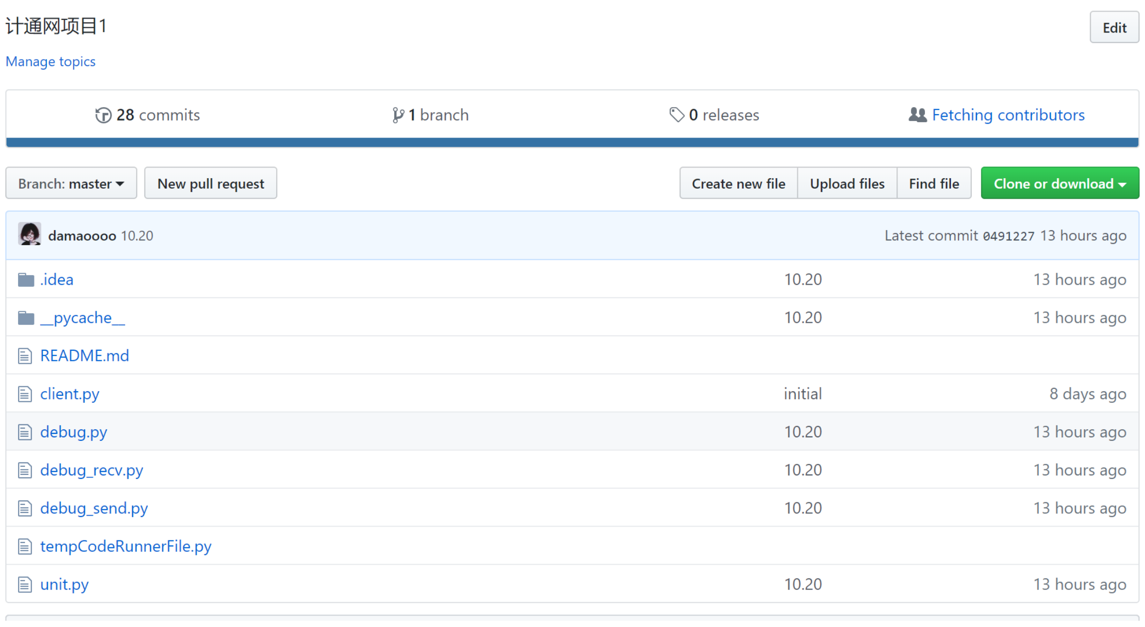
项目一阶段2 报告

项目一的阶段二花费了我大量心血，这里就总结一下如何实现。

环境:python 3.7

首先我是在国庆之前就已经写好了一个能够实现`差错控制`、`帧定位`的一个代码。然后发现自己代码写的太丑了，在加入流量控制的功能的时候，发现要改动的地方太多了，是因为自己的代码没有遵循`高内聚低耦合`的原则。那个很丑的代码我就不放了

然后就下定决心重写，因为要在多台设备之间同步，于是我使用了`github`来同步自己的代码

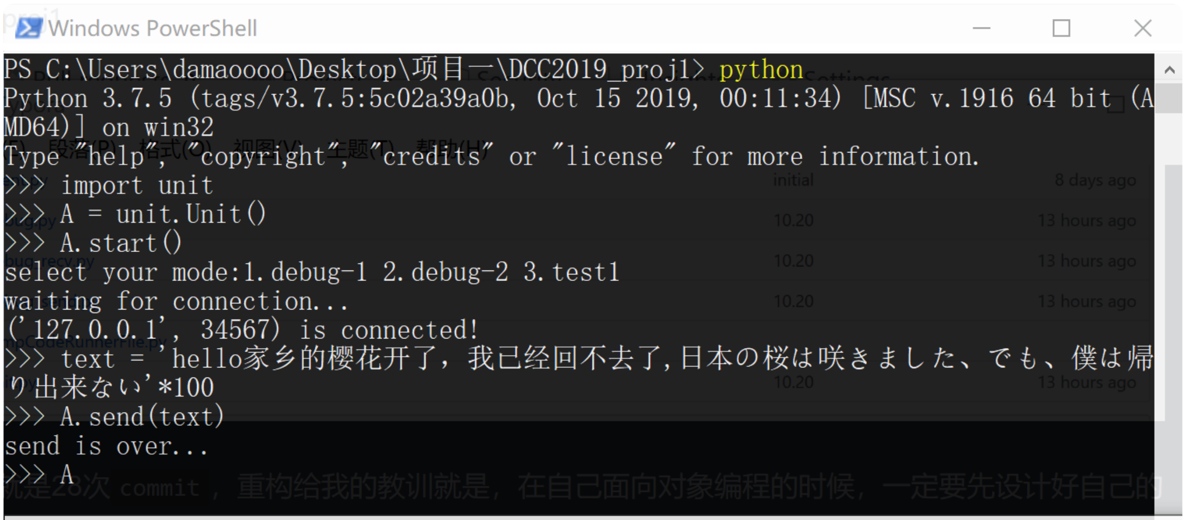


damaoooo 10.20 — Latest commit 0491227 13 hours ago

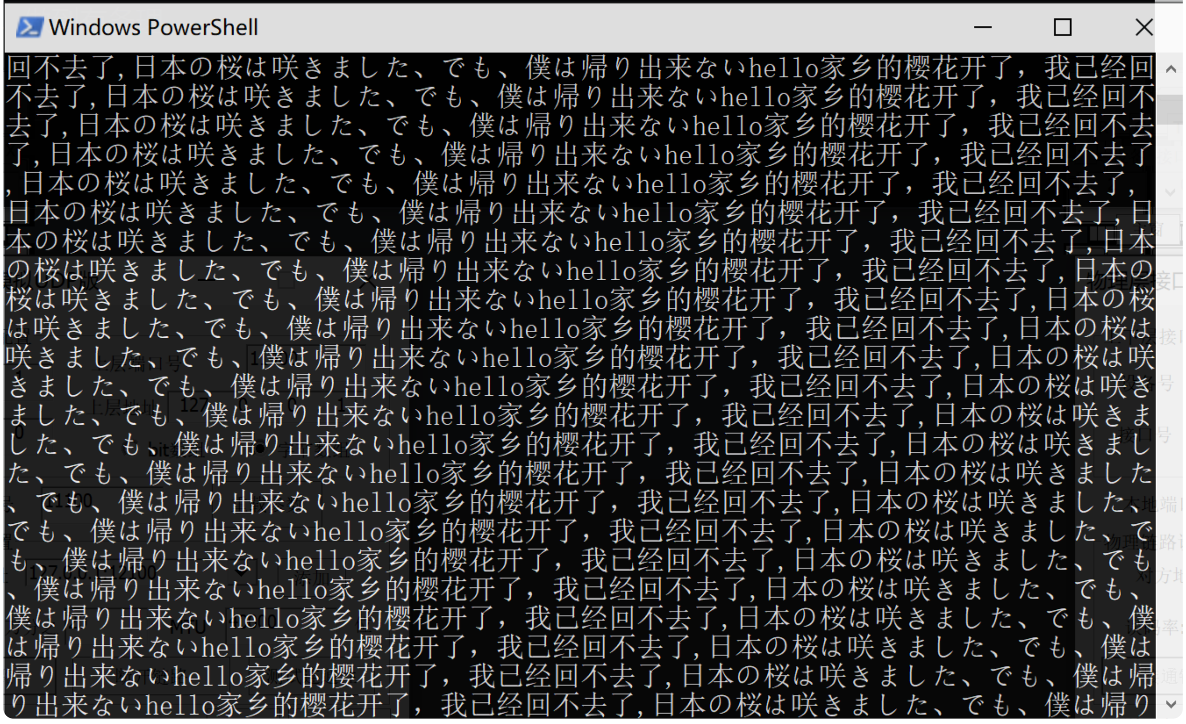
.idea	10.20	13 hours ago
__pycache__	10.20	13 hours ago
README.md		
client.py	initial	8 days ago
debug.py	10.20	13 hours ago
debug_recv.py	10.20	13 hours ago
debug_send.py	10.20	13 hours ago
tempCodeRunnerFile.py		
unit.py	10.20	13 hours ago

也不过就是28次`commit`，重构给我的教训就是，在自己面向对象编程的时候，一定要先设计好自己的流程图，做到有条理。

先上效果图，我们传输个100次中英日文夹杂的文字



```
PS C:\Users\damaoooo\Desktop\项目一\DCC2019_proj1> python
Python 3.7.5 (tags/v3.7.5:5c02a39a0b, Oct 15 2019, 00:11:34) [MSC v.1916 64 bit (A
MD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> import unit
>>> A = unit.Unit()
>>> A.start()
select your mode:1.debug-1 2.debug-2 3.test1
waiting for connection...
('127.0.0.1', 34567) is connected!
>>> text = 'hello家乡的樱花开了，我已经回不去了,日本の桜は咲きました、でも、僕は帰
り出来ない'*100
>>> A.send(text)
send is over...
>>> A
```



```
回不去了,日本の桜は咲きました、でも、僕は帰り出来ないhello家乡的樱花开了，我已经回
不去了,日本の桜は咲きました、でも、僕は帰り出来ないhello家乡的樱花开了，我已经回不
去了,日本の桜は咲きました、でも、僕は帰り出来ないhello家乡的樱花开了，我已经回不去
了,日本の桜は咲きました、でも、僕は帰り出来ないhello家乡的樱花开了，我已经回不去了
,日本の桜は咲きました、でも、僕は帰り出来ないhello家乡的樱花开了，我已经回不去了,
日本の桜は咲きました、でも、僕は帰り出来ないhello家乡的樱花开了，我已经回不去了,日
本の桜は咲きました、でも、僕は帰り出来ないhello家乡的樱花开了，我已经回不去了,日本
の桜は咲きました、でも、僕は帰り出来ないhello家乡的樱花开了，我已经回不去了,日本の
桜は咲きました、でも、僕は帰り出来ないhello家乡的樱花开了，我已经回不去了,日本の桜
は咲きました、でも、僕は帰り出来ないhello家乡的樱花开了，我已经回不去了,日本の桜は
咲きました、でも、僕は帰り出来ないhello家乡的樱花开了，我已经回不去了,日本の桜は咲
きました、でも、僕は帰り出来ないhello家乡的樱花开了，我已经回不去了,日本の桜は咲き
ました、でも、僕は帰り出来ないhello家乡的樱花开了，我已经回不去了,日本の桜は咲きま
した、でも、僕は帰り出来ないhello家乡的樱花开了，我已经回不去了,日本の桜は咲きまし
た、でも、僕は帰り出来ないhello家乡的樱花开了，我已经回不去了,日本の桜は咲きました
、でも、僕は帰り出来ないhello家乡的樱花开了，我已经回不去了,日本の桜は咲きました、
でも、僕は帰り出来ないhello家乡的樱花开了，我已经回不去了,日本の桜は咲きました、で
も、僕は帰り出来ないhello家乡的樱花开了，我已经回不去了,日本の桜は咲きました、でも
、僕は帰り出来ないhello家乡的樱花开了，我已经回不去了,日本の桜は咲きました、でも、
僕は帰り出来ないhello家乡的樱花开了，我已经回不去了,日本の桜は咲きました、でも、僕
は帰り出来ないhello家乡的樱花开了，我已经回不去了,日本の桜は咲きました、でも、僕は
帰り出来ないhello家乡的樱花开了，我已经回不去了,日本の桜は咲きました、でも、僕は帰
り出来ないhello家乡的樱花开了，我已经回不去了,日本の桜は咲きました、でも、僕は帰り
```

能传中文英文日文，我不会打韩文，不然我估计韩文也能传。

为了更好的使用，我把这个文件封装成了好几个API，只需要在python环境当中引入这个文件就可以使用了。

由于写一个服务端和写一个客户端，这样下来两部分的代码会有高度重复的部分，因此我就吧这些功能尽量整合在一块儿，作为一个网元，然后让用户来自行选择是发送还是接受。

使用方法

```
import unit#引入文件
A = unit.Unit()#实例化一个网元
A.start()#启动该网元，选择模式
A.send('text')#发送你的文本
A.recv()#接收文本
```

项目二结构设计

```
|-- Python 项目二
    |-- Unit 网元
        local->tuple
        dest->tuple
        |--start()
        |--send()
        |--receive()
        |-- tcpLayer网络层
            |--controlCenter()
            |--recvFrame()->list:frame
            |--sendFrame(str:text)->void
            |--checkXor(list:Frame)->bool:res
        |-- dataLayer 数据链路层
            |-- wrapChunk(rawFrames)->list:Frames
            |-- parseChunk(recvFrames)->list:checkedFrames
    |-- method 方法
        |--Oddcheck(list:Frames)->list:result[[row],[crowd]]
        |--bytes2Bin(bytestr:bytes)->str:rawBin
        |--bin2Frames(str:bindata)->list:Frame
        |--frames2Bin(list:Frames)->str:rawBin
        |--bin2Bytes(str:rawBin)->bytes:bytes
        |--text2Bytes(str:bytes)->str:bytes
        |--addXorCheck(list:Frame)->str:text
```

设计

按帧传输一帧8*8 bit + 2*8校验位 = 80bit，二维奇偶校验

层与层之间直接采用List的方式传帧，[[],[],[]]的方式

数据链路层:

text->text2Bytes()->bytes2Bin()->bin2Frames()->Frame->oddcheck->bin->bytes->send

recv->bin->Frame->oddcheck()->checkXor()->Frame2bin()->bin2bytes()->bytes2text()

网络层:

sender: send->recv->send

recver : recv->send : ack->recv()

建议

每个函数单独写出来，自己构造数据检查完备之后再粘贴进去

发送过程

根据误码率20/100000，传输5000个错1个，所以按照288为一帧太不划算，而且帧这个应该是在网络层包装，网络层中的极限数据bit位是4000个，我们就取3200个,也就是400字节（测试用例根本取不到）

'10110111'-->8bit为1字节

网络层

数据`hello world!你好啊嘤嘤嘤`

----转化为BYTES流---->

`b'hello world!\xe4\xbd\xa0\xe5\xa5\xbd\xe5\x95\x8a\xe5\x98\xa4\xe5\x98\xa4\xe5\x98\xa4'`

----转化为2进制字符串---->

'10101010101010101010101010101......10101010'

----按照8BIT来分成字节，然后过长则分帧---->

[['11111111','11111111','11111111',......(最多386个字节单元)......,'11111111'],

['11111111','11111111','11111111',......(最多386个字节单元)......,'11111111'],

['11111111','11111111','11111111',......(最多386个字节单元)......,'11111111'],

....

['11111111','11111111','11111111',......(最多386个字节单元)......,'11111111']]

----一次性发送8个帧---->

循环开始->8次

----选取一个帧添加头尾部---->

帧编号:`00-0f`,1字节

源IP:`ff.ff.ff.ff`,4字节，源端口:`ffff`，2字节

目的IP: `ff.ff.ff.ff`，4字节，目的端口: `ffff`,2字节

尾部：所有字节全部XOR一次，结果1字节

头部尾部算上一共要添加14字节

----发送一个帧到数据链路层---->

循环结束->8次

----进入接收信号模式---->

数据链路层

['11111111','11111111','11111111',......,'11111111']

----分校验单元---->

[['11111111','11111111'....(8个字节)......'11111111'],

['11111111','11111111'....(8个字节)......'11111111'],

....

['11111111','11111111'....(8个字节)......]]

其中[[],[],[],[]]为一个帧，里面的['111111','10101010',...（8个bit）...,'11100011']称为一个**校验单元**

----二维奇偶校验---->

[['11111111','10101010'....(10个字节，8字节数据，2字节校验)........'11111111'],

['11111111','10101010'....(10个字节，8字节数据，2字节校验)........'11111111'],

['11111111','10101010'....(10个字节，8字节数据，2字节校验)........'11111111'],

['11111111','10101010'....(10个字节，8字节数据，2字节校验)........'11111111']]

----转化为BYTES流---->

`\xef\xab\x0c\xc0......\xab\x3a`

----前后加上`\XEE\XFF`作为帧开始符，加上`\XFF\XEE`作为结束符---->

`\xee\xff\xef\xab\x0c\xc0......\xab\x3a\xff\xee`

----发送---->

接收过程

数据链路层

---- 接收BYTES流---->

`\xee\xff\xef\xab\x0c\xc0......\xab\x3a\xff\xee`

----按照 `\XEE\XFF` 开始 `\XFF\XEE` 作为帧定位符提取---->

`\xef\xab\x0c\xc0......\xab\x3a`

----判断是否为结束符 `\XFF\XEE\XDD\XFF\XFF\XDD\XEE\XFF` ---->

----转化为01流---->

'11111111000000000101010110101010......'

----按照8个字节为一个校验单位进行分帧处理---->

[['11111111','10101010'....(10个字节，8字节数据，2字节校验)........'11111111'],

['11111111','10101010'....(10个字节，8字节数据，2字节校验)........'11111111'],

['11111111','10101010'....(10个字节，8字节数据，2字节校验)........'11111111'],

['11111111','10101010'....(10个字节，8字节数据，2字节校验)........'11111111']]

----二维奇偶校验校错，去掉校验位---->

[['11111111','11111111'....(8个字节)......'11111111'],

['11111111','11111111'....(8个字节)......'11111111'],

....

['11111111','11111111'....(8个字节)......]]

----校验单位合并成为一个帧---->

['11111111','11111111','11111111',.......,'11111111']

----网络层---->

网络层

['11111111','11111111','11111111',........(最多400字节)........,'11111111']

----提取并去掉帧头尾信息---->如果XOR校验有误，BREAK，并发送ERR+帧序号

['11111111','11111111','11111111',.......(最多386字节)........,'11111111']

----转化为二进制字符串---->

'1111111101010101........010101010'

----转化为BYTES流,并缓存---->

`\xee\xff\xef\xab\x0c\xc0......\xab\x3a\xff\xee`

----接收到了结束符之后---->

----所有BYTES流解码成为数据---->

`hello world!你好啊嘤嘤嘤......`

流量控制

服务端

----发送8个帧---->接收状态信号---->决定重传8帧的起点或者是---->循环

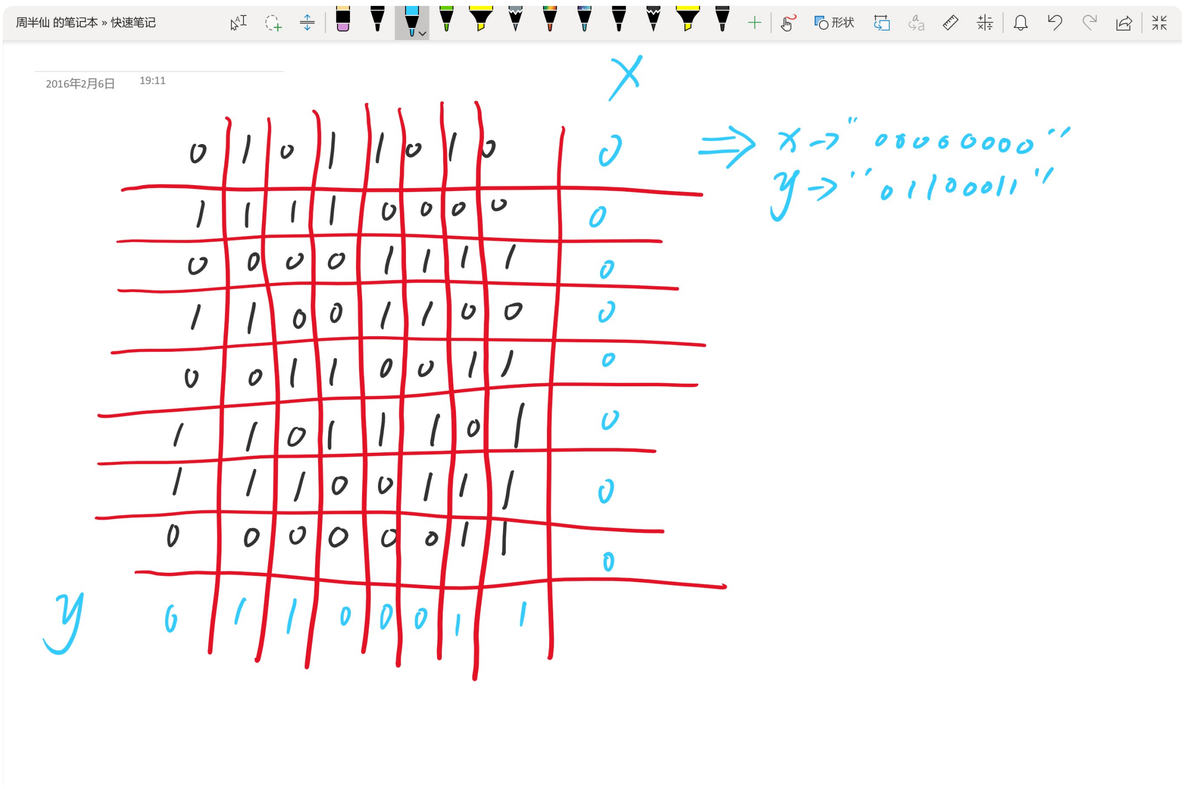
客户端

----开始接受一个帧（有很短的超时限制）---->检验是否为坏帧---->发送`ACK n`或者是`ARQ n`

物理层UDP模拟软件限制，无法使用TCP模式连接，只能用UDP来模拟TCP

图解设计

二维奇偶校验

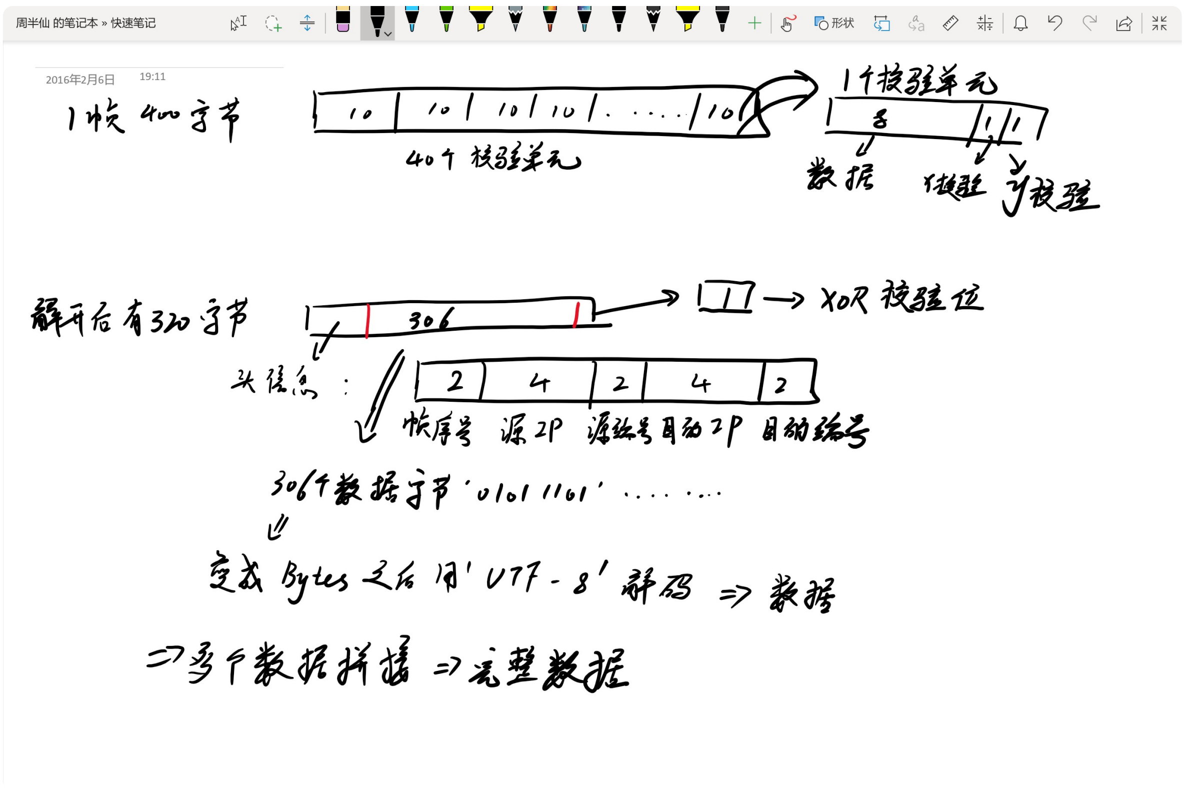

周半仙 的笔记本 » 快速笔记

2016年2月6日 19:11

$$
\begin{array}{c|cccccccc|c}
 & & & & & & & & & x \\
\hline
 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 \\
 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\
 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 \\
 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\
 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 \\
 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 0 \\
 & 1 & 1 & 1 & 0 & 0 & 1 & 1 & 1 & 0 \\
 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\
\hline
y & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 1 &
\end{array}
$$

⇒ x → "00000000"
y → "01100011"

帧结构


周半仙 的笔记本 » 快速笔记

2016年2月6日 19:11

1帧 400字节

解开后有320字节

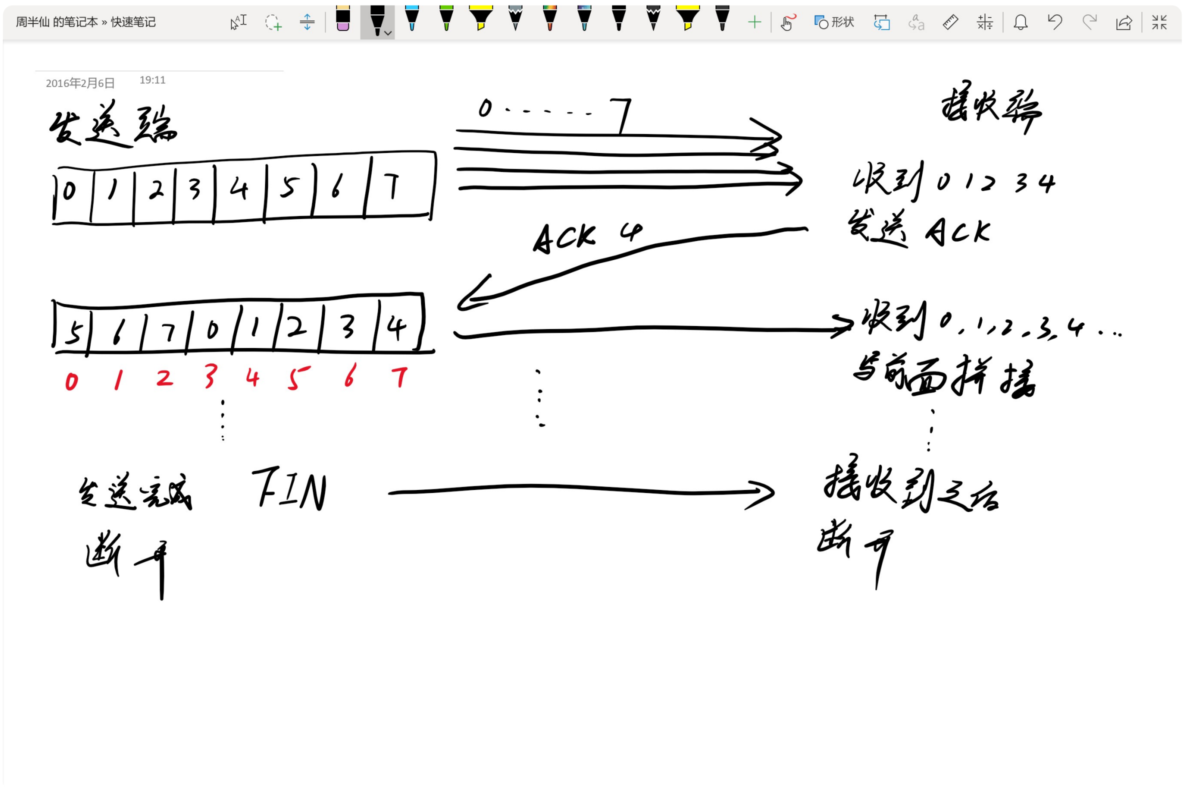
306个数据字节 '0101 1101' . . . .

⇓

变成 Bytes 之后 用 'UTF-8' 解码 ⇒ 数据

⇒ 多个数据拼接 ⇒ 完整数据

流量控制



贴代码

终于我把这个压缩到了300行以内，你说人生苦短对吧

```
from os import system
import time
from binascii import unhexlify
import socket
import select
class method():
    def findDiffrent(self,bytes1,bytes2):
        res = []
        for i in range(len(bytes1)):
            if(bytes1[i]!=bytes2[i]):
                res.append(i)
        return res
    def oddCheck(self,checkUnit):
        rowRes = [0]*8
        colunmRes = [0]*8
        for singleByteIndex in range(len(checkUnit)):
            #row:行  colunm:列
            listBytes = list(checkUnit[singleByteIndex])
            for singleBitIndex in range(len(listBytes)):
                num = int(listBytes[singleBitIndex])
                rowRes[singleByteIndex]^=num
                colunmRes[singleBitIndex]^=num
        return [''.join([str(x) for x in rowRes]),''.join([str(y) for y in
colunmRes])]
    def bytes2Bin(self,Bytes):
        binres = ''
        for i in Bytes:
            binres+=bin(i)[2:].zfill(8)
        return binres
    def bin2Frames(self,binText,Framelength):
        res = []
        leng = len(binText)
        tempFrame = []
        tempFramelen = 0
        while(leng>0):
            if(tempFramelen!=Framelength):
                tempFrame.append(binText[0:8])
                binText = binText[8:]
                tempFramelen+=1
                leng-=8
            else:
                res.append(tempFrame)
                tempFrame=[]
                tempFramelen=0
        if(res==[] or tempFrame!=res[-1]):
            res.append(tempFrame)
        return res
    def frames2Bin(self,frames):
        res = ''
        for oneFrame in frames:
            for i in oneFrame:
                res+=i
        return res
    def bin2Bytes(self,binText):
        res = b''
        length = len(binText)
        while(length>0):
            singleByte = unhexlify(hex(eval('0b'+binText[0:4]))
[2:]+hex(eval('0b'+binText[4:8]))[2:])
            res+=singleByte
            binText = binText[8:]
            length-=8
        return res
    def text2Bytes(self,text):
        return text.encode()
    def bytes2Text(self,bytesText):
        return bytesText.decode()
    def addXorCheck(self,Frame):
        sumxor = 0
        for onebyte in Frame:
            sumxor ^=eval('0b'+onebyte)
        return bin(sumxor)[2:].zfill(8)
    def direction(self,text,keystart,keyend):
        start = text.find(keystart)
        end = text.find(keyend)
        if(start!=-1 and end!=-1):
            return text[start+len(keystart):end]
        else:
            return -1
class Unit(method):
    mode,local,dest,tcp,datalink = 0,0,0,0,0
    sk = socket.socket(socket.AF_INET,socket.SOCK_DGRAM)
    st = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
    st.settimeout(30)
    conn,addr = 0,0
    sk.settimeout(30)
    def start(self):
        mode = int(input('select your mode:1.debug-1 2.debug-2 3.test'))
        if(mode == 3):
            localIp = input('input your IP->')
            localPort = input('input your Port->')
            self.local = (localIp,int(localPort))
            self.sk.bind(self.local)
            destIp = input('input opposite IP->')
            destPort = input('input opposite Port->')
            self.dest = (destIp,int(destPort))
        elif(mode == 1):
            self.local = ('127.0.0.1',11400)
            self.dest = ('127.0.0.1',11100)
            self.st.bind(('127.0.0.1',34566))
            self.st.listen(5)
            print('waiting for connection...')
            self.conn,self.addr = self.st.accept()
            print(self.addr,'is connected!')
            self.sk.bind(self.local)
        elif(mode == 2):
            self.local = ('127.0.0.1',12400)
            self.dest = ('127.0.0.1',12100)
            self.st.bind(('127.0.0.1',34567))
            print('trying to connect...')
            self.st.connect(('127.0.0.1',34566))
            self.sk.bind(self.local)
        self.tcp =
self.tcpLayer(self.local,self.dest,self.sk,self.st,self.conn)
    class tcpLayer(method):
        frameLength,frameNumber,local,dest,sendSocket,statusSocket,conn =
0,0,0,0,0,0,0
        def
__init__(self,local,dest,sendsocket,statussocket,conn,*argc,**kwargs):
            self.local = local
            self.dest = dest
            self.sendSocket = sendsocket
            self.statusSocket = statussocket
            self.conn = conn
        def getHeaders(self,sourceIp,sourcePort,destIp,destPort,frameNumber):
            res =  []
            res.append(bin(frameNumber)[2:].zfill(8))
            for i in sourceIp.split('.'):
                res.append(bin(int(i))[2:].zfill(8))
            portbin = bin(sourcePort)[2:].zfill(16)
            res.append(portbin[0:8])
            res.append(portbin[8:])
            for i in destIp.split('.'):
                res.append(bin(int(i))[2:].zfill(8))
            portbin = bin(destPort)[2:].zfill(16)
            res.append(portbin[0:8])
            res.append(portbin[8:])
            return res

        def checkXor(self,oneFrame):
            xorCheck = method.addXorCheck(self,oneFrame[:-1])
            recvXorCheck = oneFrame.pop(-1)
            return (xorCheck == recvXorCheck)
        def wrapChunk(self,oneFrame):
            chunks = []
            length = len(oneFrame)
            while(length>0):
                chunks.append(oneFrame[0:8])
                oneFrame = oneFrame[8:]
                length-=8
            for oneChunkIndex in range(len(chunks)):
                chunks[oneChunkIndex]+=
(method.oddCheck(self,chunks[oneChunkIndex]))
            return chunks

        def parseChunk(self,oneFrame):
            chunks = []
            while(oneFrame!=[]):
                oneChunk = oneFrame[0:10]
                chunks.append(oneChunk)
                oneFrame = oneFrame[10:]
            for chunkIndex in range(len(chunks)):
                oddCheckRecv = chunks[chunkIndex][-2:]
                data = chunks[chunkIndex][:-2]
                selfCheck = self.oddCheck(data)
                if(selfCheck == oddCheckRecv):
                    chunks[chunkIndex]=data
                    continue
                else:
                    diff =
[self.findDiffrent(oddCheckRecv[0],selfCheck[0]),self.findDiffrent(oddCheckRe
cv[1],selfCheck[1])]
                    x,y = diff[0],diff[1]
                    if(len(x)+len(y)==1):
                        chunks[chunkIndex] = data
                        continue
                    elif(len(x)==len(y)):
                        for i in range(len(x)):
                            data[x[i]][y[i]]^=1
                chunks[chunkIndex] = data
            res = []
            for i in chunks:
                res +=i
            return res

        def sendControlCenter(self,oneFrame):
            sendBin = self.frames2Bin(oneFrame)
            sendBytes = b'\xee\xff'+self.bin2Bytes(sendBin)+b'\xff\xee'
            self.tcp.sendSocket.sendto(sendBytes,self.dest)

        def recvControlCenter(self,rawBin):
            sourceIp,sourcePort,destIp,destPort = 0,0,0,0
            frameNumber,xorCheckRecv = 0,0
            bytesText = b''
            Frames = self.bin2Frames(rawBin,400)
            status = []
            for oneFrames in Frames:
                afterParse = self.tcp.parseChunk(oneFrames)
                frameNumber = eval('0b'+afterParse[0])
                afterParse.pop(0)
                sourceIp,sourcePort = afterParse[0:4],afterParse[4:6]
                sourceIp = '.'.join([str(eval('0b'+x)) for x in sourceIp])
                sourcePort = eval('0b'+sourcePort[0]+sourcePort[1])
                afterParse = afterParse[6:]
                destIp,destPort = afterParse[0:4],afterParse[4:6]
                destIp = '.'.join([str(eval('0b'+x)) for x in destIp])
                destPort = eval('0b'+destPort[0]+destPort[1])
                afterParse = afterParse[6:]
                isBad = self.tcp.checkXor(afterParse)
                #print(isBad,frameNumber)
                if(isBad==False):
                    status.append('ERR.'+str(frameNumber))
                    return bytesText,status
                binText = self.frames2Bin(afterParse)
                bytesText += self.bin2Bytes(binText)
                status.append('ACK.'+str(frameNumber))
            return bytesText,status


    def send(self,Text):
        frameNumber = 0
        bytesText = method.text2Bytes(self,Text)
        binText = method.bytes2Bin(self,bytesText)
        Frames = self.bin2Frames(binText,306)#40 Unit,400-40*2-14=
        dataToSend = []
        while(Frames!=[]):
            if(frameNumber<min(8,len(Frames))):
                headers = 
self.tcp.getHeaders(self.local[0],self.local[1],self.dest[0],self.dest[1],fra
meNumber)
                frame = Frames[frameNumber]
                xorRes = [self.addXorCheck(frame)]
                wrappedFrames = headers+frame+xorRes
                wrappedFrames = self.tcp.wrapChunk(wrappedFrames)
                dataToSend+=wrappedFrames
                frameNumber+=1
            else:
                self.tcpLayer.sendControlCenter(self,dataToSend)
                dataToSend = []
                status = self.tcp.conn.recv(40000).decode()
                status = status.split('|')[-2].split('.')
                if(status[0]=='ERR'):
                    Frames = Frames[int(status[1]):]
                elif(status[0]=='ACK'):
                    Frames = Frames[int(status[1])+1:]
                frameNumber = 0
        self.sk.sendto(b'\xee\xff\xad\xff\xda\xff\xee',self.dest)
        print('send is over...')
    def recv(self):
        rawBytes = b''
        bytesText = b''
        self.sk.settimeout(1000)
        firstRecv = self.sk.recv(50000)
        while(1):
            rawBytes = b''
            if(firstRecv!=b''):
                rawBytes+=firstRecv
                firstRecv = b''
            else:
                self.sk.settimeout(2)
                try:
                    rawBytes += self.sk.recv(50000)
                except:
                    print('')
            rawBins = self.bytes2Bin(rawBytes)
            afterDirect = self.direction(rawBins,bin(0xeeff)[2:],bin(0xffee)
[2:])
            if(self.bin2Bytes(afterDirect)==b'\xad\xff\xda'):
                break
            onebytesText,status = 
self.tcpLayer.recvControlCenter(self,afterDirect)
            self.st.send(('|'+'|'.join(status)+'|').encode())
            bytesText+=onebytesText
            #print(bytesText.decode())
        return bytesText.decode()

if(__name__ == '__main__'):
    A = Unit()
    A.start()
    #text = 'Thomas Jefferson and James Madison met in 1776. Could it have
been any other year? They worked together starting then to further American
Revolution and later to shape the new scheme of government. From the work
sprang a friendship perhaps incomparable in intimacy1 and the trustfulness of
collaboration2 and induration. It lasted 50 years. It included pleasure and
utility but over and above them, there were shared purpose, a common end and
an enduring goodness on both sides. Four and a half months before he died,
when he was ailing3, debt-ridden, and worried about his impoverished4 family,
Jefferson wrote to his longtime friend. His words and Madison  s reply remind
us that friends are friends until death. They also remind us that sometimes a
friendship has a bearing on things larger than the friendship itself, for has
there ever been a friendship of greater public consequence than this one? The
friendship which has subsisted5 between us now half a century, the harmony of
our po1itical principles and pursuits have been sources of constant happiness
to me through that long period. If ever the earth has beheld6 a system of
administration conducted with a single and steadfast7 eye to the general
interest and happiness of those committed to it, one which, protected by
truth, can never known reproach, it is that to which our lives have been
devoted8. To myself you have been a pillar of support throughout life. Take
care of me when dead and be assured that I should leave with you my last
affections. A week later Madison replied- You cannot look back to the long
period of our private friendship and political harmony with more affecting
recollections than I do. If they are a source of pleasure to you, what aren
t they not to be to me? We cannot be deprived of the happy consciousness of
the pure devotion to the public good with Which we discharge the trust
committed to us and I indulge a confidence that sufficient evidence will find
in its way to another generation to ensure, after we are gone, whatever of
justice may be withheld9 whilst we are here.  '*10
    #搞一个变态长的数据来进行传输测试
    text = input('input what you want to say:')
    A.send(text)
    A.st.close()
```