《计算机网络》

设计文档

（LFTP）

学 生 姓 名：黄梓轩

学　　号：16340090

时　　间：2018 年 12 月 4 日

一. 概述

本项目是一个网络应用LFTP，为互联网上的两台电脑提供传输大文件的服务。

二. 项目环境

本应用的编程语言为python2，适用于windows和linux操作系统。

服务端需要有《server.py》，《dataSolver.py》，《sender.py》，《receiver.py》文件。

客户端需要有《client.py》，《dataSolver.py》，《sender.py》，《receiver.py》文件。

三. 各文件简要说明

server.py：运行后等待客户端的请求，接收到客户请求后，创建一个新进程处理该客户的请求，原进程继续等待新的请求。

client.py:向服务端发送请求，接收到回应后，执行相应的操作（发送文件或者等待服务端发送的文件，文件路径一定要正确，否则程序会出错）

dataSolver.py：提供发送内容所需要的header，以及解开接收内容的header和内容。

sender.py:为客户端或者服务端提供发送文件的接口。

receiver.py：为客户端或者服务端提供接收文件的接口。

四. 项目的需求和实现

1.采用cs结构

具体实现：服务端的主要实现在client.py文件上，服务端的主要实现在server.py文件上，由于客户端和服务端都需要实现发送和接收文件，为避免冗余，因此将这两部分写成两个模块，由于发送信息时有很多冗余部分，如额外添加的header信息等，因此dataSolver这个模块用于处理发送信息时冗余的打包和解包过程。

2. 客户端既能下载文件，也能上传文件。

发送文件需要输入下述命令

python lftp lsend myserver mylargefile

接收发送文件需要输入下述命令

python lftp lsend myserver mylargefile

其中myserver为ip地址，mylargefile为文件路径。

具体实现：

代码见下图

```
if __name__ == "__main__":
    if len(sys.argv) != 4:
        print "the number of arguments must be 3, lget|lsend  myserver mylargefile"
    else:
        addr = (sys.argv[2] , send_port)
        mylargefile = sys.argv[3]
        sk = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
        if sys.argv[1] == "lsend":
            requestSend(addr , mylargefile , sk)
        elif sys.argv[1] == "lget":
            requestGet(addr , mylargefile , sk)
        else
            print 'first argument must be lget or lsend'
```

对参数进行解析，若为lsend，则要向服务端请求发送文件，若

为lget，则要向服务端请求得到文件，若不是上述两者，则输出错误提示并结束程序。

3. 应用必须使用UDP作为传输协议

具体实现：所有的socket都使用udp的socket。传输的主要流程是服务端收到新客户端的请求，创建一个新的进程监听新的端口处理请求。

三次握手结束后，两端有一端开始发送文件，根据流量控制和阻塞控制的窗口决定一个RTT要传输多少个包，每次发的包在发送文件的位置是最近接收到的ack位置。发n个包，则每发一个包就等待RTT/n的时间再发一个包，一个RTT结束后，再通过两个窗口决定下一个RTT要传输多少个包。

而接收方则只接受自己要ack的包，接收的包seq num不等于ack num则把这个不需要的包直接丢掉。若seq num大于ack num，则可能发生了丢包，因此发送一个ack包回去。若seq num小于ack num，则不发送ack包回去。

传输结束后，客户端关闭，同时服务端关闭处理该客户端的子进程。

4. 应用必须像TCP一样100%可靠

5. 应用必须实现像TCP一样的流控制

6. 应用必须实现像TCP一样的阻塞控制

具体实现：UDP的不可靠性主要体现在丢包和乱序这两个问题上，至于报文内容的可靠性由于UDP有checkSum，因此不成问题，同时为了验证客户端和服务端的联通性，模仿TCP实现三次握手。

为了实现丢包，乱序，流控制，阻塞控制的处理，首先预先规定在发送的UDP报文的内容中，前16个字节为拓展的header(bool类型占1个字节，int占4个字节)。

SYN	ACK	send（判断请求是下载还是上传）	补齐
Sequence Number（int）			
Ack Number（int）			
Receive window（int）			

丢包，乱序：

靠syn num和ack num来解决丢包和乱序问题，当接收方接收到一个包后，若seq num比ack num小，那么说明该包已经接收过了，不发送ack，若seq num等于ack num，则说明该包是需要的，将内容放到缓冲区。若seq num比ack num大，那么说明顺序乱了或者包丢了，发送ack包。接收方接收到3次重复ack，说明丢包发生。接收方每次发包都是从最近接收到的ack位置开始，保证了传输数据的顺序不会出错。

```
while nowAckNum < fileSize:
    try:
        data,addr = recvSk.recvfrom( Mss )
    except:
        #time out
        break
    recvDataHead , recvDataContent = dataSolver.splitData(data)
    if recvDataHead['seqNum'] < nowAckNum:
       # don't send data
        continue
    elif recvDataHead['seqNum'] == nowAckNum:
        putDataIntoCache(recvDataContent)
        nowAckNum += len(recvDataContent)
    else:
        #debug
        print 'seqNum ' , recvDataHead['seqNum']  , ' != ackNum ' , nowAckNum
    cwdWindow = cache.maxsize - cache.qsize()
    ackHead = dataSolver.getSendAckHeader(nowAckNum , cwdWindow)
    ackContend = dataSolver.packTheHeader(ackHead)
    recvSk.sendto(ackContend , addr)
```

流控制：

靠Receive window来实现流控制，接收方的主线程负责接收数据包，新开一个线程负责将缓冲区内容写入磁盘。

```
writeThread = threading.Thread(target=writeData,args=[fileName,fileSize,])
#writeThread.daemon = True
writeThread.start()
```

```
def writeData(fileName,fileSize):
    global cache , recvContinue , mutex
    file = open(fileName, 'wb')
    writePos = 0
    oneTimeWriteChars = 1

    while writePos < fileSize and recvContinue:
        if not cache.empty() and mutex.acquire(1):
            writeCharsNum =  min(cache.qsize() , oneTimeWriteChars)
            writeStr = ''
            for i in range(0 , writeCharsNum):
                writeStr += cache.get()
            file.write(writeStr)
            writePos += writeCharsNum
            mutex.release()

    file.close()
```

当主线程接收到发送方的包时，将包内的内容解包，若seq num大于等于ack num，发送一个ack包，ack包中的Receive window告知发送方缓冲区剩余空间大小，

```
cwdWindow = cache.maxsize - cache.qsize()
ackHead = dataSolver.getSendAckHeader(nowAckNum , cwdWindow)
ackContend = dataSolver.packTheHeader(ackHead)
recvSk.sendto(ackContend , addr)
```

发送方根据Receive window和拥塞控制的Cong window决定发送包的数量，为避免连接中断，每个RTT都至少发一个包。

```
sendPackNum = max(1 , min(congWin , flowControlWindowSize) / Mss)
```

阻塞控制：

阻塞控制需要计算RTT，在本应用中，将RTT固定为0.5。

由于接收方既要发送数据包，又要接收ack，因此开了一个线程来接收ack包。

```
getAckThread = threading.Thread(target=getAckMessage,args=[sk,flieSize])
getAckThread.daemon = True
getAckThread.start()
```

给socket设置一个超时时间，若超时，说明发生了超时，进行阻塞控制对congWin和阈值thresold进行设定，超时2次说明可能客户端终止了，那么结束该线程。

```
sk.settimeout(timeLimit)
while recentAckNum < flieSize and sameTimeOutCount < 2:
    try:
        data , addr = sk.recvfrom(dataMaxSize)
    except:
        #time out
        Threshold = congWin / 2
        congWin = 1 * Mss
        print 'get ack pack timeout!!!!!'
        sameTimeOutCount += 1
        continue
```

接收ack包后，对内容进行解包，若这个包的ack num与recentAckNum相同，这说明可能发生了丢包，因此对重复的ack次数进行计数，若计数到3，这说明丢包了，那么进行拥塞控制的快速恢复。若这个的ack比recentAckNum大，那么更新recentAckNum。

若此时congWin比阈值小，那么说明此时是慢启动状态，进行慢启动的拥塞控制。

```
recvDataHead , recvDataContent = dataSolver.splitData(data)
#重复ack
if recentAckNum == recvDataHead['ackNum']:
    sameAckRepateTime += 1
    print 'same ack pack ' , recentAckNum
    #丢包
    if sameAckRepateTime == 2:
        Threshold = congWin / 2 + 3 * Mss
        congWin = Threshold
        print 'loss pack!!!!!!'
elif recentAckNum < recvDataHead['ackNum']:
    sameAckRepateTime = 0
    recentAckNum = recvDataHead['ackNum']
    #慢启动
    if congWin < Threshold:
        congWin += 1 * Mss
flowControlWindowSize = recvDataHead['cwdWindow']
```

每个RTT发送n个包后判断congWin是否比阈值大，若是，则进行阻塞避免。

```
#阻塞避免
if congWin >= Threshold:
    congWin += 1 * Mss
```

7. 应用在同一时间能够提供服务给多个客户端

为简单展示起见，本应用最多支持两个客户端，可以调高supportPortNum支持更多客户端。注意要确保13001, 13002, 13003端口开启了udp服务。

```
12    dataMaxSize = 1024
13    recv_port = 13001
14    supportPortNum = 2
```

```
48        #13002 13003
49        for i in range(1,supportPortNum+1):
50            avialablePort.put(recv_port + i)
```

当服务端接收到新的请求后，开启一个新的进程监听别的端口并将新端口放到udp报文中告知客户端。同时由于进程开启需要时间，因此主进程先休眠一下，等子进程开启完并监听端口后才发送这个SYNACK包。

```
newPort = getNewPort()
if newPort != -1:
    newProcess = Process(target=solveSendCommand,
                         args=(avialablePort,newPort,addr,fileName,fileSize,))
    newProcess.start()
    time.sleep(0.5)
    newPortStr = struct.pack('i' , newPort)
    recvSk.sendto(synAckContent + newPortStr, addr)
```

这里注意不能让子进程的新端口发送这个包，因为似乎学校或者电脑的防火墙不会接受陌生端口，这里因为是服务端之前发过syn包

去云服务器的13001端口，因此该端口不算陌生，但服务端没有发过报文到子进程的新端口，因客户端无法接收子进程新端口发送的报文。

也不能把原来的socket作为参数传进去，因为socket这个类不能作为创建新进程的参数。

这里de好久的bug。

8. 提供有用的debug信息

在传输文件的时候，提供了一些debug信息，具体可以在测试文档中见到中间过程输出的debug信息。

五. 项目文件中主要的接口简要说明。

《server.py》

GetNewPort：用于得到新的可用的端口。

```
def getNewPort():
    global avialablePort
    if avialablePort.empty():
        return -1
    else:
        return avialablePort.get()
```

solveSendCommand：处理客户端发送文件的请求，主函数会用该函数创建一个新的进程处理新的客户端的请求。当接收到客户端的ack后，说明三次握手成功，调用receiver模块的recvSendFile函数接收客户端发送的文件。

```
def solveSendCommand(avialablePort , newPort , addr , fileName , fileSize):
    sk = socket.socket(socket.AF_INET,socket.SOCK_DGRAM)
    sk.bind(("" , newPort))
    #sk.settimeout(5)
    data,addr = sk.recvfrom(dataMaxSize)
    recvDataHead , recvDataContent = dataSolver.splitData(data)
    if recvDataHead['ack']:
        receiver.recvSendFile(sk , addr , fileName , fileSize)
    sk.close()
    avialablePort.put(newPort)
```

solveGetCommand：处理客户端得到文件的请求，主函数会用该函数创建一个新的进程处理新的客户端的请求。当接收到客户端的ack后，说明三次握手成功，调用sender模块的sendFile函数向客户端发送文件。

```
def solveGetCommand(avialablePort , newPort , addr , fileName , fileSize):
    sk = socket.socket(socket.AF_INET,socket.SOCK_DGRAM)
    sk.bind(("",newPort))
    #sk.settimeout(5)
    data,addr = sk.recvfrom(dataMaxSize)
    recvDataHead , recvDataContent = dataSolver.splitData(data)
    if recvDataHead['ack']:
        sender.sendFile( sk , addr , fileName , fileSize)
    sk.close()
    avialablePort.put(newPort)
```

《client.py》

requestSend:向服务端请求上传文件，进行三次握手，三次握手成功后，调用sender模块的sendFile函数进行文件发送。

```
def requestSend(addr , mylargefile , sk):
    #三次握手开始
    fileSize = os.path.getsize(mylargefile)
    fileSizeStr = struct.pack('i', fileSize)
    #send syn
    synHead = dataSolver.getSendSynHeader()
    synContent = dataSolver.packTheHeader(synHead)
    sk.sendto(synContent + fileSizeStr + mylargefile , addr)
    while(True):
        data , addr = sk.recvfrom(dataMaxSize)
        recvDataHead , recvDataContent = dataSolver.splitData(data)
        #get syn ack
        if recvDataHead['syn'] and recvDataHead['ack'] :
            newPort = struct.unpack('i' , recvDataContent)[0]
            addr = (sys.argv[2] , newPort)
            print 'new addr ' ,  addr
            #send ack
            ackHead = dataSolver.getSendAckHeader()
            ackContent = dataSolver.packTheHeader(ackHead)
            sk.sendto(ackContent , addr)
            break
    #三次握手结束 ，  发送文件
    sender.sendFile( sk , addr , mylargefile , fileSize)
```

requestGet:向服务端请求下载文件，进行三次握手，三次握手成功后，调用receiver模块的recvSendFile函数进行文件发送。

```
def requestGet(addr , mylargefile , sk):
    #三次握手开始
    #send syn
    synHead = dataSolver.getRecvSynHeader()
    synContent = dataSolver.packTheHeader(synHead)
    sk.sendto(synContent + mylargefile , addr)
    while(True):
        data , addr = sk.recvfrom(dataMaxSize)
        recvDataHead , recvDataContent = dataSolver.splitData(data)
        #get syn ack
        if recvDataHead['syn'] and recvDataHead['ack'] :
            portAndFileSize = struct.unpack('ii' , recvDataContent)
            newPort = portAndFileSize[0]
            fileSize = portAndFileSize[1]
            addr = (sys.argv[2] , newPort)
            #send ack
            ackHead = dataSolver.getRecvAckHeader()
            ackContent = dataSolver.packTheHeader(ackHead)
            sk.sendto(ackContent , addr)
            break
    #三次握手结束 ， 接受文件
    receiver.recvSendFile(sk , addr , mylargefile , fileSize)
```

《dataSolver.py》

getSendSeqHeader， getSendSynHeader.....：写了数个这种获得header的函数，后面还有几个。为三次握手的不同情况，以及发送文件和接受文件传输报文的不同情况提供不同的头文件，有一点点冗余。

```
def getSendSeqHeader(seqNum = 0, cwdWindow = 0):
    head = {
        'syn': False,
        'ack': False,
        'send':True,
        'seqNum': seqNum ,
        'ackNum': 0 ,
        'cwdWindow': cwdWindow
    }

    return head

def getSendSynHeader():
    head = {
        'syn': True,
        'ack': False,
        'send':True,
        'seqNum': 0 ,
        'ackNum': 0 ,
        'cwdWindow': 0
    }
    return head
```

splitData：将udp报文解包，返回它的header信息和内容信息

```
def splitData( data ):
    dataHead = {
        'syn': None,
        'ack': None,
        'send':None,
        'seqNum': None,
        'ackNum': None ,
        'cwdWindow': None
    }
    head = struct.unpack('???iii' , data[:16])
    dataHead['syn'] = head[0]
    dataHead['ack'] = head[1]
    dataHead['send'] = head[2]
    dataHead['seqNum']  = head[3]
    dataHead['ackNum']  = head[4]
    dataHead['cwdWindow']  = head[5]
    return dataHead , data[16:]
```

packTheHeader:将header打包成字符串。

```
def packTheHeader(dataHead):
    return struct.pack('???iii', dataHead['syn'] , dataHead['ack'] , dataHead['send']
                       , dataHead['seqNum'] , dataHead['ackNum'] , dataHead['cwdWindow'] )
```

《sender》

sendFile：用于不断地发送文件，由于既要发送文件内容，也要接收回传的ack包，因此要新开一个线程执行getAckMessage函数来接收ack包。

```
def sendFile(sk , addr , mylargefile , flieSize):
    global recentAckNum , Threshold ,  congWin  , RTT , sameTimeOutCount , flowControlWindowSize
    initGlobalVariable()
    flie = open(mylargefile,'rb')
    getAckThread = threading.Thread(target=getAckMessage,args=[sk,flieSize])
    getAckThread.daemon = True
    getAckThread.start()
    while recentAckNum < flieSize and sameTimeOutCount < 2:
        seqNum = recentAckNum
        #阻塞避免
        if congWin >= Threshold:
            congWin += 1 * Mss
        #print
        sendPackNum = max(1 , min(congWin , flowControlWindowSize) / Mss)
        print 'RTT is' , RTT , 'congWin is' , congWin , 'sendPackNum '\
        , sendPackNum , 'flowControlWindowSize' , flowControlWindowSize , 'recentAckNum' , recentAckNum
        ackContentArr = []
        for i in range (0, sendPackNum):
            if seqNum >= flieSize:
                break
            head = dataSolver.getSendSeqHeader(seqNum)
            ackContent = dataSolver.packTheHeader(head)
            dataSize = getSendDataSize(ackContent)
            flie.seek(seqNum)
```

```
        #没有一次性读完，选择一个个读取字符，可以知道文件结尾在哪
        if seqNum + dataSize >= flieSize:
            for j in range(0 , dataSize):
                if seqNum >= flieSize:
                    break
                ackContent = ackContent + struct.pack("1s"  , flie.read(1) )
                seqNum += 1
        else:
            packStr = str(dataSize) + 's'
            ackContent = ackContent + struct.pack(packStr  , flie.read(dataSize) )
            seqNum += dataSize
        ackContentArr.append(ackContent)

    for i in range (0, len(ackContentArr)):
        sk.sendto(ackContentArr[i] , addr)
        time.sleep( RTT / sendPackNum)
flie.close
if recentAckNum < flieSize:
    print 'timeout too much , stop send'
else:
    print 'send file succesfully'
```

getAckMessage：获取接收方发送的ack包，同时进行阻塞控制。

```
def getAckMessage(sk,flieSize):
    global recentAckNum , congWin , Threshold , sameTimeOutCount , flowControlWindowSize
    sameAckRepateTime = 0
    sk.settimeout(timeLimit)
    while recentAckNum < flieSize and sameTimeOutCount < 2:
        try:
            data , addr = sk.recvfrom(dataMaxSize)
        except:
            #time out
            Threshold = congWin / 2
            congWin = 1 * Mss
            print 'get ack pack timeout!!!!!'
            sameTimeOutCount += 1
            continue
        sameTimeOutCount = 0
        recvDataHead , recvDataContent = dataSolver.splitData(data)
        #重复ack
        if recentAckNum == recvDataHead['ackNum']:
            sameAckRepateTime += 1
            print 'same ack pack ' , recentAckNum
            #丢包
            if sameAckRepateTime == 2:
                Threshold = congWin / 2 + 3 * Mss
                congWin = Threshold
                print 'loss pack!!!!!!'
        elif recentAckNum < recvDataHead['ackNum']:
            sameAckRepateTime = 0
            recentAckNum = recvDataHead['ackNum']
            #慢启动
            if congWin < Threshold:
                congWin += 1 * Mss
        flowControlWindowSize = recvDataHead['cwdWindow']
    if sameTimeOutCount >= 2:
        print 'connect time out , stop sending'
```

《receiver》

recvSendFile:用于接受文件，由于既要接受文件又要将文件写入磁盘，因此为写文件的函数WriteData新开了一个线程。

```
def recvSendFile(recvSk , addr , fileName , fileSize):
    global cache  , recvContinue , mutex
    print 'begin to recv file ' , fileName , ' size is ' , fileSize
    initGlobalVariable()

    nowAckNum = 0
    writeThread = threading.Thread(target=writeData,args=[fileName,fileSize,])
    #writeThread.daemon = True
    writeThread.start()
    recvSk.settimeout(5)
    while nowAckNum < fileSize:
        try:
            data,addr = recvSk.recvfrom( Mss )
        except:
            #time out
            break
        recvDataHead , recvDataContent = dataSolver.splitData(data)
        if recvDataHead['seqNum'] < nowAckNum:
           # don't send data
            continue
        elif recvDataHead['seqNum'] == nowAckNum:
            putDataIntoCache(recvDataContent)
            nowAckNum += len(recvDataContent)
        else:
            #debug
            print 'seqNum ' , recvDataHead['seqNum']  , ' != ackNum ' , nowAckNum
        cwdWindow = cache.maxsize - cache.qsize()
        ackHead = dataSolver.getSendAckHeader(nowAckNum , cwdWindow)
        ackContend = dataSolver.packTheHeader(ackHead)
        recvSk.sendto(ackContend , addr)
```

putDataIntoCache:将传输的文件内容存到缓冲里。

```
def putDataIntoCache(recvDataContent):
    global cache
    while cache.full():
        pass
    while True:
        if mutex.acquire(1):
            for index in range(0 , len(recvDataContent) ):
                cache.put(recvDataContent[index] )
            mutex.release()
            break
```

writeData：用于将缓冲区内的数据写入磁盘中。

```
def writeData(fileName,fileSize):
    global cache , recvContinue , mutex
    file = open(fileName, 'wb')
    writePos = 0
    oneTimeWriteChars = 128

    while writePos < fileSize and recvContinue:
        if not cache.empty() and mutex.acquire(1):
            writeCharsNum =  min(cache.qsize() , oneTimeWriteChars)
            writeStr = ''
            for i in range(0 , writeCharsNum):
                writeStr += cache.get()
            file.write(writeStr)
            writePos += writeCharsNum
            mutex.release()

    file.close()
```