LFTP 项目报告

陈铭涛 16340024

陈慕远 16340025

LFTP 项目设计

项目简介

本次提交的 LFTP 项目使用 Python 进行编写，实现了基于 UDP 的文件传输，实现了与 TCP 类似的连接状态和流量控制与拥塞控制。程序分为客户端与服务端，在命令行下使用。服务端支持多个客户端同时进行文件的上传或下载。服务器启动时默认监听在 0.0.0.0:9999 下，也可通过命令行参数指定监听的地址与端口号。客户端使用如下的三个命令分别进行服务器上文件列表打印，文件上传与文件下载：

```
python3 client.py ls serveraddr
python3 client.py lsend serveraddr filename
python3 client.py lget serveraddr filename
```

其中，serveraddr 的格式为 ip:port 或 domain:port，支持使用域名作为服务器地址。文件上传时不能使用服务器上已存在的文件名。

程序执行流程

程序实现了一个 reliableUDP 模块，程序的应用部分调用该模块进行可靠的文件传输。

服务端的应用部分根据通信对象的 ip 地址和端口区分不同的连接所处的应

用层逻辑状态，根据客户端发送的请求做出返回文件列表，接收文件以及发送文件的操作。当接收 reliableUDP 调用关闭连接的接口后删除通信对象的状态。

客户端的应用部分在启动后首先对用户所指定的各个参数进行分析，然后调用 reliableUDP 发送握手请求到指定的服务端，三次挥手完成后构成连接，应用将发送用户指定的请求命令到服务器，根据服务器的响应执行请求或打印出错误信息。在命令执行完成后，客户端程序将调用 reliableUDP 的关闭连接发送第一次挥手，待四次挥手完成后退出程序。

reliableUDP 模块设计简介

reliableUDP 模块在 LFTP 应用进行文件数据传输时使用，实现了基于 UDP 的可靠数据传输。

reliableUDP （简称 rUDP）在 发送和接收的数据包的表头结构与 TCP 的表头结构基本相同，在表头中实际被模块使用的字段为：源端口，目标端口，序列号码，确认号码，ACK，SYN，FIN，窗口大小，数据偏移量，以及校验和。其余字段大部分默认置为 0。

rUDP 中包含了 server 与 client 主要两个部分，在通讯过程中由客户端发起构建连接握手与释放连接挥手，当连接构建完成后服务器与客户端都可以向对方发送信息，使用双方各自的 seq 号码标记发送数据包的序号以保证传输过程中的顺序可靠。

对每一个包，发送方会将不包含 checksum 的表头与数据合并计算出一个 2 字节的校验和作为简单的错误校验。校验和的计算对所有数据按 16 比特的组计算所有数据的 1 的补数和(1’ s complement sum)。接收方对接收到的数据包含

checksum 位使用相同的方法进行计算，若计算获得的 checksum 结果为全 1 则验证通过。在校验和的实现中，由于考虑到发送方使用的 socket 的 ip 地址与接收方所识别到的发送者地址不一定相同，因此没有加入实际 tcp 中计算 checksum 时使用的 pseudo-header。

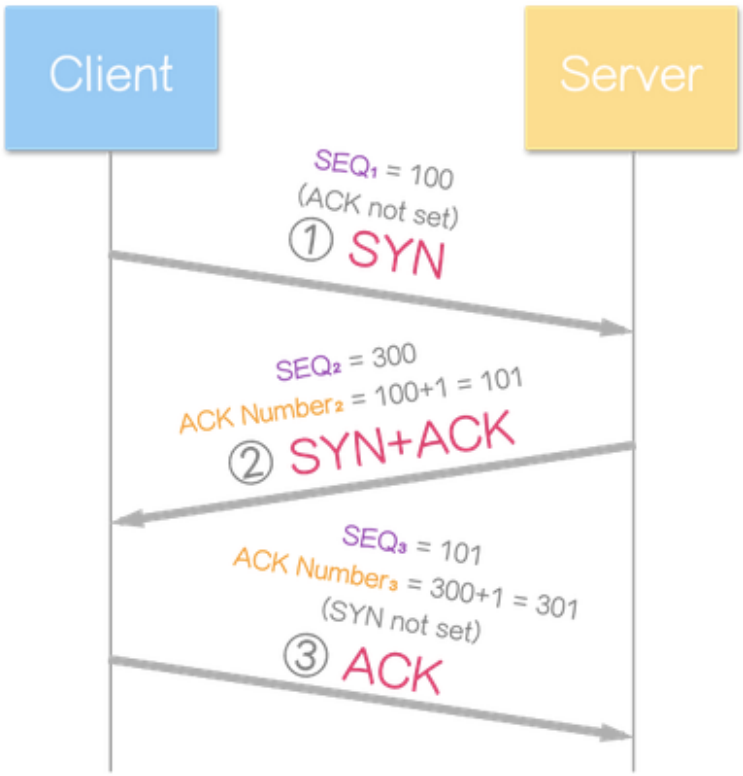
类似 TCP，对于一个连接服务端的状态分别有：CLOSED, LISTEN, SYN_REVD, ESTABLISHED, CLOSE_WAIT, LAST_ACK。客户端的状态有: CLOSED, SYN_SENT, ESTABLISHED, FIN_WAIT_1, FINWAIT_2, TIME_WAIT。状态在实现的 Python 代码中使用两个枚举类进行标识：

```
RecvStates = Enum('RecvStates', ('CLOSED', 'LISTEN',
'SYN_REVD', 'ESTABLISHED', 'CLOSE_WAIT', 'LAST_ACK'))

SendStates = Enum('SendStates', ('CLOSED', 'SYN_SENT',
'ESTABLISHED', 'FIN_WAIT_1', 'FIN_WAIT_2', 'TIME_WAIT'))
```

对于每一个连接，rUDP 客户端将发送第一个握手消息，SYN 位为 1，seq 序号为一个随机数，收到该握手消息后服务端将发送第二次握手消息，SYN 和 ACK 位为 1，ack 序号为收到的 seq 序号加 1，seq 序号为另一个随机数。收到第二次握手的消息后，客户端将发送第三次握手的消息，ACK 位为 1，ack 值为收到的 seq 序号加一。

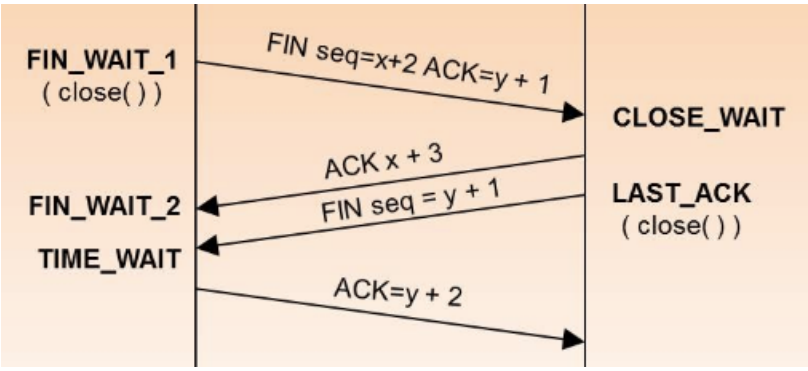
建立连接三次握手的过程如下：



（数值仅为说明，实际实现中 seq 序号为一个随机值）

建立连接后客户端与服务端进入 ESTABLISHED 状态，所有应用的数据传输在此状态下进行，双方第一次进行数据传输时的 seq 序号为其初始随机的 seq 序号下一，后续包的 seq 序号为该随机值加上累计传输的字节数。收到发送方传输的数据包后，若该数据包的 seq 值为接收方当前已 ack 到的数值，则返回的 ack 包的 ack 数值为此前的 ack 值加上收到的包的长度，并将数据放入接收窗口，否则返回的 ack 值不变。发送方对发送的每一个包绑定一个 timer，默认当数据包发出 1 秒后仍未收到大于其 seq 值的 ack 时进行重传。

在完成所有数据的传输后，客户端将发送第一次挥手信息，FIN 位为 1，服务端接收到后发送第二次和第三次的挥手信息，客户端接收后返回第四次挥手信息结束连接。大致图示如下：

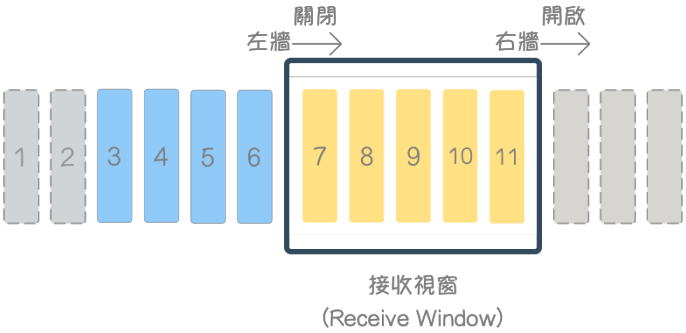


流量控制

TCP 流量控制（Flow Control）是为了避免高速传送端瘫痪低速接收端而实现的，在本应用中，流量控制由接收方维持的接收缓冲区实现。

由于发送者与接收者传输、读取的速率不相等，接收方需要将资料暂存在接收缓冲区，等待接收方应用层读取资料后，再从接收缓冲区中清除。接收缓冲区使用环形队列（circular queue）来实现，使空间有效率的复用。本应用中，每个数据包的大小为 5KB，而接收缓冲区的大小为 250KB，也就是说，接收缓冲区最多可以同时存放 50 个数据包。接收缓冲区中有两种空间：已接受并确认收到，应用层尚未读取的分组；空的缓冲区。

实际上，“环形”仅为一个概念上的说法，实现仍为线性结构，通过下标的转换来模拟实现一个环形队列。环形队列中还需维护一个接收窗口，它是用来计算接收缓冲区的机制，其概念如下图：

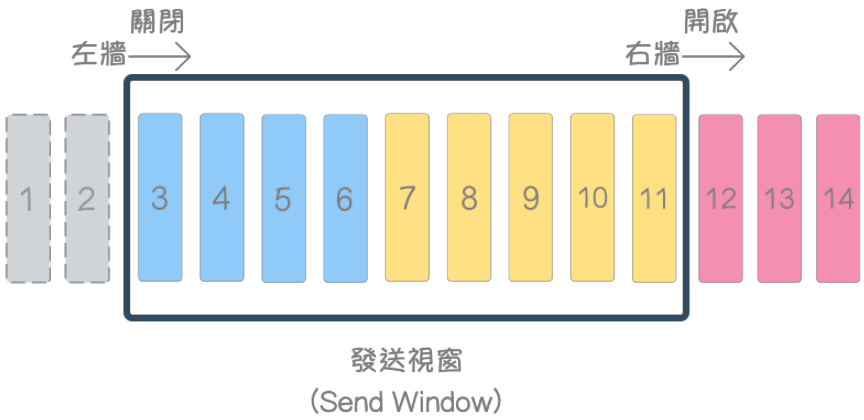


当接收方收到数据包，并回复确认后，接收窗口会关闭，即左墙往右移。

当接收方应用层读取资料后，资料会被清除，接收窗口会开启，即右墙往右移。

为了维护接收窗口，需要引入两个变量：lastByteRead 与 lastByteRcvd。其中 lastByteRcvd 表示的是最新收到的数据包，而 lastByteRead 表示的是最新读取的数据包。显然，lastByteRead 与上图中的左墙对应，而 lastByteRcvd 与蓝色块的最左端对应。除此之外，应用中还会维护一个 length 变量，每当添加了一个数据包后就增加它的值，这样接收视窗的值就可以表示为：(MAX_BUFFER_SIZE - length) / PACKET_SIZE，其中 MAX_BUFFER_SIZE 为 250KB，而 PACKET_SIZE 即为 5KB。每当 lastByteRead 与 lastByteRcvd 的值超过 PACKET_SIZE 的大小时，就将它重新置为 0，这样就达到了环形队列的效果。

发送方也会维持一个发送缓冲区，就像接收缓冲区中数据包被应用层读取后才会被清除一样，发送缓冲区中的数据包经发送并由接收方回复确认后才会被清除。与接收缓冲区不同的是，发送缓冲区中有三种空间：已被传送但未被确认的分组；准备传送的分组；空的缓冲区。其概念如下图：



蓝色部分为发送完毕、等待确认的分组，黄色部分为准备传送的分组，其他部分为可以存放分组的空缓冲区。发送缓冲区也存在一个发送视窗，但由于流量控制（暂不考虑拥塞控制），这个窗口是由接收端在回复确认时发送的接收窗口大小确定的，当接收窗口比较小时，说明接收方此时需要处理较多的数据，继续传送较多的数据可能导致瘫痪；而当接收窗口较大时，说明接收方比较空闲，可以尝试传送更多的数据。特别的，当接收窗口大小为 0 时，此时不应发送更多的数据，发送方停止发送。当接收方应用层处理数据后，接收缓冲区出现空闲了，此时由接收方通知发送方继续进行数据传送。

对应上图，本应用维护三个变量来实现发送缓冲区中数据包的填充、清除：`lastByteSent`，`lastByteAcked`，`lastByteReady`，分别对应最新发送的分组，最新被确认的分组，以及最新能够被发送的分组。

拥塞控制

TCP 流量控制是为了避免高速传送端瘫痪低速接收端而实现的，而 TCP 拥塞控制是用于避免高速传送端拥塞网络而实现的。TCP 拥塞控制的实现主要体现在发送方。

为了实现拥塞控制，最重要的是为发送缓冲区维护两个变量：拥塞窗口 `cwnd` 与慢启动门槛（`ssthresh`）。

在流量控制中提到，TCP 的发送窗口大小由接收端返回的接收窗口决定，实际上这是不准确的，真正的发送窗口大小为接收窗口大小与拥塞窗口大小间的最小值。确定初始拥塞窗口大小后，首先进入慢启动。慢启动在最初阶段发送的数据包很少，但每当一个数据包被确认后，拥塞窗口就会扩大一个数据包的大小，

这样每当一个发送窗口中的数据包都被发送并确认后，拥塞窗口就扩大了一倍。也就是说，慢启动阶段拥塞窗口的大小是指数成长的。

显然，慢启动不能无止尽的增加拥塞窗口的大小，所以这里设立一个慢启动门槛，当拥塞窗口到达此门槛时，就进入拥塞避免状态，在拥塞状态下，并非每个数据包被确认都会增加拥塞窗口，而是当一个窗口内的数据包都被发送后才增加拥塞窗口，这样就能够将窗口控制为线性增长。

在拥塞避免阶段，如果检测到拥塞（数据包发送超时等），则将慢启动门槛设置为此时的拥塞窗口的一半，拥塞窗口设置为 1，在当前超时数据包被确认后进入新的慢启动阶段。

注意到，当接收方接收窗口小于拥塞窗口时，说明接收端接收缓冲区不足；当接收端接收窗口大于拥塞窗口时，表示网络此时处于拥塞状态。

应用模块

应用通过继承 rUDP 下的 application 子模块中的 app 抽象类 来使用 rUDP。继承后的派生类中应当包含一个 rUDP 的客户端或服务端对象，在构造 rUDP 对象时将应用自身对象传入供 rUDP 进行调用。当 rUDP 连接接收到数据消息时，将把数据存入接收窗口并调用 notify_process_data 通知应用处理接收的数据。应用需要实现 process_data 方法并在方法中调用 rUDP 的 consume_rcv_buffer 方法获取接收数据进行处理。当 rUDP 接收到新的 ack 数据后将调用 notify_next_move 通知应用进行下一步操作（传输文件的下一部分，关闭连接等）。 在应用需要执行的操作结束后，调用 rUDP 的 finish_conn 方法进行挥手结束连接。

日志模块

rUDP 下包含 logger 文件，在程序执行中应用与 rUDP 均调用该文件进行调试信息日志的输出，输出日志将存在程序执行目录下的 lftp.log 文件下。

应用部分设计简介

LFTP 的应用部分分为 server 和 client，分别的使用方法如下：

1. 服务端：

```
python3 server.py [-h] [-p PORT] [-a ADDR] [-d DATADIR]
```

接收的各参数如下：

- -h 打印出程序参数帮助信息
- -p 程序监听的端口号
- -a 程序监听的 ip 地址
- -d 程序服务文件的根目录，客户端进行上传或下载时将对该目录下的文件进行操作，默认为程序的执行目录。

2. 客户端

```
python3 client.py [-h] command ServerAddr [filename]
```

各参数介绍如下：

- -h 打印出程序参数帮助信息
- command 程序指令，支持的指令为列出文件 ls，下载文件 lget，上传文件 lsend。
- ServerAddr 目标服务器地址，需指定 ip 或域名地址加端口号
- Filename 当操作为上传或下载时需指定该参数，表示需要上传或下载的文件名

程序启动后，将首先使用 argparser 对参数进行处理，若参数合法，服务端将调用 rUDP 开始对指定的地址进行监听。客户端将启动 rUDP 对目标服务器进行连接的建立。建立后将发送对应的指令至服务器（客户端与服务器应用层交互的指令可参考 LFTP Document 文件）。服务器根据其实际数据目录下的情况返回对客户端指令的响应。若操作为上传或下载且服务器响应操作有效，则发送文件一方将首先使用一个数据包发送文件的大小，然后进行文件数据的传输。

在连接建立后的传输过程中，所有非 ack 的数据包大小均设为 5k，每个包数据部分的前 4 个字节代表这个包的实际长度，对于长度不足 5116 字节的发送数据，应用部分进行发送前会对其末尾部分填充全 0 的 padding。

程序文档

对程序的各类与方法的介绍可参考 LFTP 参考文档 文件。

程序测试与截图

测试环境

测试环境为分别位于日本和中国香港的两台服务器，两服务器之间的连接时延如下：

```
➜  test git:(master) ✗ ping jp.chenmt.science
PING jp.chenmt.science (89.31.126.185) 56(84) bytes of data.
64 bytes from 89.31.126.185: icmp_seq=1 ttl=48 time=52.3 ms
64 bytes from 89.31.126.185: icmp_seq=2 ttl=48 time=52.7 ms
64 bytes from 89.31.126.185: icmp_seq=3 ttl=48 time=53.3 ms
64 bytes from 89.31.126.185: icmp_seq=4 ttl=48 time=52.8 ms
64 bytes from 89.31.126.185: icmp_seq=5 ttl=48 time=60.0 ms
64 bytes from 89.31.126.185: icmp_seq=6 ttl=48 time=52.3 ms
64 bytes from 89.31.126.185: icmp_seq=7 ttl=48 time=53.5 ms
64 bytes from 89.31.126.185: icmp_seq=8 ttl=48 time=67.0 ms
64 bytes from 89.31.126.185: icmp_seq=9 ttl=48 time=54.6 ms
64 bytes from 89.31.126.185: icmp_seq=10 ttl=48 time=52.4 ms
64 bytes from 89.31.126.185: icmp_seq=11 ttl=48 time=53.3 ms
64 bytes from 89.31.126.185: icmp_seq=12 ttl=48 time=52.2 ms
64 bytes from 89.31.126.185: icmp_seq=13 ttl=48 time=53.4 ms
64 bytes from 89.31.126.185: icmp_seq=14 ttl=48 time=55.9 ms
```

（香港至日本）

```
3. root@ONEVPS180708020917: ~/lftpy/test (ssh)
➜  test git:(master) ✗ ping wick.chenmt.science
PING wick.chenmt.science (103.118.40.34) 56(84) bytes of data.
64 bytes from 103.118.40.34: icmp_seq=1 ttl=55 time=52.6 ms
64 bytes from 103.118.40.34: icmp_seq=2 ttl=55 time=53.8 ms
64 bytes from 103.118.40.34: icmp_seq=3 ttl=55 time=53.4 ms
64 bytes from 103.118.40.34: icmp_seq=4 ttl=55 time=52.3 ms
64 bytes from 103.118.40.34: icmp_seq=5 ttl=55 time=52.7 ms
64 bytes from 103.118.40.34: icmp_seq=6 ttl=55 time=65.0 ms
64 bytes from 103.118.40.34: icmp_seq=7 ttl=55 time=63.1 ms
64 bytes from 103.118.40.34: icmp_seq=8 ttl=55 time=52.3 ms
64 bytes from 103.118.40.34: icmp_seq=9 ttl=55 time=52.3 ms
64 bytes from 103.118.40.34: icmp_seq=10 ttl=55 time=52.4 ms
64 bytes from 103.118.40.34: icmp_seq=11 ttl=55 time=54.3 ms
64 bytes from 103.118.40.34: icmp_seq=12 ttl=55 time=53.1 ms
64 bytes from 103.118.40.34: icmp_seq=13 ttl=55 time=56.3 ms
64 bytes from 103.118.40.34: icmp_seq=14 ttl=55 time=52.4 ms
64 bytes from 103.118.40.34: icmp_seq=15 ttl=55 time=54.9 ms
64 bytes from 103.118.40.34: icmp_seq=16 ttl=55 time=52.8 ms
64 bytes from 103.118.40.34: icmp_seq=17 ttl=55 time=52.3 ms
64 bytes from 103.118.40.34: icmp_seq=18 ttl=55 time=53.7 ms
^C
--- wick.chenmt.science ping statistics ---
18 packets transmitted, 18 received, 0% packet loss, time 34573ms
rtt min/avg/max/mdev = 52.337/54.483/65.020/3.572 ms
➜  test git:(master) ✗
```

（日本至香港）

使用 iperf 工具测得两服务器间 udp 带宽大约为 9.4Mbits/s:

```
➜ test git:(master) ✗ iperf -s -u -i 1 -p 443 -l 1380
------------------------------------------------------------
Server listening on UDP port 443
Receiving 1380 byte datagrams
UDP buffer size:  208 KByte (default)
------------------------------------------------------------
[  3] local 89.31.126.185 port 443 connected with 103.118.40.34 port 53343
[ ID] Interval       Transfer     Bandwidth        Jitter   Lost/Total Datagrams
[  3]  0.0- 1.0 sec  1.12 MBytes  9.37 Mbits/sec   0.834 ms    0/  849 (0%)
[  3]  1.0- 2.0 sec  1.12 MBytes  9.41 Mbits/sec   0.981 ms    0/  852 (0%)
[  3]  2.0- 3.0 sec  1.11 MBytes  9.33 Mbits/sec   1.103 ms    0/  845 (0%)
[  3]  3.0- 4.0 sec  1.13 MBytes  9.44 Mbits/sec   0.648 ms    0/  855 (0%)
[  3]  4.0- 5.0 sec  1.12 MBytes  9.37 Mbits/sec   1.138 ms   26/  849 (3.1%)
[  3]  4.0- 5.0 sec  26 datagrams received out-of-order
[  3]  5.0- 6.0 sec  1.12 MBytes  9.41 Mbits/sec   1.217 ms    0/  852 (0%)
[  3]  6.0- 7.0 sec  1.11 MBytes  9.35 Mbits/sec   1.697 ms    0/  847 (0%)
[  3]  7.0- 8.0 sec  1.12 MBytes  9.41 Mbits/sec   1.703 ms    0/  852 (0%)
[  3]  8.0- 9.0 sec  1.12 MBytes  9.40 Mbits/sec   0.808 ms    0/  851 (0%)
[  3]  9.0-10.0 sec  1.11 MBytes  9.33 Mbits/sec   1.708 ms    0/  845 (0%)
[  3]  0.0-10.0 sec  11.2 MBytes  9.37 Mbits/sec   2.459 ms   26/ 8504 (0.31%)
[  3]  0.0-10.0 sec  27 datagrams received out-of-order
```

使用香港服务器作为服务端进行测试，使用的测试传输的文件为一个大小为

10M 的文件：

```
➜ test git:(master) ✗ la
total 79M
-rw-r--r-- 1 root root  10M Dec  4 20:49 10mb.bin
-rw-r--r-- 1 root root 970K Dec  4 10:02 2mb.bin
-rw-r--r-- 1 root root 2.9M Dec  4 10:59 3mb.bin
-rw-r--r-- 1 root root  42K Dec  4 01:07 aa
-rw-r--r-- 1 root root 1023 Dec  3 23:21 command-doc.md
-rw-r--r-- 1 root root  52M Dec  4 20:55 lftp.log
-rw-r--r-- 1 root root  14M Dec  4 01:13 ll
```

下载

服务端启动应用：

```
➜ test git:(master) ✗ python3 ../src/server.py
The address to listen on is 0.0.0.0:9999
```

客户端列出服务端文件列表：

```
➜ test git:(master) ✗ python3 ../src/client.py ls wick.chenmt.science:9999
target server is 103.118.40.34, port: 9999
10mb.bin lftp.log aa 2mb.bin command-doc.md 3mb.bin ll

➜ test git:(master) ✗
```

客户端启动下载：

```
→ test git:(master) ✗ time python3 ../src/client.py lget wick.chenmt.science:99
99 10mb.bin
target server is 103.118.40.34, port: 9999
Receiving the file now ..., size: 10475520
Downloaded 7.08146%.
```

完成下载查看文件 md5：

```
→ test git:(master) ✗ time python3 ../src/client.py lget wick.chenmt.science:99
99 10mb.bin
target server is 103.118.40.34, port: 9999
Receiving the file now ..., size: 10475520
File download completed
python3 ../src/client.py lget wick.chenmt.science:9999 10mb.bin  14.55s user 1.0
8s system 66% cpu 23.583 total
→ test git:(master) ✗ md5 10mb.bin
zsh: command not found: md5
→ test git:(master) ✗ md5sum 10mb.bin
ac899f6c5b0069c86d78f11ffa2e2e3f  10mb.bin
→ test git:(master) ✗
```

由计时，算得速率为 3.39Mbits/s

查看服务器上文件 md5 值，验证文件传输完整：

```
2. root@CubeCloud-2018411230: ~/lftpy/test (ssh)
→ test git:(master) ✗ md5sum 10mb.bin
ac899f6c5b0069c86d78f11ffa2e2e3f  10mb.bin
→ test git:(master) ✗
```

上传

启动客户端：

```
→ test git:(master) ✗ time python3 ../src/client.py lsend wick.chenmt.science:9
999 10mb.bin
target server is 103.118.40.34, port: 9999
Sending the file now...
File upload completed
python3 ../src/client.py lsend wick.chenmt.science:9999 10mb.bin  17.76s user 1.
31s system 29% cpu 1:04.70 total
→ test git:(master) ✗
```

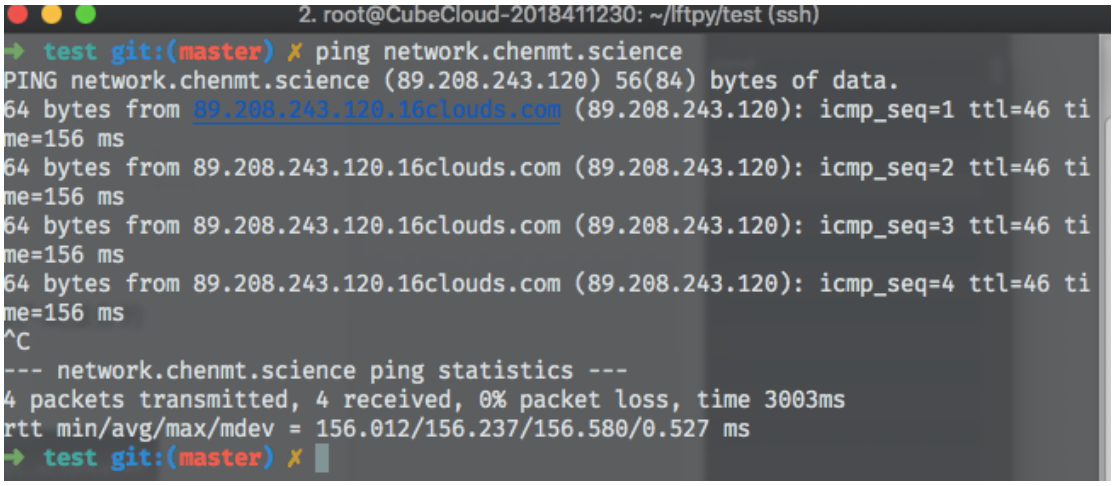
速率为 1.23Mbits/s

服务端接收文件 md5 正确：

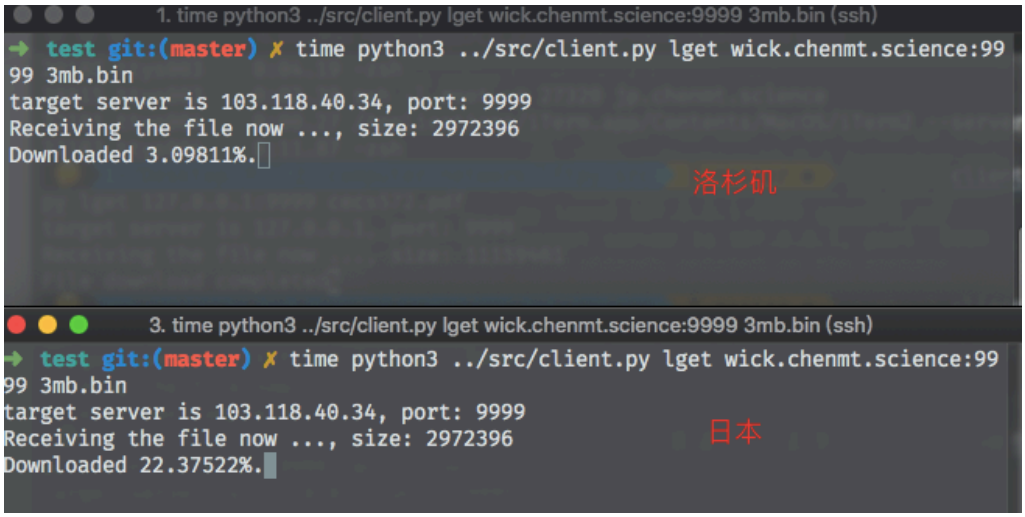
```
→ test git:(master) ✗ md5sum 10mb.bin
ac899f6c5b0069c86d78f11ffa2e2e3f  10mb.bin
```

多客户端测试

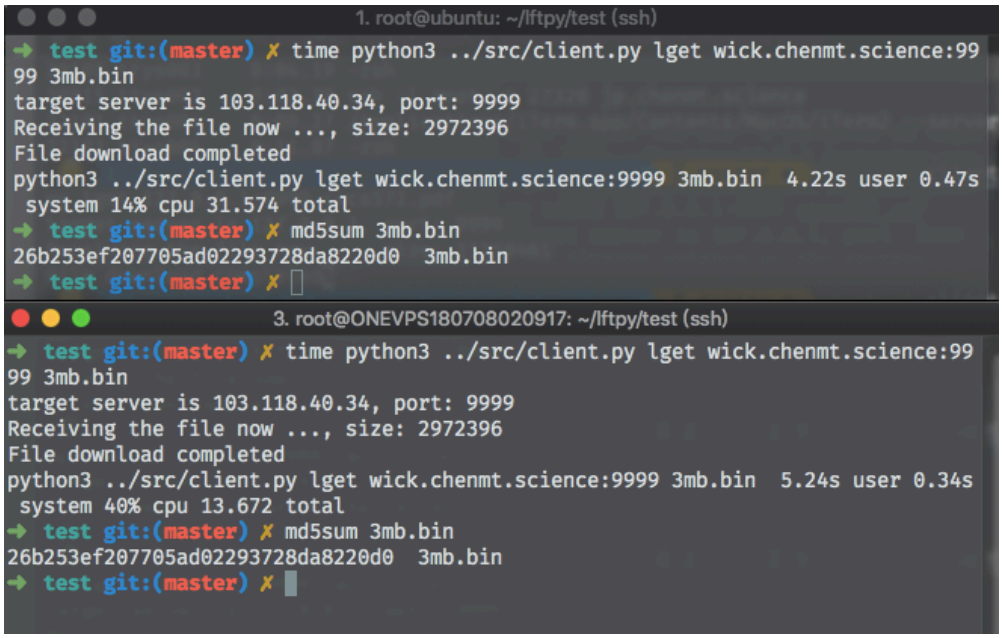
加入一台位于美国洛杉矶的服务器作为客户端进行服务端对多个客户端服务的测试，洛杉矶服务器时延如下：



启动两个客户端：



传输结束结果：



可算得速率分别为 0.76Mbits/s 和 1.76Mbits/s。