计算机网络期中项目

姓名：吴宇祺 学号：16340242 专业及方向：软件工程（数字媒体）

需求

1. 支持大文件传输，例如1G的电影
2. 包括客户端和服务端，遵从命令格式
3. 使用UDP作为传输层协议
4. 100%可靠性（大量测试）
5. 流控制
6. 堵塞控制
7. 服务端并发访问控制
8. 提供有效查错信息

设计文档

一、实验环境及文件结构

操作系统：Window 7 编程语言：python 版本 ：3.7

文件结构：

-

二、UDP协议和客户端和服务端架构

1. 客户端
2. 服务端
3. 命令格式：

```
$ cd code
$ LFTP lsend 127.0.0.1 ../testData/saveVideo.avi
$ LFTP lget 127.0.0.1 D:/saveVideo.avi
```

三、命令传输和链接建立/断开：

- client接收到命令之后，向server发送连接请求（握手），包括client选择的（随机）初始序列号；
- server是“总是运行”的；若server不处于运行状态，client检测到超时重发；如果请求超时5次，向控制台返回“Server does not Response”。
- server主线程启动，开启请求监听线程；
  - 若server监听到连接请求，返回客户端IP地址，接口和初始序列号给主线程
  - 如果该连接请求的IP地址未被处理过，创建新请求线程处理该请求，并将线程号加入到线程队列（先来先服务），每次取队列首部的线程进行服务；
- 请求处理线程
  - 线程唤醒，传入客户端IP地址，端口，命令和初始序列号
  - 线程分析命令
  - 若命令为上传文件，服务端发送：确认command的ACK和分配给该线程的RcvBuffer大小；
    - 客户端接收到该ACK分组后，更新rwnd，发送文件信息：文件名，文件字节数，文件格式和指定文件路径
    - 服务端接收到文件信息分组后，若文件路径不存在，返回deny请求；若存在指定文件路径初始化文件，开始接收
  - 若命令为下载文件（和RcvBuffer），服务端发送确认command的ACK；
    - 客户端接收到该ACK之后，发送文件名，文件格式和目标文件路径；
    - 服务端接收到文件信息分组后，在本地搜索文件是否存在；若不存在返回deny请求，若存在，开始发送。
  - 接收到断开命令，回复ack进入半连接状态，接收到客户端第二次发送的断开连接，回复ack并完全断开；
- **非阻塞接收分组**

  udp的套接字接收 socket.recvfrom() 默认是阻塞式的，就是在（被动）接收分组之前程序都不会从recvfrom函数返回；这样效率低下也无法实现流水线传输。用函数socket.setblocking(0)设置为非阻塞式的。非阻塞的recvfrom没有接收到分组就要函数返回时会抛出异常，需要进行异常处理。基本代码：

```
while True:
    try :
        reply, addr = s.recvfrom(receiveSize)
        break
    except BlockingIOError:
        pass
```

例如上传文件：

客户端：

```
    # Client: Upload.py
    # set up connection:
    while True:
        command = 1  #上传命令
        print("sending command ", command ," and initial seqNum ", InitialSeqNum)
```

```
        if Timer == TTL:                                    #发送命令和初始序列号
            s.sendto(struct.pack(commandformat, command, InitialSeqNum), (dstIP,
dstPort))
            Timer = 0
            TTL *= 2                                        #超时TTL加倍
        Timer += 1
        try :
            print(receiveSize)
            reply, addr = s.recvfrom(receiveSize)       #接收服务端对请求连接的去人和分配
缓存大小
            RcvBuffer, seq = struct.unpack(commandformat, reply)
            print("command pkt ack recieved")

            Timer = 3
            while True:
                if Timer == TTL:
                    print("sending file info")
                    fileInfo = bytes(filePath.encode('utf-8')) +
bytes(fileName.encode('utf-8')) + bytes(("."+fileFormat).encode('utf-8'))
                    s.sendto(fileInfo, (dstIP, dstPort))  #发送将要上传的文件信息
                Timer += 1
                try :
                    reply, addr = s.recvfrom(receiveSize) #接收ack，开始文件数据传输
                    print("server : fileInfo recieved ( connection setup")
                    break
                except BlockingIOError:
                    pass
            break
        except BlockingIOError:
            pass

```

服务端建立连接：主函数监听到连接请求之后分析命令，进入replyUpload函数处理上传请求或者进入replyDownload处理下载请求

```
#Server main
# 服务端打开socket接受连接请求
s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
s.bind(('127.0.0.1', 9999))
print('Bing UDP on 9999')

#监听连接请求
text, addr = s.recvfrom(struct.calcsize(commandformat))
command, expSeqNum = struct.unpack(commandformat, text)
print("Received command from ", addr, " command : ", command, " expSeq : ",
expSeqNum)

if command == 1:   # 上传命令

    replyUpload(s, expSeqNum, addr)


elif command == 0: # 下载命令
```

```
    RcvBuffer = expSeqNum
    replyDownload(s, RcvBuffer, addr)

s.close()
# Server replyDownload
    RcvBufferSize = 20 # 分配buffer数量
    s.sendto(struct.pack(commandformat, RcvBufferSize, expSeqNum), addr)

    # 接收文件信息
    fileInfo, addr = s.recvfrom(1024)
    f = open(str(fileInfo, encoding = 'utf-8'), mode='wb')
    print("file info : ", fileInfo)
    s.sendto(struct.pack(commandformat, RcvBufferSize, expSeqNum), addr)
```

四、差错检测和恢复设计：

发送方：

1. GBN：累计确认

```
#Client Upload：
# event: 接收到ACK
    if Sendbase <= ackSeq:        # 累计确认，之前的都是正确确认收到的
        Sendbase = ackSeq + mss

        while len(Window) != 0:
            if Window[0].seqNum < Sendbase:
                del Window[0]
            else: break

        if Sendbase != NextSeqNum:  #如果当前还有已发送但未被确认的分组：reset timer
            Timer = 0;
```

2. 超时间隔加倍

```
    # Client: Upload.py
    # set up connection：
    while True:
        command = 1  #上传命令
        print("sending command ", command ," and initial seqNum ", InitialSeqNum)
        if Timer == TTL:                                    #发送命令和初始序列号
            s.sendto(struct.pack(commandformat, command, InitialSeqNum), (dstIP, dstPort))
            Timer = 0
            TTL *= 2                                    #超时TTL加倍
        Timer += 1
```

3. 快速重传（见拥塞控制）

接收方：

1. 确认最后一个按序到达的分组的序列号
2. 使用优先队列，按序列号从小到大缓存失序到达的分组

```
# Server replyUpload()

#初始化：
    ReceiveBuffer = PriorityQueue(RcvBufferSize) # 按缓存的seqNum从小到大排列
    bufferedSeq = []
    rwnd = RcvBufferSize            # 缓存可用空间

# event :接收到新分组
    seq, ack, w = struct.unpack(format, data);

    if seq < expSeqNum:    # 丢弃冗余分组
        logf.write("redundant pkt："+ str(seq)+ "\n")
        continue
    elif seq > expSeqNum:  # 若缓存仍有空间，缓存乱序到达的pkt
        if not ReceiveBuffer.full() and seq not in bufferedSeq:
            logf.write("put buffer : " + str(seq)+ "\n")
            ReceiveBuffer.put(initPacket(seq, w))
            bufferedSeq.append(seq)
            rwnd -= 1

    else :                  #seq == expSeqNum:
        ackSeq = seq
        f.write(w)
        expSeqNum += len(w)

        #检查buffer中有无 顺序可写的数据包
        if not ReceiveBuffer.empty():       # buffer不为空
            tmp = ReceiveBuffer.get()       # 取队头元素
            bufferedSeq.remove(tmp.seqNum)

            while tmp.seqNum <= expSeqNum:
                if tmp.seqNum < expSeqNum:    # 若队列中有小于expNum的缓存（冗余），丢弃
                    if ReceiveBuffer.empty():
                        break
                    tmp = ReceiveBuffer.get()
                    bufferedSeq.remove(tmp.seqNum)

                elif expSeqNum == tmp.seqNum: # 确认，移出缓冲并写入磁盘
                    ackSeq = tmp.seqNum
                    f.write(w)
                    expSeqNum += len(tmp.data)
                        if ReceiveBuffer.empty():
                            break
                    tmp = ReceiveBuffer.get()
                    bufferedSeq.remove(tmp.seqNum)

                if expSeqNum < tmp.seqNum:
```

```
                    ReceiveBuffer.put(tmp)

            bufferedSeq.append(tmp.seqNum)
            rwnd = RcvBufferSize - ReceiveBuffer.qsize()

            # 发送ack
            ackPkt = struct.pack(ackformat, ackSeq, rwnd)
            s.sendto(ackPkt, addr)

```

五、流控制机制：

1. 建立连接握手时，接收方将RcvBuffer传输给发送方，发送方用于初始化变量rwnd

例如上传：

```
    #Server：replyUpload.py
    RcvBuffer, seq = struct.unpack(commandformat, reply)
    #...#
    rwnd = RcvBuffer
```

2. 发送方维护两个变量：LastByteSent和LastByteAcked并使得一下条件满足时才允许发送分组

$$LastByteSent - LastByteAcked <= rwnd * MSS$$

```
# Client Upload
    #发送分组条件：
    if (LastByteSent-LastByteAcked) <= rwnd*mss and (NextSeqNum-Sendbase)/mss < windowSize:
        #从文件二进制六中读取大小为MSS的分组
        s.sendto(struct.pack(format, NextSeqNum, 0, data), (dstIP, dstPort))
        LastByteSent = NextSeqNum - 1   #更新LastByteSent

    elif (LastByteSent-LastByteAcked) > rwnd*mss:
        # logf记录 发送单个字节流控制分组 日志
    s   .sendto(struct.pack(flowCtlrFormat, NextSeqNum-mss, -1), (dstIP, dstPort))

    #event ： 接收到分组确认后，更新 rwnd 和 LastByteAcked：
    ackSeq, rwnd = struct.unpack(ackformat, reply)
    LastByteAcked = ackSeq + mss  # 流控制中，ack seq 不变化
```

六、堵塞控制机制：

发送方维护两个变量：

1. **拥塞窗口** cwnd 并保证 在一个发送方未被确认的数据量并会超过拥塞窗口和接收窗口中的最小值：

$$LastByteSent - LastByteAcked <= min\{cwnd, rwnd\} * MSS$$

2. **慢启动阈值** ssthresh
3. **丢包事件**定义：超时或者收到接收方的3个冗余ACK；

慢启动，拥塞避免伪代码：

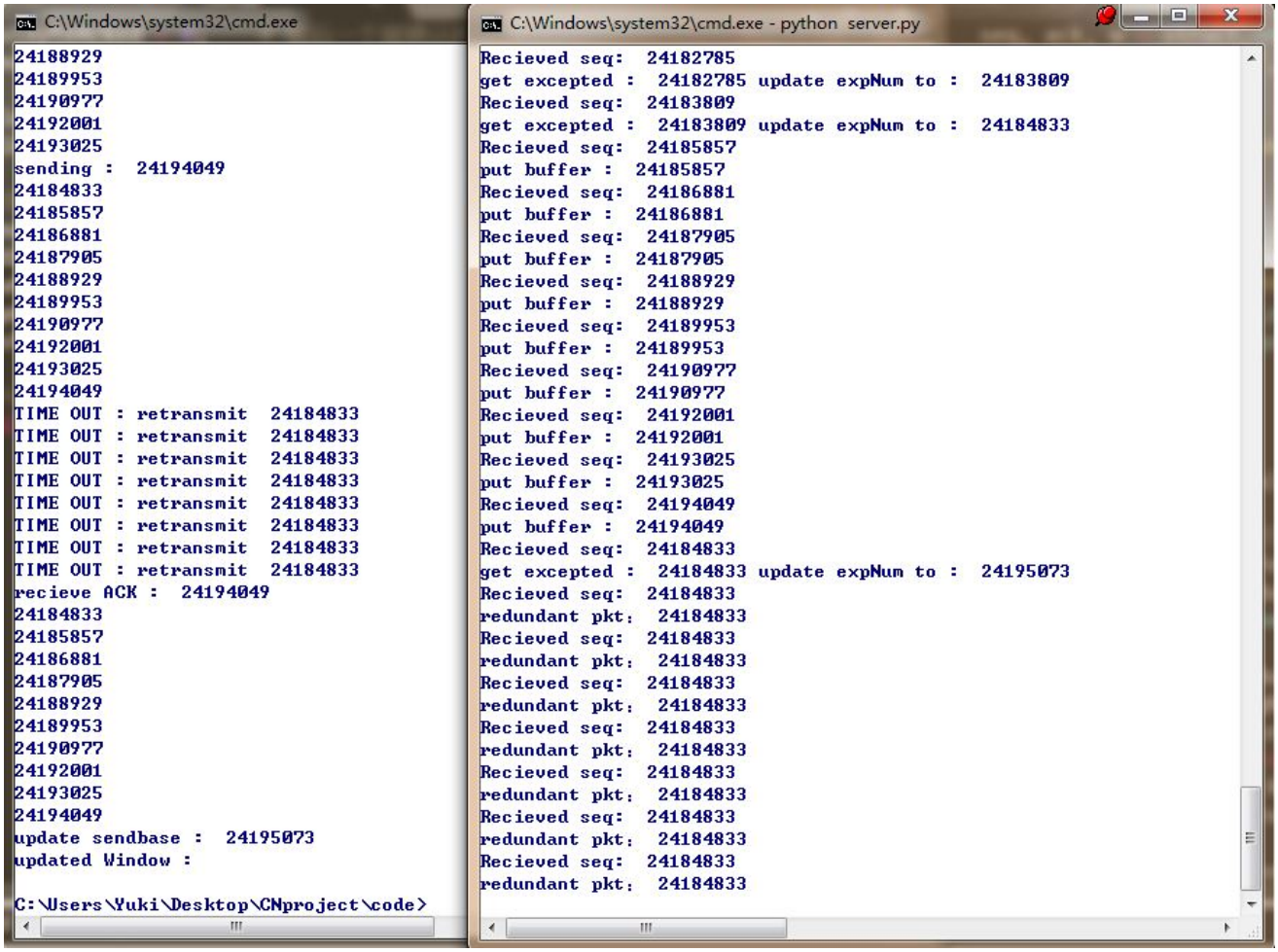
> 初始化：cwnd = 1*MSS，ssthresh = 16，dupACKcount = 0，state = 慢启动（0）；
>
> 慢启动：
>
> **WHILE** True
>
> event : 接收newACK
>
> if state == 0
>
> cwnd += 1
>
> else
>
> cwnd += 1/cwnd
>
> dupACKcount = 0
>
> event : 接收dupACK
>
> dupACKcount++
>
> event : cwnd >= ssthresh
>
> state = 1 #进入拥塞避免状态
>
> event : timeout or dupACKcount == 3
>
> ssthresh = cwnd / 2
>
> cwnd = 1
>
> state = 0
>
> 重传丢失分组（冗余ACK 的下一分组序号） #快速重传

```
# Client Upload
    #event : ACK received, with field value of y
    reply = 0;
    try:
        reply, dstAddr = s.recvfrom(receiveSize)
        ackSeq, rwnd = struct.unpack(ackformat, reply)

        LastByteAcked = ackSeq + mss  # 流控制中，ack seq 不变化
        if lastACK != ackSeq:
```

```
            if State == 0:
                cwnd += 1
            else:
                cwnd += 1/cwnd
            dupACKcount = 0
            lastACK = ackSeq
        else :
            duoACKcount += 1

    #event : timer out / duoACKcount
    Timer += 1
    if Timer == TTL or dupACKcount == 3:
        State = 0
        ssthresh = cwnd / 2
        cwnd = 1
        duoACKcount = 0
        Timer = 0
        # if Sendbase != NextSeqNum and (LastByteSent-LastByteAcked) <= rwnd*mss:
        if len(Window) != 0:
            logf.write("TIME OUT : retransmit " +str(Window[0].seqNum)+ "\n")
            retransmit = struct.pack(format, Window[0].seqNum, 0, Window[0].data)
            s.sendto(retransmit, (dstIP, dstPort))
```

测试文档

测试：

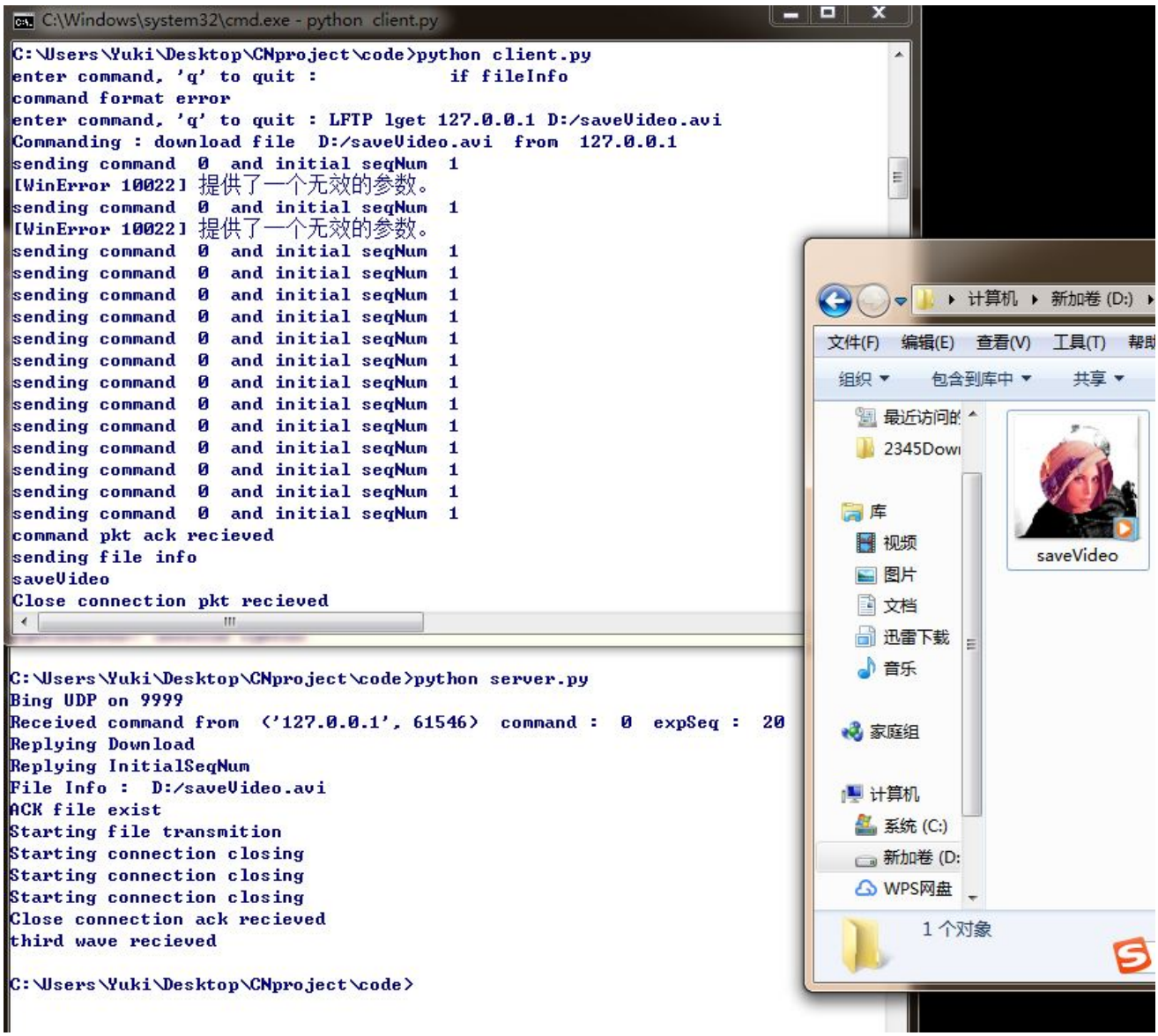

C:\Windows\system32\cmd.exe
24188929
24189953
24190977
24192001
24193025
sending : 24194049
24184833
24185857
24186881
24187905
24188929
24189953
24190977
24192001
24193025
24194049
TIME OUT : retransmit 24184833
TIME OUT : retransmit 24184833
TIME OUT : retransmit 24184833
TIME OUT : retransmit 24184833
TIME OUT : retransmit 24184833
TIME OUT : retransmit 24184833
TIME OUT : retransmit 24184833
TIME OUT : retransmit 24184833
recieve ACK : 24194049
24184833
24185857
24186881
24187905
24188929
24189953
24190977
24192001
24193025
24194049
update sendbase : 24195073
updated Window :
C:\Users\Yuki\Desktop\CNproject\code>
C:\Windows\system32\cmd.exe - python server.py
Recieved seq: 24182785
get excepted : 24182785 update expNum to : 24183809
Recieved seq: 24183809
get excepted : 24183809 update expNum to : 24184833
Recieved seq: 24185857
put buffer : 24185857
Recieved seq: 24186881
put buffer : 24186881
Recieved seq: 24187905
put buffer : 24187905
Recieved seq: 24188929
put buffer : 24188929
Recieved seq: 24189953
put buffer : 24189953
Recieved seq: 24190977
put buffer : 24190977
Recieved seq: 24192001
put buffer : 24192001
Recieved seq: 24193025
put buffer : 24193025
Recieved seq: 24194049
put buffer : 24194049
Recieved seq: 24184833
get excepted : 24184833 update expNum to : 24195073
Recieved seq: 24184833
redundant pkt: 24184833
Recieved seq: 24184833
redundant pkt: 24184833
Recieved seq: 24184833
redundant pkt: 24184833
Recieved seq: 24184833
redundant pkt: 24184833
Recieved seq: 24184833
redundant pkt: 24184833
Recieved seq: 24184833
redundant pkt: 24184833
Recieved seq: 24184833
redundant pkt: 24184833

上传：

```
C:\Users\Yuki\Desktop\CNproject\code>python client.py
enter command, 'q' to quit : LFTP lsend 127.0.0.1 ../testData/saveVideo.avi
Commanding : upload file  ../testData/saveVideo.avi  to  127.0.0.1
saveVideo
sending command  1  and initial seqNum  1
sending command  1  and initial seqNum  1
command pkt ack recieved
sending file info
server : fileInfo recieved ( connection setup
Starting file transmition
Starting connection closing
Starting connection closing
Close connection ack recieved
third wave recieved
enter command, 'q' to quit : LFTP lsend 127.0.0.1 ../testData/saveVideo.avi
Commanding : upload file  ../testData/saveVideo.avi  to  127.0.0.1
saveVideo
sending command  1  and initial seqNum  1
sending command  1  and initial seqNum  1
command pkt ack recieved
sending file info
server : fileInfo recieved ( connection setup
Starting file transmition
Starting connection closing
Starting connection closing
Close connection ack recieved
third wave recieved
enter command, 'q' to quit :
```

```
C:\Users\Yuki\Desktop\CNproject\code>python server.py
Bing UDP on 9999
Received command from  ('127.0.0.1', 59943)  command :  1  expSeq :  1
File info :  b'D:/saveVideo.avi'
Close connection pkt recieved

C:\Users\Yuki\Desktop\CNproject\code>
```

下载：


C:\Windows\system32\cmd.exe - python client.py
C:\Users\Yuki\Desktop\CNproject\code>python client.py
enter command, 'q' to quit : if fileInfo
command format error
enter command, 'q' to quit : LFTP lget 127.0.0.1 D:/saveVideo.avi
Commanding : download file D:/saveVideo.avi from 127.0.0.1
sending command 0 and initial seqNum 1
[WinError 10022] 提供了一个无效的参数。
sending command 0 and initial seqNum 1
[WinError 10022] 提供了一个无效的参数。
sending command 0 and initial seqNum 1
sending command 0 and initial seqNum 1
sending command 0 and initial seqNum 1
sending command 0 and initial seqNum 1
sending command 0 and initial seqNum 1
sending command 0 and initial seqNum 1
sending command 0 and initial seqNum 1
sending command 0 and initial seqNum 1
sending command 0 and initial seqNum 1
sending command 0 and initial seqNum 1
sending command 0 and initial seqNum 1
sending command 0 and initial seqNum 1
command pkt ack recieved
sending file info
saveVideo
Close connection pkt recieved
C:\Users\Yuki\Desktop\CNproject\code>python server.py
Bing UDP on 9999
Received command from ('127.0.0.1', 61546) command : 0 expSeq : 20
Replying Download
Replying InitialSeqNum
File Info : D:/saveVideo.avi
ACK file exist
Starting file transmition
Starting connection closing
Starting connection closing
Starting connection closing
Close connection ack recieved
third wave recieved
C:\Users\Yuki\Desktop\CNproject\code>
计算机 ▸ 新加卷 (D:) ▸
文件(F) 编辑(E) 查看(V) 工具(T)
组织 包含到库中 共享
最近访问的
2345Down
库
视频
图片
文档
迅雷下载
音乐
家庭组
计算机
系统 (C:)
新加卷 (D:
WPS网盘
saveVideo
1 个对象