# Lab3 Socket编程

目录

# 1 Client/Server模型

## 1.1 总述

在此Client/Server模型（下简称CS模型）中，我们会实现一个简易的服务端与客户端，客户端向服务器发送连接请求，服务器开始向客户端发送文件数据。

## 1.2 协议

在此简单模型中，没有设计控制协议，仅传输数据。

## 1.3 实现

本模型代码实现位于`./CS`目录下。

- 服务端网络通信部分

```
while True:
      self.lock.acquire()
      self.file.seek(sent_data)
      buffer = self.file.read(self.chunk_size)
      self.lock.release()
      if not buffer:
            client_socket.close()
            break
            client_socket.send(buffer)
            sent_data += self.chunk_size
```

- 客户端网络通信部分

```
s = socket(AF_INET, SOCK_STREAM)
s.connect(addr)
while True:
         buffer = s.recv(8192)
         if not buffer:
                  break
         f.write(buffer)
```

## 1.4 配置与运行

- 服务端

```
python3 server.py [-h] [-p PORT] [-f FILE]
```

  - `FILE`为被传输文件的路径，若此参数留空，服务端将生成10MB大小的随机文件。
  - `PORT`为服务器监听端口，默认值为52000。
  - 您可以使用`--help`或`-h`参数来查看帮助。

- 客户端

```
python3 client.py [-h] [-s SERVER] [-p PORT] [-o OUTPUT_PATH]
```

  - `SERVER`为服务器IP地址，留空将使用默认值`10.0.0.1`。
  - `PORT`为服务器端口，默认值为52000。
  - `OUTPUT_PATH`为下载文件保存位置，留空将保存在内存中。
  - 您可以使用`--help`或`-h`参数来查看帮助。

# 2 P2P模型

## 2.1 总述

在此P2P模型中，我们会实现P2P服务端与客户端。

服务端保存有完整的文件数据与所有连接的对等方信息，服务端接受并响应客户端请求。

客户端会从服务器获取在线对等方列表，同时从服务器与对等方下载数据；并且会开启一个小型服务端，向其他对等方提供数据下载服务。考虑到P2P网络中对等方的网络具有不稳定性，我们在客户端中加入了异常恢复功能，即使意外地与对等方失去连接，也不会导致文件下载失败。

在本模型中，我们并不预先分配`tracker file`，因为在实际的P2P网络中，除了特定的tracker服务器，假定其他任何节点的可用性都是十分荒唐的。本模型中，我们动态管理在线的peer列表，也动态地选择文件块下载源，如果需要，只需要稍微修改客户端，本模型也可以适用于有多个tracker服务器的P2P网络中。

## 2.2 协议

本P2P模型协议中，大部分种类的数据报文由报文头（HEAD）和正文（BODY）组成，也存在只有正文的纯数据报文。

一个普通的报文中，HEAD与BODY部分由`\r\n`分隔。HEAD部分由操作码（OP）与参数（args）组成，参数数量由操作类型决定，参数间使用空格分隔。一个普通报文格式如下：

```
OP [arg1 arg2 ...]
BODY
```

操作码如下：

- CHEL
  - 本操作码控制报文在客户端向服务器或对等方发起连接请求时由客户端发送。用于与服务器或对等方建立连接并请求加入P2P网络。
  - 报文头格式：

    ```
    CHEL  port
    ```

  - 参数：
    - `port`：客户端P2P上传服务端口，P2P网络中其他对等方可以通过该端口连接客户端并下载数据。
    - 本操作码未定义BODY部分。
  - 响应：

        - 如果无错误，服务器将返回SHEL控制报文，对等方将返回PHEL控制报文。

- SHEL
    - 本操作码控制报文在客户端向服务器发起初始化请求之后由服务端发送。用于告知客户端加入请求被接受，目标文件的基础信息、在线对等方列表会随此报文返回。
    - 报文头格式：

```
SHEL filesize chuncksize
```

    - 参数：
        - `filesize`：目标文件大小，以byte计。
        - `chuncksize`：文件分块大小，以byte计。
        - 本操作码BODY段包括了使用json编码的对等方列表，列表中每一项为`(IP,PORT)`形式的地址。
    - 响应：
        - 本报文无需响应，收到此报文后，客户端可向服务器发出其他请求。
- PHEL
    - 本操作码控制报文在对等方接受网络中其他对等方连接后返回。用于告知请求方连接被接受。
    - 报文头格式：

```
PHEL
```

    - 参数：
        - 本操作码无参数，且未定义BODY段。
    - 响应：
        - 本报文无需响应，收到此报文后，请求方可向接受方发出其他请求。
- CLREQ
    - 本操作码控制报文用于请求获取对等方已经下载的文件块列表。
    - 报文头格式：

```
CLREQ
```

    - 参数：
        - 本操作码无参数，且未定义BODY段。
    - 响应：
        - 对等方接受到本报文之后，返回`CLRSP`报文。
- CLRSP
    - 本操作码控制报文用于向发送请求的对等方发送本地已下载的文件块。
    - 报文头格式：

```
CLRSP
```

    - 参数：

        - 本操作码无参数。
        - 本操作码BODY段包括了使用json编码的文件块列表，列表中每一项为文件块编号
    - 响应：
        - 本报文无需响应，收到此报文后，请求方可向接受方发出其他请求。
- DREQ
    - 本操作码控制报文用于向对等方或者服务器请求数据块。
    - 报文头格式：

```
DREQ chunkid
```

    - 参数：
        - `chunkid`：数据块编号。
        - 本操作码未定义BODY部分。
    - 响应：
        - 被请求方发送`DATA`报文，返回被请求的数据内容。
- DATA
    - 本操作码控制报文用于向请求方发送被请求的数据块。
    - 报文头格式：

```
DATA offset size
```

    - 参数：
        - `offset`：相对于文件起始的偏移量。
        - `size`：数据块大小。
        - 本报文BODY部分为数据块数据，数据块过长时应分段传输。如果本报文无法放入全部数据，则应在本报文之后，连续传输无报文头的纯数据报文，直至数据块全部传输。
    - 响应：
        - 接收方无需响应本报文，且在接受到指定长度的数据之前不应向传输方发出其他报文。
- CHECK
    - 本操作码控制报文用于发起或回应校验请求。
    - 报文头格式：

```
CHECK checksum
```

    - 参数：
        - `checksum`：MD5校验码。
        - 本操作码未定义BODY部分。
    - 响应：
        - 对于服务端，若检验通过，则用本操作码返回相同的校验值，否者返回`ERROR`报文。
        - 对于客户端，收到此请求表明校验通过，无需响应。

- ERROR
    - 本操作码控制报文用于向连接方报告错误。
    - 报文头格式：

```
ERROR errorid
```

    - 参数：
        - `errorid`：错误类型。
        - BODY 部分包含补充错误信息（可选）。
    - 错误类型：
        - `UNKNOWN_COMMAND`：未知命令，报文头错误或在错误的状态发送了报文。
        - `OFFSET_OVERFLOW`：`DREQ`请求的数据块偏移不在合法范围内。
        - `CHUNK_UNDOWNLOADED`：`DREQ`请求的数据块还未下载。
        - `CHECKSUM_NOT_FIT`：校验码不匹配。

## 2.3 实现

P2P模型的代码位于`./P2P/`目录下，代码文件如下：

- `./P2P/utils.py`实现了报文解析、MD5计算、异常类等通用组件。
- `./P2P/server.py`实现了服务端。
- `./P2P/client.py`实现了客户端。

客户端与服务器的通信流程如下：

1. 客户端完成初始化，向服务器发送自己的监听端口。
2. 服务器返回文件信息、对等方列表。
3. 客户端随机选择一块文件块下载。
4. 重复第3步，直至下载完成。
5. 计算文件校验值，向服务器发送校验值。

对等方之间的通信流程如下：

1. 客户端获得一个对等方监听地址，向对等方发送自己的监听端口。
2. 对等方返回连接接受报文，同时新建一条与请求方监听端口的连接。
3. 客户端请求对等方可用的文件块。
4. 对等方返回已经下载的文件块列表。
5. 客户端在可用且自己未下载的文件块中选择一块，若无，则在一定时间后再次请求。
6. 客户端向对等方请求选中的数据块。
7. 对等方向请求者发送选中的数据块。
8. 重复3-7步，直至文件下载完成。

## 2.4 配置与运行

本项目代码可能需要运行在Python3.9或以上版本环境中。

### 2.4.1 配置文件

服务端与客户端的配置均位于`config.ini`中。

- 服务端

  服务端支持以下配置：

  - `port`：监听端口，默认为52000。
  - `ip`：监听地址，默认为`0.0.0.0`。
  - `file_path`：文件路径，留空则使用随机生成的文件。
  - `file_size`：文件大小，仅当`file_path`为空时生效。

- 客户端

  客户端支持以下配置：

  - `port`：监听端口，默认为52000。
  - `ip`：监听地址，默认为`0.0.0.0`。
  - `server_ip`：服务器ip。
  - `server_port`：服务器端口。
  - `file_path`：下载文件保存地址，留空则保存于内存中。注意：此参数生效时可能导致多个客户端并发读写，引发未知错误，因此不建议在有多个客户端同时运行时设置此参数。

### 2.4.2 运行参数

请注意：命令行参数生效优先级高于配置文件。

- 服务端

```
python3 server.py [-h] [-p PORT] [-f FILE] [--config CONFIG]
```

  - `FILE`为被传输文件的路径，若此参数留空，服务端将生成10MB大小的随机文件。
  - `PORT`为服务器监听端口，默认值为52000。
  - `CONFIG`为配置文件地址，默认为`config.ini`。
  - 您可以使用`--help`或`-h`参数来查看帮助。

- 客户端

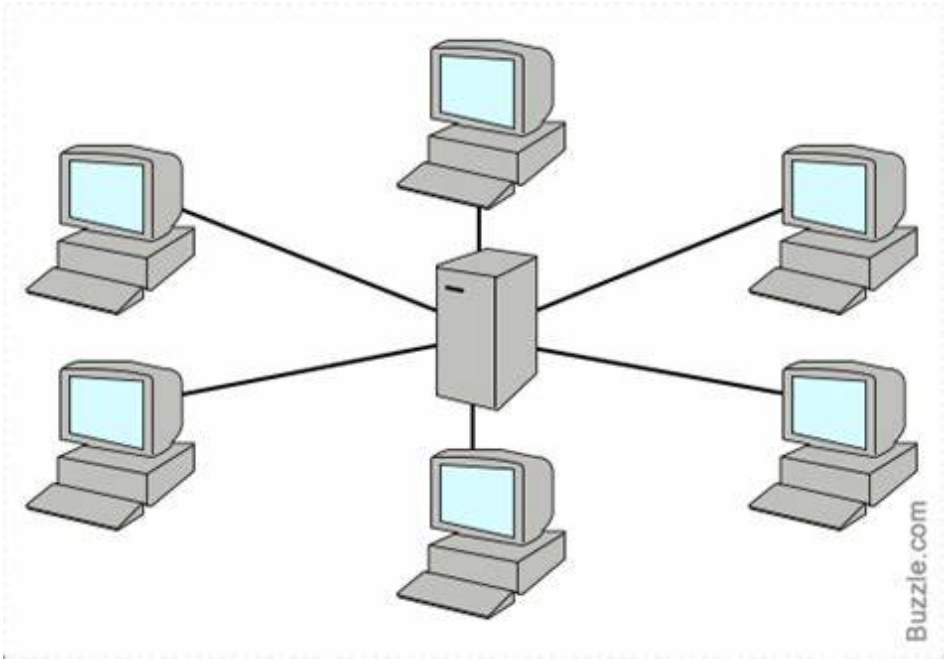
```
python3 client.py [-h] [-s SERVER] [-sp SERVER_PORT] [-p PORT] [-o
OUTPUT_PATH] [--config CONFIG]
```

  - `SERVER`为服务器IP地址，留空将使用默认值`10.0.0.1`。
  - `SERVER_PORT`为服务器端口，默认值为52000。
  - `PORT`为监听端口，默认为51000.
  - `OUTPUT_PATH`为下载文件保存位置，留空将保存在内存中。
  - `CONFIG`为配置文件地址，默认为`config.ini`。
  - 您可以使用`--help`或`-h`参数来查看帮助。

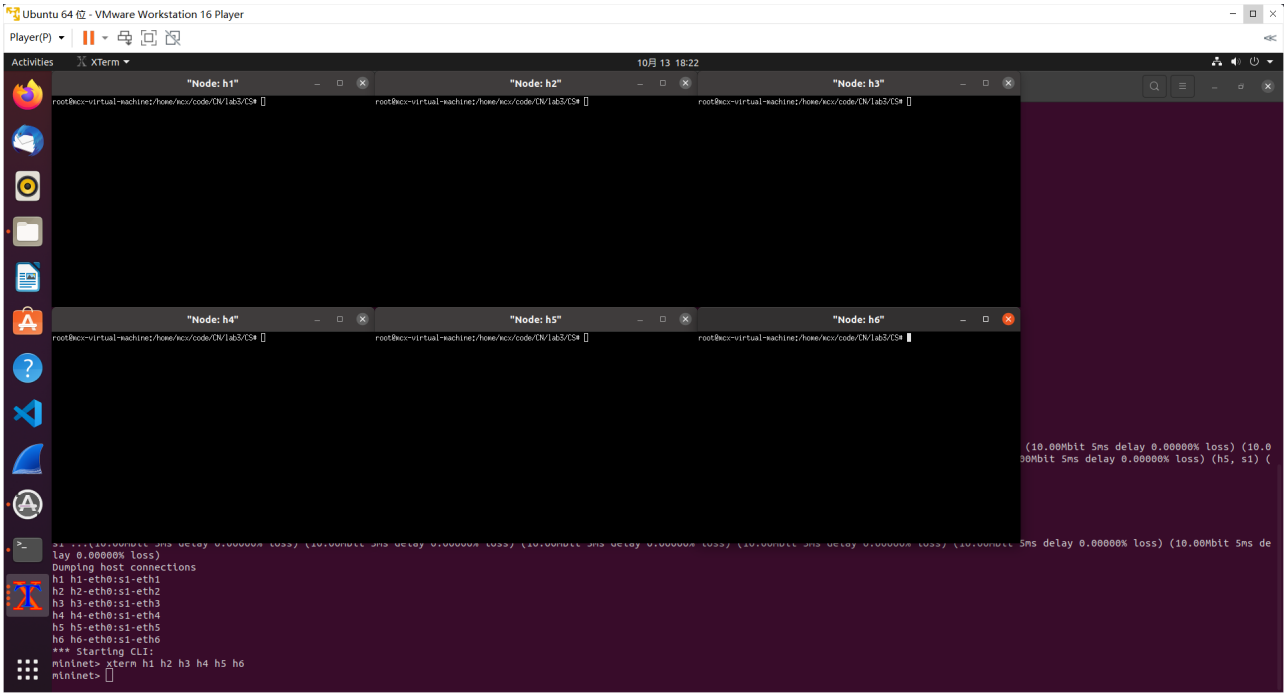
# 3 运行与分析

## 3.1 测试准备

运行`network.py`构造网络结构，网络结构如下图，图中每一条链路带宽为10Mbps，延时为5ms。




输入`xterm h1 h2 h3 h4 h5 h6`打开所有节点的终端。

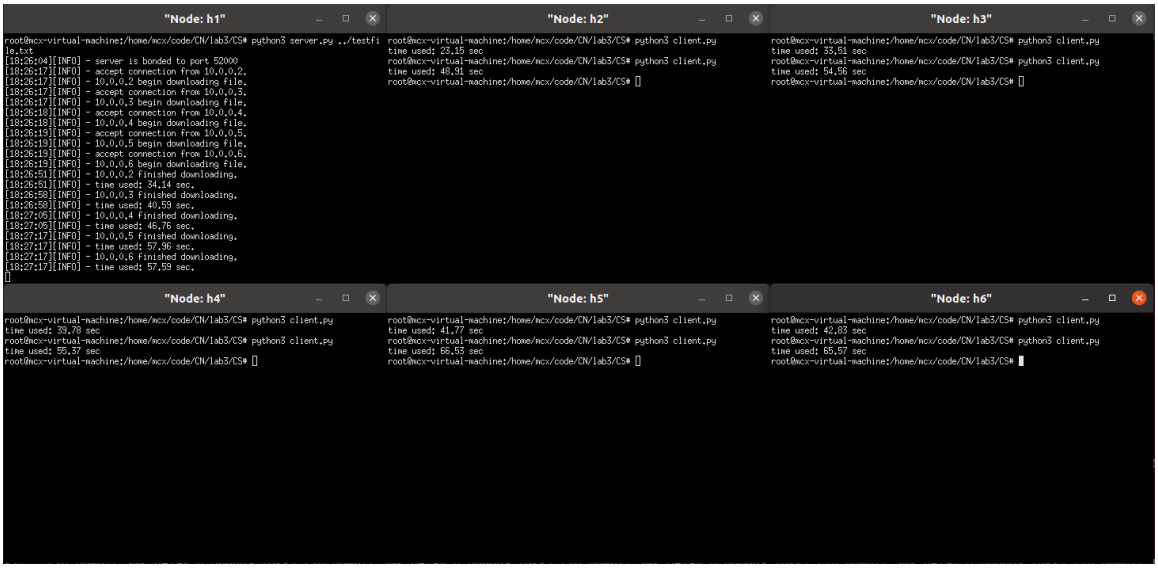



## 3.2 CS模型传输测试

- 随机生成10MB文件进行测试


  - 平均传输时间：36.2秒
  - 平均传输速度：282kB/s

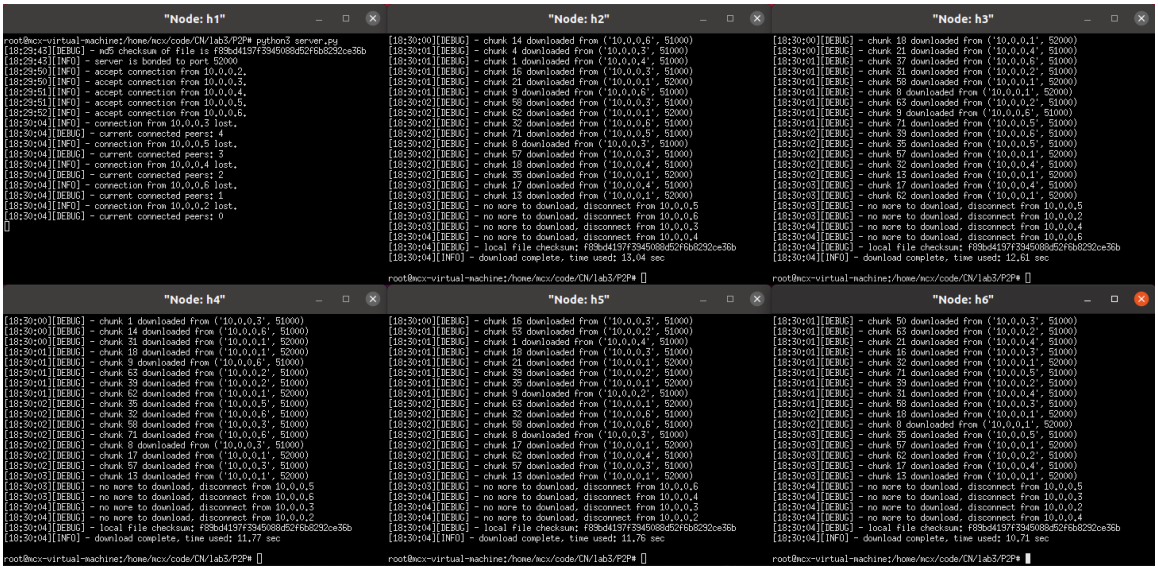
- 文本文件`testfile.txt`传输测试




  - 平均传输时间：58.2秒
  - 平均传输速度：264kB/s

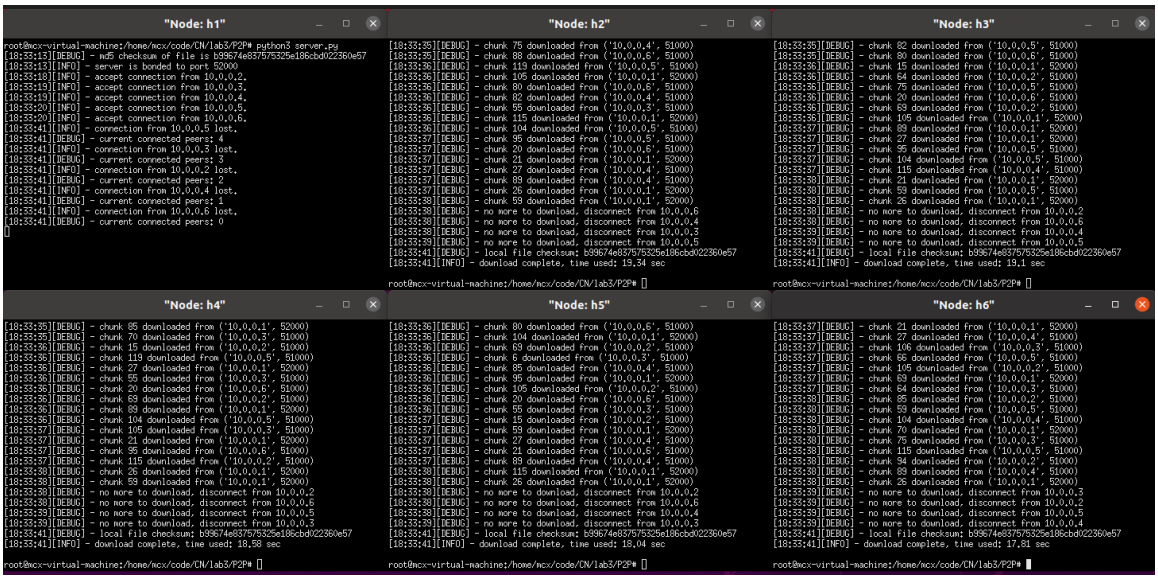
## 3.3 P2P模型传输测试

- 随机生成10MB文件进行测试




- 平均传输时间：12.0秒
- 平均传输速度：855kB/s

- 文本文件`testfile.txt`传输测试




- 平均传输时间：18.6秒
- 平均传输速度：827kB/s

## 3.4 测试结果

注：由于TCP缓存的存在，服务端显示的传输时长会小于客户端，本测试中以客户端传输时长为准。

将下载用户数从1逐渐增加到5，统计所有客户端下载完10MB文件所需要的平均时间，结果如下表所示：

| 下载用户数 | CS模型 | P2P模型 |
|---|---|---|
| 1 | 8.8s | 10.3s |
| 2 | 15.6s | 10.9s |
| 3 | 23.2s | 11.5s |
| 4 | 30.8s | 11.4s |
| 5 | 38.2s | 11.6s |

在有5个下载用户时，CS模型与P2P模型每个用户的平均下载速度如下表所示：

| | 随机文件(10M) | testfile.txt(~15M) |
|---|---|---|
| CS模型 | 282kB/s | 264kB/s |
| P2P模型 | 855kB/s | 827kB/s |

## 3.5 总结

CS模型受限于服务器的上传带宽，当用户总下载带宽大于服务器上传带宽时，服务器的上传带宽成为网络瓶颈，导致用户的带宽无法被充分利用。在我们的测试中，随着用户增加，下载速度呈反比例函数趋势下降。

P2P模型相比CS模型更为复杂，需要额外的控制协议，这带来了一些额外开销。但P2P模型允许用户之间分享文件，用户可以从其他对等方获得数据，更好地利用了每一位用户的带宽，提升了网络的整体传输速率。从测试数据中可以看出，随着用户量增加，每个用户的传输时间几乎不会增加。