

# 计算机网络期中项目实验报告

方向	计算机应用	班级	软工教务 2 班
学号	16340117	学号	16340115
学生	李江涛	学生	李佳铭
实验分工			
李江涛	服务端	李佳铭	客户端





## 一、 测试环境

1. 使用编程语言：python3
2. 实验环境：windows 10

## 二、 设计流程

### 1、文件结构：

#### ● 客户端：

 client.py	2018/11/29 13:00	PY 文件	2 KB
 ClientGet.py	2018/11/29 12:54	PY 文件	3 KB
 ClientSend.py	2018/11/29 12:26	PY 文件	4 KB
 Header.py	2018/11/28 15:38	PY 文件	1 KB





client.py: 客户端主程序

ClientGet.py: 储存下载文件使用的类与函数

ClientSend.py: 储存发送文件使用的类与函数

Header.py: 数据包头部结构

#### ● 服务端：

 Header.py	2018/11/26 19:20	Python 源文件	1 KB
 server.py	2018/11/23 22:10	Python 源文件	2 KB
 ServerGet.py	2018/11/27 15:34	Python 源文件	2 KB
 ServerSend.py	2018/11/27 15:34	Python 源文件	5 KB

Header.py: 数据包头部结构

server.py: 服务端主程序

ServerGet.py: 封装了服务端接收文件的类

ServerSend.py: 封装了服务端传输文件的类

## 2、命令使用:

如果 myserver 使用 myserver, 则会使用默认服务器地址发送

```
Please enter a command:
1. LFTP lsend myserver filename
2. LFTP lget myserver filename
3. exit
LFTP lget myserver 2.png
packet number: 614

Please enter a command:
1. LFTP lsend myserver filename
2. LFTP lget myserver filename
3. exit
LFTP lget 172.18.33.0 2.png
packet number: 614
```

## 3、UDP 使用:

- 服务端

```
IP_PORT = ('', 6666)
```

```
commandServerSocket = socket(AF_INET, SOCK_DGRAM)
```

```
commandServerSocket.bind(IP_PORT)
```

- 客户端

```
UDPSocket = socket(AF_INET, SOCK_DGRAM)
```

```
UDPSocket.bind(("", 6666))
```

#### 4、实现类似于 TCP 的 100%可靠传输：

- 利用 UDP 自带的检验和

首先，我们利用了 UDP 自带的差错检测机制，UDP 会实现检验和，如果数据有差错，就会丢弃该分组，因此只需检测丢包。

- SR 协议的使用

- 发送方

每次发送时将数据包缓存起来，以备重传，并将 UNACK 添加到窗口的最后。

```
fileContent = fileHandler.read(READSIZE)
```

```
head = Header(self.seq, self.rwnd).getHeader()
```

```
content = pack('ii', head['seq'], head['rwnd']) + fileContent
```

```
self.packets.append(content)
```

```
self.window.append(UNACK)
```

```
self.serverSocket.sendto(content, self.clientAddress)
```

```
self.seq += 1
```

当收到此时未确认的第一个包的序列号时，将这个包以后的所有已确认的包移出缓存，并滑动窗口位置。

```
self.window[0] = ACK
```

```
while len(self.window) > 0 and self.window[0] == ACK:
```

```
self.window.pop(0)
```

```
self.packets.pop(0)
```

```
self.sendBase += 1
```

当收到的包不是第一个未确认的包时，将该包标识为已确认。

```
self.window[receSeq - self.sendBase] = ACK
```

当接收 ACK 超时的时候，此时有两种情况，一种是该数据包丢包了，此时直接将发送方缓存中的第一个包发出去；另一种情况是接收方的接收窗口为 0，发送方已经没有数据包发送，因此也就没有 ACK 返回，此时发送方发送序列号为-1 的包，直到接收方的接收窗口大小大于 0 时才继续发送。

```
if len(self.packets) > 0:
```

```
    content = self.packets[0]
```

```
    self.serverSocket.sendto(content, self.clientAddress)
```

```
    self.ssthresh = int(self.cwnd) / 2
```

```
    self.cwnd = 1
```

```
    duplicateAck = 0
```

```
else:
```

```
    head = Header(-1, self.rwnd).getHeader()
```

```
    content = pack('ii', head['seq'], head['rwnd'])
```

```
    self.serverSocket.sendto(content, self.clientAddress)
```

### ○ 接收方

接收到包后返回 ack:

```
rwnd = windowSize - cnt + self.rcv_base - int(windowSize/5)
```

```
head = Header.Header(seq, rwnd).getHeader()
```

```
ack = pack('ii', head['seq'], head['rwnd'])
```

```
self.clientSocket.sendto(ack, serverAddress
```

滑动窗口的维护与读入：

接收到包后，判断缓存中有哪些包可以被交付给上层

while True:

```
    if self.window2[self.rcv_base % windowSize] == self.rcv_base:
```

```
        filehandler.write(self.window[self.rcv_base % windowSize])
```

```
        self.rcv_base += 1
```

```
    else:
```

```
        break
```

## 5、实现类似于 TCP 的流控制：

通过维护接收方剩余的缓存窗口大小，然后计算能发送的包的数量

- 发送方

接收 ack 时，更新接收方窗口大小

```
receSeq, self.rwnd = unpack('ii', data)
```

- 接收方

返回 ack 的时候，维护 rwnd 变量并发回：

```
rwnd = windowSize - cnt + self.rcv_base - int(windowSize/5)
```

```
head = Header.Header(seq, rwnd).getHeader()
```

```
ack = pack('ii', head['seq'], head['rwnd'])
```

```
self.clientSocket.sendto(ack, serverAddress)
```

## 6、实现类似于 TCP 的阻塞控制：

使用了 TCP 拥塞控制算法

- 慢启动

当 cwnd 小于阈值 ssthresh 时，每次收到一个新的 ack，都将 cwnd 加 1。

```
self.cwnd += 1
```

如果收到一个冗余的 ack，就将冗余计数加一；当冗余计数到达 3 的时候，就进入快速恢复阶段。

```
duplicateAck += 1
```

当超时的时候，重新进入慢启动状态。

```
self.ssthresh = int(self.cwnd) / 2
```

```
self.cwnd = 1
```

```
duplicateAck = 0
```

- 拥塞避免

当 cwnd 大于阈值 ssthresh 时，每次收到一个新的 ack，将 cwnd 加  $1/cwnd$ 。

```
if self.cwnd >= self.ssthresh:
```

```
    self.cwnd += 1 / int(self.cwnd)
```

如果收到一个冗余的 ack，就将冗余计数加一；当冗余计数到达 3 的时候，就进入快速恢复阶段。

```
duplicateAck += 1
```

当超时的时候，重新进入慢启动状态。

```
self.ssthresh = int(self.cwnd) / 2
```

```
self.cwnd = 1
```

```
duplicateAck = 0
```

- 快速恢复

当冗余 ack 到达三个时，进入快速恢复阶段。

```
if duplicateAck == 3:
```

```
    self.ssthresh = int(self.cwnd) / 2
```

```
    self.cwnd = self.ssthresh + 3
```

此时每次收到一个冗余 ack，都将 cwnd 加 1。

```
self.cwnd += 1
```

当收到一个新的 ack 时，进入拥塞避免状态。

```
if duplicateAck == 3:
```

```
    self.cwnd = self.ssthresh
```

```
    duplicateAck = 0
```

当超时的时候，重新进入慢启动状态。

```
self.ssthresh = int(self.cwnd) / 2
```

```
self.cwnd = 1
```

```
duplicateAck = 0
```

## 7、实现服务端同时支持多用户传输：

采用多线程的方式，服务端主程序运行在主线程中，可以持续接收不同用户的命令输入，每当有用户发送下载或上传的命令时，就新建一个线程，并为该线程分配新的端口号以及 socket。

```
if command[1] == 'lsend':
```

```
    print(str(clientAddress) + ' is uploading on port ' + str(p))
```

```
    threading.Thread(target=ServerGet(serverSocket, clientAddress,  
int(command[4])).getFile, args=(command[3], port, p)).start()
```

```
elif command[1] == 'lget':
```

```
    print(str(clientAddress) + ' is downloading on port ' + str(p))
```

```
    threading.Thread(target=ServerSend(serverSocket, clientAddress,  
int(command[4])).sendFile, args=(command[3], port, p)).start()
```

```
else:
```

```
    string = 'Error Input!'
```

```
    commandServerSocket.sendto(string.encode('utf-8'), clientAddress)
```

之所以使用多线程是为了让服务端与每个用户传输文件时不会互相阻塞，并且主程序可以持续接收其他用户输入的命令。而之所以使用不同的 socket，



首先是为了让服务端接收数据时不会将每个用户的数据混在一起，其次是有线程为其 socket 设置超时，不会影响到其他线程的正常使用。

### 三、 结果与测试

- 1G 文件传输

- 从服务端下载

```
server is listening on port 6666
LFTP lget myserver 超人: 钢铁之躯.rmvb 32
('172.18.35.20', 6666) is downloading on port 6667
download end!
The file size is: 1459.6731204986572 MB

Please enter a command:
1. LFTP lsend myserver filename
2. LFTP lget myserver filename
3. exit
LFTP lget myserver 超人: 钢铁之躯.rmvb
packet number: 765290
has received: 10000 packet
has received: 20000 packet
has received: 30000 packet
has received: 40000 packet
has received: 50000 packet
has received: 60000 packet
has received: 70000 packet
has received: 80000 packet
has received: 90000 packet
has received: 100000 packet
has received: 110000 packet
```

- 上传至服务端

```
Please enter a command:
1. LFTP lsend myserver filename
2. LFTP lget myserver filename
3. exit
LFTP lsend myserver 超人: 钢铁之躯.rmvb
fileSize: 1459.6731204986572 MB
end!
```

- 测试 100%传输，模拟随机丢包

使发送的每一个包有一定概率被随机丢掉

```
if random.random() > 0.1:
```

```
    self.serverSocket.sendto(content, self.clientAddress)
```

last packet 就是接收到的上一个包的序号，而 current packet 就是现在接受的包的序号。

可以看到，虽然有很多失序的包，但是最终还是传完了整个文件。

```
Please enter a command:
1. LFTP lsend myserver filename
2. LFTP lget myserver filename
3. exit
LFTP lget myserver 2.png
packet number: 614
disorder packet: last packet: 6 current packet: 8
disorder packet: last packet: 12 current packet: 14
disorder packet: last packet: 20 current packet: 7
disorder packet: last packet: 7 current packet: 13
disorder packet: last packet: 13 current packet: 22
disorder packet: last packet: 22 current packet: 21
disorder packet: last packet: 21 current packet: 23
disorder packet: last packet: 20 current packet: 22
disorder packet: last packet: 588 current packet: 591
disorder packet: last packet: 592 current packet: 589
disorder packet: last packet: 590 current packet: 593
disorder packet: last packet: 599 current packet: 601
disorder packet: last packet: 605 current packet: 600
disorder packet: last packet: 600 current packet: 607
disorder packet: last packet: 607 current packet: 606
disorder packet: last packet: 606 current packet: 608

Please enter a command:
1. LFTP lsend myserver filename
2. LFTP lget myserver filename
```

- 测试流控制，模拟减慢缓存读入速度

使缓存只有一定概率读入文件，降低缓存读入速度

```
if random.random() < debug:
```

```
while True:
```

```
    if self.window2[self.rcv_base % windowSize] == self.rcv_base:
```

```
        filehandler.write(self.window[self.rcv_base % windowSize])
```

```
        self.rcv_base += 1
```

```
    else:
```

```
        break
```

可以看到  $rwnd < 0$  时，它会产生超时并发送序号为-1 的报文询问返回结果。

```
rwnd: 13      rwnd: 0
rwnd: 12      TimeOut!
rwnd: 11      rwnd: 14
rwnd: 10      rwnd: 13
rwnd: 9       rwnd: 12
rwnd: 8       rwnd: 13
rwnd: 7       rwnd: 12
rwnd: 6       rwnd: 13
rwnd: 5       rwnd: 12
rwnd: 4       rwnd: 11
rwnd: 3       rwnd: 10
rwnd: 2       rwnd: 9
rwnd: 1       rwnd: 8
rwnd: 0       rwnd: 7
TimeOut!      download end!
rwnd: 0       The file size is: 1.1709604263305664 MB
TimeOut!
```

- 测试拥塞避免，模拟拥塞环境

一般来说，一次发包的数量越多，拥塞的可能性就越大。所以当一次发包的数量越多，使丢包的几率加大来模拟网络拥塞环境。

```
if random.random() > 0.01 * self.cwnd:
```

```
    self.serverSocket.sendto(content, self.clientAddress)
```

将  $cwnd$  初始化为 1， $ssthresh$  初始化为 8，一开始进入慢启动阶段。

```

cwnd: 1 ssthresh: 8 duplicateAck: 0
cwnd: 2 ssthresh: 8 duplicateAck: 0
cwnd: 3 ssthresh: 8 duplicateAck: 0
cwnd: 4 ssthresh: 8 duplicateAck: 0
cwnd: 5 ssthresh: 8 duplicateAck: 0
cwnd: 6 ssthresh: 8 duplicateAck: 0
cwnd: 7 ssthresh: 8 duplicateAck: 0
cwnd: 8 ssthresh: 8 duplicateAck: 0

```

当 cwnd 到达阈值时，进入拥塞避免阶段。

```

cwnd: 8 ssthresh: 8 duplicateAck: 0
cwnd: 8.125 ssthresh: 8 duplicateAck: 0
cwnd: 8.25 ssthresh: 8 duplicateAck: 0
cwnd: 8.375 ssthresh: 8 duplicateAck: 0
cwnd: 8.5 ssthresh: 8 duplicateAck: 0
cwnd: 8.625 ssthresh: 8 duplicateAck: 0
cwnd: 8.75 ssthresh: 8 duplicateAck: 0
cwnd: 8.875 ssthresh: 8 duplicateAck: 0
cwnd: 9.0 ssthresh: 8 duplicateAck: 0

```

当出现三个冗余 ack 时，进入快速恢复阶段。

```

cwnd: 9.111111111111111 ssthresh: 8 duplicateAck: 0
cwnd: 9.222222222222221 ssthresh: 8 duplicateAck: 0
cwnd: 9.222222222222221 ssthresh: 8 duplicateAck: 1
cwnd: 9.222222222222221 ssthresh: 8 duplicateAck: 2
cwnd: 7.5 ssthresh: 4.5 duplicateAck: 3
cwnd: 8.5 ssthresh: 4.5 duplicateAck: 3
cwnd: 9.5 ssthresh: 4.5 duplicateAck: 3
cwnd: 10.5 ssthresh: 4.5 duplicateAck: 3
cwnd: 11.5 ssthresh: 4.5 duplicateAck: 3
cwnd: 12.5 ssthresh: 4.5 duplicateAck: 3
cwnd: 13.5 ssthresh: 4.5 duplicateAck: 3
cwnd: 14.5 ssthresh: 4.5 duplicateAck: 3

```

当出现超时，重新进入慢启动阶段。

```

cwnd: 14.0 ssthresh: 3.0 duplicateAck: 3
cwnd: 15.0 ssthresh: 3.0 duplicateAck: 3
Timeout!
cwnd: 1 ssthresh: 8.0 duplicateAck: 0
cwnd: 2 ssthresh: 8.0 duplicateAck: 0
cwnd: 3 ssthresh: 8.0 duplicateAck: 0
cwnd: 4 ssthresh: 8.0 duplicateAck: 0
cwnd: 5 ssthresh: 8.0 duplicateAck: 0
cwnd: 6 ssthresh: 8.0 duplicateAck: 0

```

- 多用户测试，使用不同的电脑同时运行客户端，其中，主机

192.168.199.111 向服务端上传文件，主机 192.168.199.171 从服务端下载文件。

主机 192.168.199.111:

```
Please enter a command:
1. LFTP lsend myserver filename
2. LFTP lget myserver filename
3. exit
LFTP lsend 192.168.199.179 img.jpg
fileSize: 0.21481037139892578 MB
end!
```

主机 192.168.199.171:

```
Please enter a command:
1. LFTP lsend myserver filename
2. LFTP lget myserver filename
3. exit
LFTP lget 192.168.199.179 2.png
packet number: 614

Please enter a command:
1. LFTP lsend myserver filename
```

服务端:

```
server is listening on port 6666
('192.168.199.111', 6666) is uploading on port 6667
('192.168.199.171', 6666) is downloading on port 6668
('192.168.199.111', 6666) upload end!
('192.168.199.171', 6666) download end!
```

#### 四、 源代码

<https://github.com/smiletomisery/LFTP>