the impact of server upload bandwidth limitations on the overall network transfer speed. But at the cost of more computation, random disk reads and writes, and additional communication overhead. In addition, P2P transport requires a well-designed set of protocols to load-balance the network's nodes. These are all challenges for designing P2P applications.

## References

- [1] <http://moodle.speit.sjtu.edu.cn/moodle/mod/assign/view.php?id=9425>, Oct. 2020.