大连理工大学

计算机网络实验报告

课程名称：计算机网络实验

学院（系）：电子信息与电气工程学部

专　　业：计算机科学与技术

班　　级：电计 1905 班

学　　号：201961203

学生姓名：陈豪宇

2021 年 12 月 8 日

实验项目列表

序号	实验项目名称	分数占比	分数
1	小试牛刀	20	
2	进阶难度	30	
3	创新学习	30	
4	报告	20	
5			
6			
7			
8			
9			
10			
11			
12			
13			
14			
15			
16			
17			
18			
总计			

大连理工大学实验报告

学院（系）：电子信息与电气工程学部 专业：计算机科学与技术 班级：电计 1905

姓　　名：陈豪宇　　学号：201961203　　组：陈豪宇

实验时间：2021.12.7 实验室：_____成绩：_____

实验一 在一台主机上通过回环地址实现客户-服务器间的 socket 通信

一、实验要求

分别建立服务器和客户机，实现客户机和服务器的连接和通信，从客户机向服务器传输你的个人信息和任意文件信息，在服务器上显示你发送的信息和文件信息。

二、实验内容

首先创建服务器，导入一些 python 中使用 socket 编程的相关库。

```
from PyQt5 import uic
from PyQt5.QtWidgets import QApplication
from socket import *
from threading import Thread
```

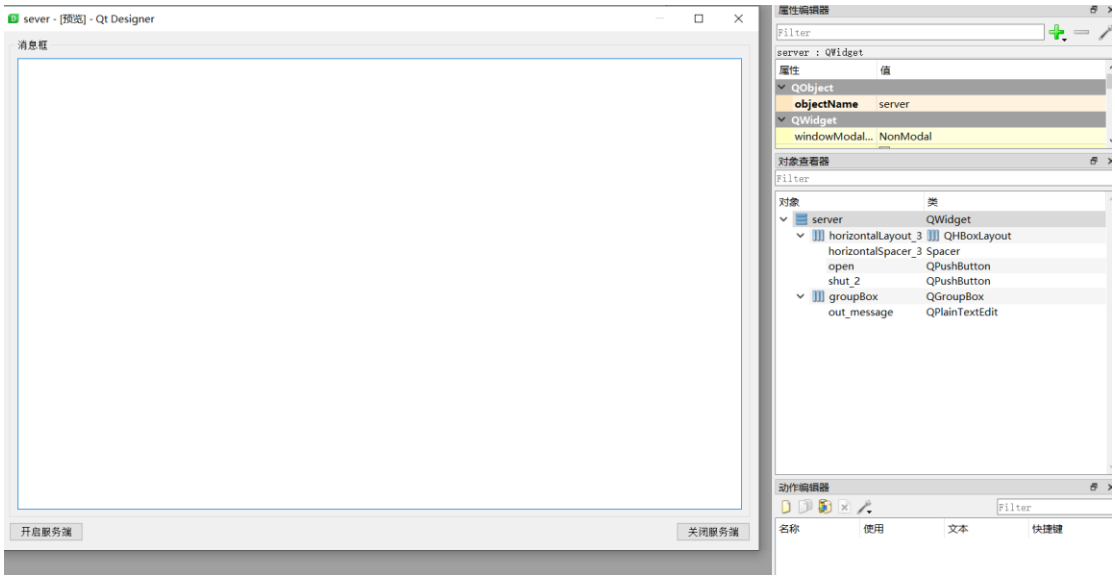
定义 IP 号为空，这样可以接收从任意 IP 地址传来的信息或请求，再定义端口号为 50001，只有客户机绑定相同的端口号才可以访问服务器。

```
# 主机地址为空字符串，表示绑定本机所有网络接口ip地址
IP = ''
# 端口号
PORT = 50001
# 定义一次从socket缓冲区最多读入1024个字节数据
BUFLEN = 1024
```

在 python 中定义 Server 的类，先写其中的__init__初始化函数，在初始化函数中，现从上而下解释语句的含义，显示绑定了本服务器所对应的 ui 图形界面，之后中 ui 界面中的几个按钮通过 clicked.connect 函数对应到本类中几个函数 Estlabish、Shut_2，其中 Estlabish 函数为该服务器的初始化，Shut_2 函数为关闭该 socket。dataSocket 和 serverSocket 是类中的变量，类中函数都可以访问。

```
class Server():
    def __init__(self):
        # data=Data()
        self.ui = uic.loadUi('./first_server.ui')
        self.ui.open.clicked.connect(self.Establish)
        self.ui.shut_2.clicked.connect(self.Shut_2)
        self.dataSocket = None
        self.serverSocket = None
        count = 0
```

创建的第一个 ui 界面预览如下，Plain Text Edit 命名为 out_message，左下角和右下角的 Botton 分别为 open、shut_2，这里的名字对应着类中调用 ui 函数库时所对应的按键名称。



此为类中的 Establish 函数，目的是实例化 socket 对象，参数 AF_INET 表示该 socket 网络层使用 IP 协议，参数 SOCK_STREAM 表示该 socket 传输层使用 tcp 协议，再通过全局变量 IP 和 PORT 绑定地址和端口 socket，设置监听状态，最多接受 8 个排队的客户端请求，最后三行代码为创建子线程，防止在运行主线程时出现主线程停顿在监听状态而导致点击开启客户端按键后 ui 界面卡死或者停止运行的情况出现。target 参数指定新线程要执行的函数，如果新线程函数需要参数，在 args 里面填入参数，注意参数是元组。

```
def Establish(self,count):
    # 实例化一个socket对象
    # 参数 AF_INET 表示该socket网络层使用IP协议
    # 参数 SOCK_STREAM 表示该socket传输层使用tcp协议
    self.serverSocket = socket(AF_INET, SOCK_STREAM)
    # socket绑定地址和端口
    self.serverSocket.bind((IP, PORT))
    # 使socket处于监听状态，等待客户端的连接请求
    # 最多接受8个等待连接的客户端
    self.serverSocket.listen(8)
    # target 参数 指定 新线程要执行的函数
    # 注意，这里指定的函数对象只能写一个名字，不能后面加括号，
    # 如果加括号就是直接在当前线程调用执行，而不是在新线程中执行了
    # 如果 新线程函数需要参数，在 args里面填入参数
    # 注意参数是元组， 如果只有一个参数，后面要有逗号，像这样 args=('参数1',)
    th = Thread(target=self.start, args=(count,))
    # 设置新线程为daemon线程
    th.setDaemon(True)
    # 执行start 方法，就会创建新线程，
    # 并且新线程会去执行入口函数里面的代码。
    # 这时候 这个进程 有两个线程了
    th.start()
```

运行至 Estlabish 函数末时，转去执行它的子线程，该 start 函数为监听客户端的连接请求，并打印出请求连接的相关信息，在末尾继续开一个子线程来去执行接受信息的死循环，防止该线程在这里卡住，以导致程序运行出错。

```
def start(self,count):
    # 此处采用阻塞式监听
    print('等待客户端的连接...')
    sock, addr = self.serverSocket.accept()
    count = count + 1
    print('接受一个客户端连接:', addr,'  目前在线人数为：',count)
    th = Thread(target=self.recv,args=(sock,))
    th.setDaemon(True)
    th.start()
```

此函数为以死循环来接收对方客户机发送的信息，将信息存储在 BUFLEN 变量中，如果没有读取到信息则说明对方关闭了连接，所以应该停止该死循环然后退出程序，如果信息不为空的话，则在该服务器的 out_message 界面显示收到对方信息并打印信息的内容，再向客户机发送服务端已收到的消息。

```
def recv(self,dataSocket):
    while True:
        # 尝试读取对方发送的消息
        # BU FLEN 指定从接收缓冲里最多读取多少字节
        recved = dataSocket.recv(BUFLEN)
        # 如果返回空bytes，表示对方关闭了连接
        # 退出循环，结束消息收发
        if not recved:
            break
        # 读取的字节数据是bytes类型，需要解码为字符串
        info = recved.decode()
        self.ui.out_message.appendPlainText(f'收到对方信息：  {info}')
        # 发送的数据类型必须是bytes，所以要编码
        dataSocket.send(f'服务端接收到了信息 {info}'.encode())
    dataSocket.close()
```

此 Shut_2 函数为调用 close 函数来关闭 socket 的连接，在程序的最后为运行 qt5 的相关程序，创建 Server 对象，调用运行 ui 界面。

```
73      def Shut_2(self):
74          # 服务端也调用close()关闭socket
75
76          self.serverSocket.close()
77
78  app = QApplication([])
79  stats = Server()
80  stats.ui.show()
81  app.exec_()
```

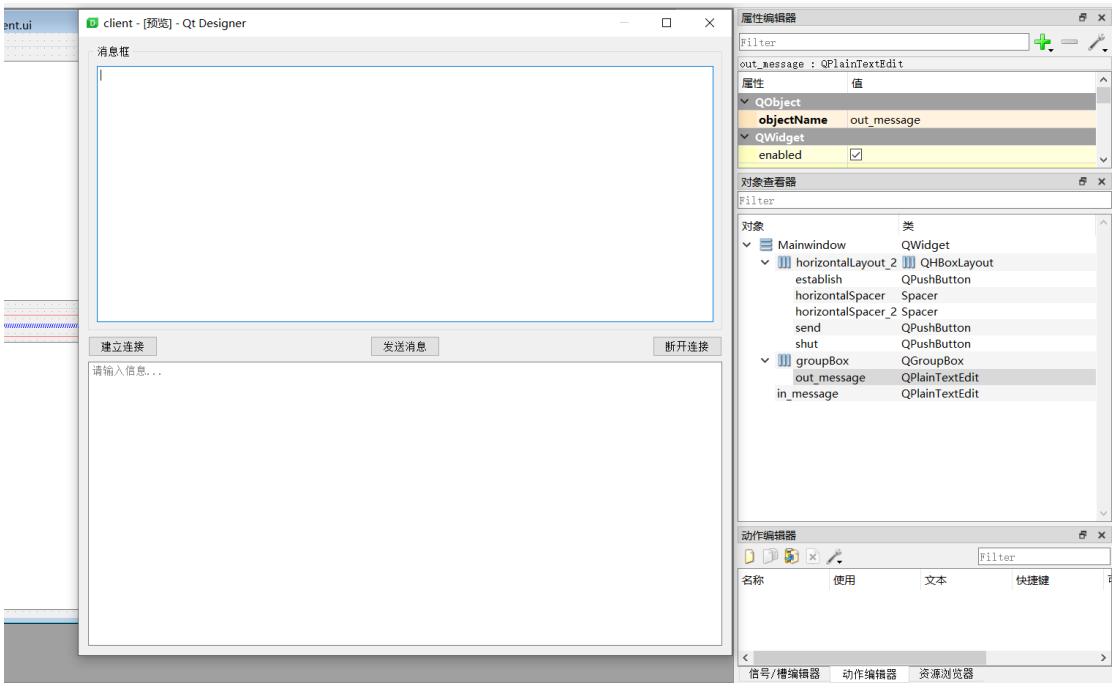
此为创建客户机的相关程序，首先一如既往的导入库的函数，并设置 IP 为 127.0.0.1，此为通过回环地址实现客户-服务器间的 socket 通信的 IP 地址，端口号注意到要与服务器保持一致，定义一次从 socket 缓冲区最多读入 1024 个字节数据。

```
from PyQt5 import uic
from PyQt5.QtWidgets import QApplication
from socket import *
IP = '127.0.0.1'  # 10.5.228.94  127.0.0.1
SERVER_PORT = 50001
BUFLEN = 1024

```

同服务器的相关代码结构，也先是载入客户机的 ui 文件，绑定各个按钮，并初始化类中的变量。

```
class Client():
    def __init__(self):
        self.ui = uic.loadUi('./first_client.ui')
        self.ui.establish.clicked.connect(self.getConnect)
        self.ui.send.clicked.connect(self.Send)
        self.ui.shut.clicked.connect(self.Shut)
        self.dataSocket = None

```

服务器的预览界面如下图所示，由两个文本框和三个按钮组成。

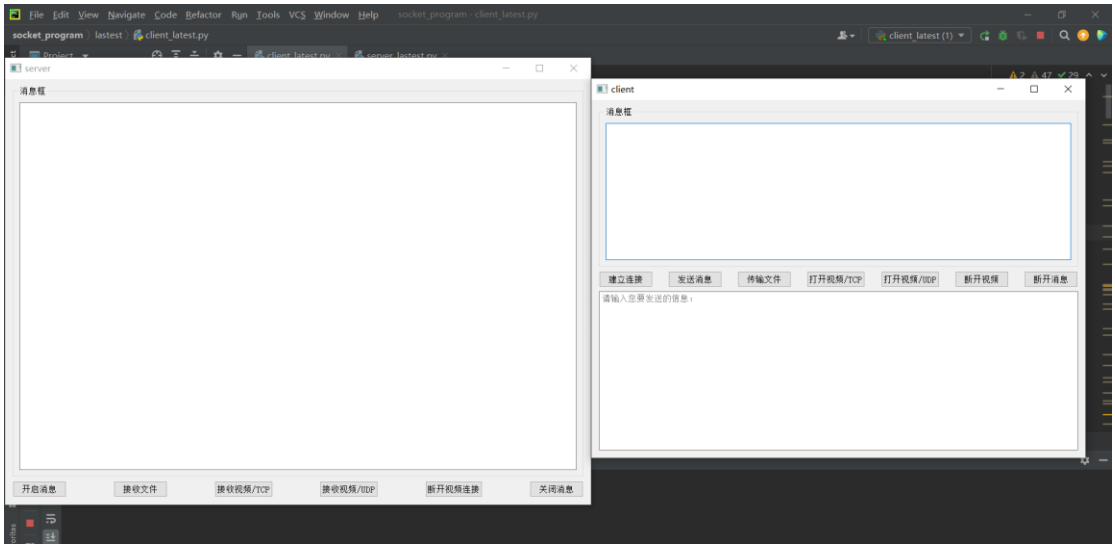


如下图所示为客户机的剩余部分代码，包含三个类中的函数，其中getConnect 函数为实例化 socket 对象并指明所用的协议为 TCP；Send 函数为读取下面的文本款中的内容并将它通过 socket 的 send 函数去发送至服务器，发送之后便清空发送窗口已发送的消息内容，再每次发送完成之后服务器都会回复消息表示已收到，所以需要接收并打印在 ui 界面的上部分的文本框中，如果发送的消息没有收到回应则证明对方的服务器未与自己连接；最后一个 Shut 的类函数就是关闭 socket 连接，之后便是主程序中通过类来实例化对象并调用相关的函数来执行 ui 界面和主程序的代码。由于在客户机端未出现监听或者是死循环来接收数据，因此在客户机的消息发送端未采用像服务器的子线程的方法。

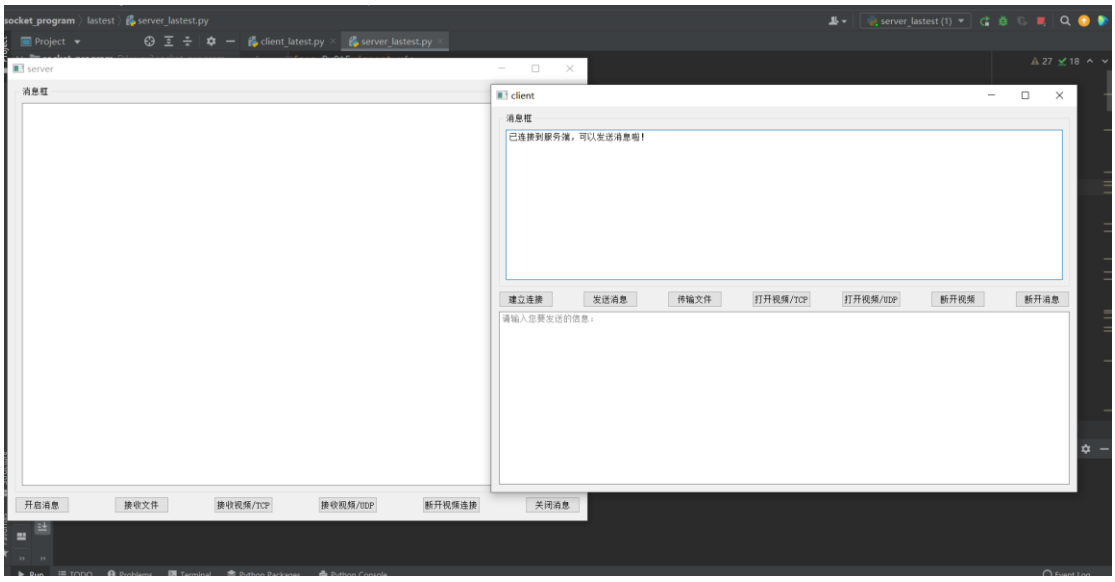
```
    def getConnect(self):

        # 实例化一个socket对象，指明协议
        self.dataSocket = socket(AF_INET, SOCK_STREAM)
        # 连接服务端socket
        self.dataSocket.connect((IP, SERVER_PORT))
    def Send(self):
        # 接受框中的消息
        toSend = self.ui.in_message.toPlainText()
        # 接收完成后，清空消息
        self.ui.in_message.clear()
        # 发送消息，也要编码为 bytes
        self.dataSocket.send(toSend.encode())
        # 等待接收服务端的消息
        recved = self.dataSocket.recv(BUFLEN)
        # 如果返回空bytes，表示对方关闭了连接
        if not recved:
            return
        self.ui.out_message.appendPlainText(recved.decode())

    def Shut(self):
        self.dataSocket.close()
app = QApplication([])
stats = Client()
stats.ui.show()
app.exec_()
```

三、实验结果与分析

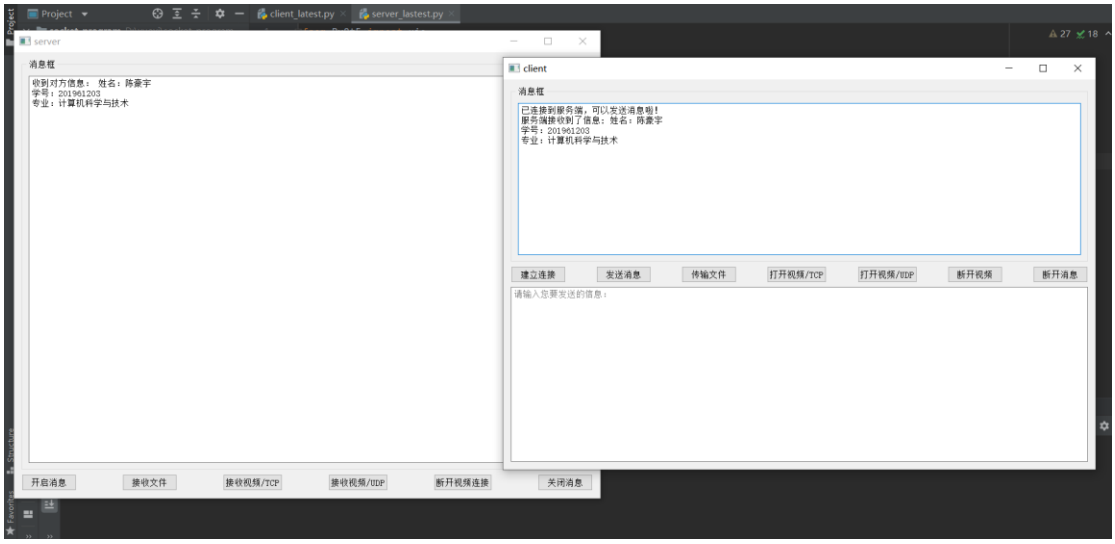
由于在最后一个实验的部分，自己整合了发送消息和视频传输的功能为一体的完整程序，并且在最后完善了相关的代码功能，因此演示阶段的截图将采用最后一个实验的程序截图，Server 和 Client 界面如下所示：



首先点击 server 窗口的“开启消息”便可以创建好服务器端的 socket，然后点击 client 窗口的“建立连接”即可以连接上服务器，并在客户的消息框中提示已连接到服务器，具体界面如下图所示：



在客户机的下方文本框中输入自己将要发送的消息，并点击“发送消息”按键便可以调用客户机中类的发送的相关函数将消息发送过去，如下图所示为发送自己的相关信息：



只要让客户机连接上客户机就可以继续发送消息，但是一次性发送消息的长度受程序中的定义的 BUFLEN 控制。

四、讨论、感想

在本次实验中，自己学习到了 socket 相关的基础编程方法，以及创建简易的 ui 界面，并将 ui 界面上面的按键、文本框等连接到程序中，在刚开始写服务端时遇见程序停止运行情况，之后通过查阅相关资料了解到多线程的方法，可以通过多线程来解决主线程中的等待接收信息的情况，参考网站放置于本报告的最后一个实验的末尾，自己也学习到了 python 编程的相关技巧和方法，之后也可以考虑继续优化 ui 的界面，增加 logo 图标以及颜色等等，此次实验收获良多。

大连理工大学实验报告

学院（系）：电子信息与电气工程学部 专业：计算机科学与技术 班级：电计 1905

姓　　名：陈豪宇　　学号：201961203　　组：陈豪宇

实验时间：2021.12.7 实验室：_____成绩：_____

实验二 实现两台主机在局域网内的客户-服务器 socket 通信

一、实验要求

实现 TCP 通信，实现能够在局域网中进行两台主机之间的 socket 通信。

二、实验内容

实验内容与上述相似，将客户端的目的 IP 设置为局域网中的另一台正在接收消息的服务器的 IP，同时需要更改服务器的监听状态为 while True 的死循环，提取每台客户机的地址信息来区别每个客户机发送的消息，他的基本群聊原理为某一个客户机向服务器发送消息，在自己的聊天页面打印出自己的消息，同时服务器识别该地址和 socket 连接，再向除此客户机以外的所有客户机都转发该消息，并且告诉他们这是由哪个 IP 发来的消息，具体代码解释见下。

首先还是服务器的部分，导入 python 库函数和初始化变量，不做过多解释。

```
1   from PyQt5 import uic
2   from PyQt5.QtWidgets import QApplication
3   from threading import Thread
4   import cv2
5   from socket import *
6   import numpy
7   # 主机地址为空字符串，表示绑定本机所有网络接口ip地址
8   IP = ''
9   # 端口号
10  PORT = 50001
11  # 定义一次从socket缓冲区最多读入60000个字节数据
12  BUFLEN = 60000
13  Send_BUFLEN = 1024
```

绑定按钮，将“开启聊天室”按钮绑定到类的 Chat 函数，这里设置了服务器也可以发送消息，向每一个已经连接的客户机发送消息并且有着不一样的标识，但是在实际操作中出现 Bug 因此后续测试没有使用服务器的 Send 功能。

```
class Server():
    def __init__(self):
        # data=Data()
        self.ui = uic.loadUi('./server_latest.ui')
        self.ui.establish.clicked.connect(self.Establish)
        self.ui.shut.clicked.connect(self.Shut)
        self.ui.open_videoTCP.clicked.connect(self.Video_TCP)
        self.ui.open_videoUDP.clicked.connect(self.Video_UDP)
        self.ui.shut_video.clicked.connect(self.Shut_Video)
        self.ui.open_chat.clicked.connect(self.Chat)
        self.ui.send.clicked.connect(self.Server_Send)
        self.dataSocket = None
        self.serverSocket = None
        self.TCPsocket = None
        self.UDPsocket = None
        self.addr = []
        self.address_all = []
        self.count = 0

```

这里初始化与上述通过回环地址实现客户-服务器间的 socket 通信的初始化；类似，只是最后需要开启一个子线程来监听每一个客户端的请求，否则只能连接一个客户端，其他客户请求时会被积极拒绝。

```
    def Chat(self):
        self.count = 0
        # 实例化一个socket对象
        self.serverSocket = socket(AF_INET, SOCK_STREAM)
        # socket绑定地址和端口
        self.serverSocket.bind((IP, PORT))
        # 使socket处于监听状态，等待客户端的连接请求
        self.serverSocket.listen(8)
        th = Thread(target=self.Chat_start, args=())
        # 设置新线程为daemon线程
        th.setDaemon(True)
        th.start()

```

Chat_start 函数实现的功能时单独使用一个子线程来监听客户端的请求，如果有客户端发送了加入群聊的请求，则服务器响应接受并且保存输出他的相关信息，并回复该客户“已连接到群聊天，可以发送消息啦！”，在最后再开子线程来接受该客户端发送的信息，由此进行死循环。

```
104     def Chat_start(self):
105         # 此处采用阻塞式监听
106         while True:
107             self.ui.out_message.appendPlainText('等待其他用户的连接...')
108             dataSocket, address = self.serverSocket.accept()
109             self.count = self.count + 1
110             self.addr.append(dataSocket)
111             self.address_all.append(address[0])
112             print('接受一个用户连接:', address,'  目前在线人数为：',self.count)
113             dataSocket.send('已连接到群聊天，可以发送消息啦！'.encode())
114             th = Thread(target=self.Chat_recv,args=(dataSocket,))
115             th.setDaemon(True)
116             th.start()
117
```

该函数的功能为接收某一个客户机发送的消息，并处理这个客户的地址等信息整合后转发给除此客户以外的所有客户该条聊天记录，也在客户端显示该消息。

```
118     def Chat_recv(self,dataSocket):
119         while True:
120             # 尝试读取对方发送的消息
121             # BUFLEN 指定从接收缓冲里最多读取多少字节
122             recved = dataSocket.recv(Send_BUFLEN)
123             # 如果返回空bytes，表示对方关闭了连接
124             # 退出循环，结束消息收发
125             if not recved:
126                 break
127             # 读取的字节数据是bytes类型，需要解码为字符串
128             info = recved.decode()
129             count_temp = 0
130             for iii in self.addr:
131                 if iii == dataSocket:
132                     break
133                 count_temp = count_temp + 1
134             # print(self.address_all[count_temp])
135             temp_str = str(self.address_all[count_temp])+f'的信息：{info}'
136             self.ui.out_message.appendPlainText(temp_str)
137             # 发送的数据类型必须是bytes，所以要编码
138             for ii in self.addr:
139                 if ii != dataSocket:
140                     ii.send(temp_str.encode())
141         dataSocket.close()
142
```

接下来是客户机的代码，Chat_Connect 函数为实例化 socket 对象，连接群聊的服务器，如果连接成功则服务器会回复一条通知消息，之后开启子线程持续接收服务器发送的消息，并将其打印输出在自己的客户机消息框中。

```
76      def Chat_Connect(self):
77          self.choose = 2
78          # 实例化一个socket对象，指明协议
79          self.dataSocket = socket(AF_INET, SOCK_STREAM)
80          # 连接服务端socket
81          self.dataSocket.connect((IP, SERVER_PORT))
82          # self.ui.out_message.appendPlainText('正在连接上群聊服务器 ip:', IP)
83          print('正在连接上群聊服务器 ip:', IP)
84          # 接收连接成功的消息
85          recved = self.dataSocket.recv(BUFLEN)
86          self.ui.out_message.appendPlainText(recved.decode())
87          th = Thread(target=self.Chat_start, args=())
88          # 设置新线程为daemon线程
89          th.setDaemon(True)
90          th.start()
91
92      def Chat_start(self):
93          while True:
94              # 尝试读取对方发送的消息
95              # BUFLEN 指定从接收缓冲里最多读取多少字节
96              recved = self.dataSocket.recv(BUFLEN)
97              # 如果返回空bytes，表示对方关闭了连接
98              # 退出循环，结束消息收发
99              if not recved:
100                 break
101             # 读取的字节数据是bytes类型，需要解码为字符串
102             info = recved.decode()
103             self.ui.out_message.appendPlainText(info)
104             # 发送的数据类型必须是bytes，所以要编码
105         self.dataSocket.close()
```

Client › Send() › else

由于在前一个问题中使用了 Send 函数，并且绑定了“发送消息的按钮”，因此针对是两个设备的通信和群聊的功能不一样，群聊的时候一直在监听服务器的消息，因此 send 函数不能接收任何消息，使用 self.choose 来辨识出客户机正在进行的哪一种的通信方式。

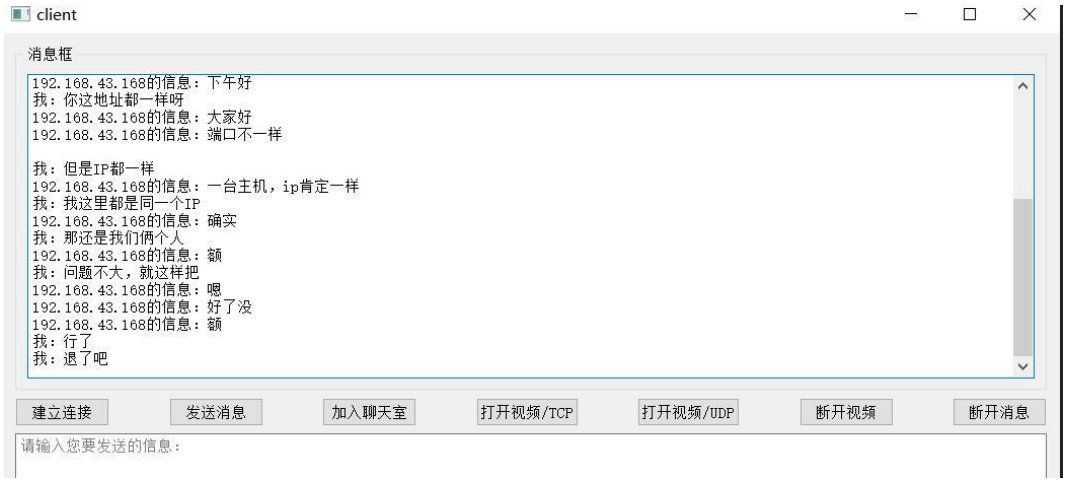
```
59          elif self.choose == 2:
60              # 接受框中的消息
61              toSend = self.ui.in_message.toPlainText()
62              # 接收完成后，清空消息
63              self.ui.in_message.clear()
64              # 发送消息，也要编码为 bytes
65              self.dataSocket.send(toSend.encode())
66              s = '我：'+toSend
67              self.ui.out_message.appendPlainText(s)
68          else:
69              print('发送失败！')
70              self.ui.out_message.appendPlainText('发送失败！')
71
```

三、实验结果与分析

首先在服务器窗口点击“开启聊天室”后，客户端可以点击“加入聊天室”按钮，加入成功之后会有提示信息，并且的客户端的页面下会显示当前有多少个客户加入了聊天。如下图所示为在自己主机运行服务器和一个客户机，在另一个主机上连接同一个热点 WIFI 使用 anacodan 软件开启的三个客户机，因此另一台主机的 IP 相同但是端口不同。

聊天的界面如下所示，服务器窗口显示所有人的聊天记录，并显示 IP 来标识，但是同一个主机开启的多客户端未能区别开来。

如下图所示是客户机的截图。



四、讨论、感想

本次实验在两个设备通讯的基础上做了进一步的改进，使得可以接收多客户机的连接并转发消息让所有客户机来接收从而实现群聊的功能，主要还是在两个设备通讯的代码拷贝之后的基础上要修改客户机的功能，让客户机也能持续接收消息，并且只在一个线程中接收，否则会出现歧义导致程序异常结束，此外需要在服务端实现简易的才处理消息和转发消息的选择等。此次实验增强了自己对于程序运行过程的思考能力，在实现目的的同时不能影响基础代码的运行。

大连理工大学实验报告

学院（系）：电子信息与电气工程学部 专业：计算机科学与技术 班级：电计 1905

姓　　名：陈豪宇　　学号：201961203　　组：陈豪宇

实验时间：2021.12.7 实验室：＿＿＿＿＿＿＿＿成绩：＿＿＿＿＿＿＿＿

实验三 本机/两台主机之间视频传输

一、实验要求

由客户端采集摄像头图像后经 Socket 传输到服务器端并在服务器端显示出来。

二、实验内容

同理省略开头的导入库函数以及初始化变量，直接解释 TCP 实现视频传输的功能，Video_TCP 为实例化 socket 对象并绑定对应的地址，每进行一步都输出提示信息，然后进入子线程处理服务器的请求功能，如果子线程接收到请求，则继续开启线程，原理与上述大体相同，不过多解释。

```
156     def Video_TCP(self):
157         self.ui.out_message.appendPlainText('Server: Ready for the Transmission of video')
158         self.TCPsocket = socket(AF_INET, SOCK_STREAM)
159         self.ui.out_message.appendPlainText('Server: Choose the TCP protocol')
160         # 将Socket（套接字）绑定到地址
161         self.TCPsocket.bind((IP, PORT))
162         self.ui.out_message.appendPlainText('Server: Binding the IP and PORT')
163         # 最多监听8个端口
164         self.TCPsocket.listen(8)
165         self.ui.out_message.appendPlainText('Server: Waiting for connecting videos using TCP...')
166         # 接受TCP连接并返回，其中conn是新的套接字对象，可以用来接收和发送数据，addr是链接客户端的地址。
167         th = Thread(target=self.Video_accept_TCP, args=())
168         th.setDaemon(True)
169         th.start()
170
171     def Video_accept_TCP(self):
172         conn, addr = self.TCPsocket.accept()
173         self.ui.out_message.appendPlainText('Successfully connected using TCP protocol ')
174         th = Thread(target=self.Video_star_TCP, args=(conn,))
175         th.setDaemon(True)
176         th.start()
177
```

在 Video_star_TCP 函数中，循环接收每一帧的数据，确认接收到的数据长度后给客户机回复一个'ACK'，代表已经收到消息，让客户机去自己计算延时和发送速率，之后进行对数据的处理，转字节流、转码等等，最后调用 opencv 来打开摄像头传输数据，在键盘上按下 ESC 按钮退出视频，如果直接点击视频右上角的关闭会导致对方未接收数据停止运行程序，从而使得界面卡顿住。

```
def Video_star_TCP(self,connect):
    while True:
        # 循环接收每一帧数据
        length = connect.recv(1024).decode()
        print('客户端发送的大小信息:',length)  # 首先接收来自客户端发送的大小信息
        if length == "Close":
            break
        if length != 0:
            # 确认已经接受数据长度
            connect.send('ACK'.encode())
            # 接收整张图片
            bytedata = connect.recv(BUFLEN)
            img = numpy.asarray(bytearray(bytedata), dtype="uint8")
            # 解码进行显示
            img = cv2.imdecode(img, cv2.IMREAD_COLOR)
            cv2.imshow("server_TCP", img)
            connect.send('ACK'.encode())
            if cv2.waitKey(1) == 27:
                self.Shut_Video()
                break

```

在客户机 Video_TCP 也是相似的代码实现，提示每一步的输出，绑定 socket 使用 TCP，从摄像头采集图像，接收每一帧的数据并开启子线程传递必要的参数。

```
def Video_TCP(self):
    # socket.AF_INET 用于服务器与服务器之间的网络通信
    # socket.SOCK_STREAM 代表基于TCP的流式socket通信
    self.ui.out_message.appendPlainText('Client: Ready for the Transmission of video')
    self.TCPsock = socket(AF_INET,SOCK_STREAM)
    self.ui.out_message.appendPlainText('Client: Choose the TCP protocol')
    # 连接服务端
    address_server = (IP, SERVER_PORT)
    self.ui.out_message.appendPlainText('Client: Binding the IP and PORT')
    self.ui.out_message.appendPlainText('Client: Waiting for connecting videos using TCP...')
    self.TCPsock.connect(address_server)
    self.ui.out_message.appendPlainText('Client: Successfully connecting videos using TCP!')
    # 从摄像头采集图像
    # 参数是0,表示打开笔记本的内置摄像头,参数是视频文件路径则打开视频
    capture = cv2.VideoCapture(0)
    # capture.read() 按帧读取视频
    # ret,frame 是capture.read()方法的返回值
    # 其中ret是布尔值，如果读取帧正确，返回True;如果文件读到末尾，返回False。
    # frame 就是每一帧图像，是个三维矩阵
    ret, frame = capture.read()
    encode_param = [cv2.IMWRITE_JPEG_QUALITY, 50]
    self.ui.out_message.appendPlainText('Goto show the video')
    th = Thread(target=self.TCP_start, args=(ret,frame,encode_param,capture))
    th.setDaemon(True)
    th.start()

```

在子线程中，如果读帧正确则继续进行，依次进行图片转码、压缩图像、转字节流、转矩阵、解码、显示图片，生成数据后先发送解码后的长度，如果服务器收到则继续回复‘ACK’，收到确认之后开启传输图像数据，并在此时开始计时，服务器收到图像数据后继续回复确认，此时计时结束，根据时间和图像的长度来计算发送速率和延时。

```
def TCP_start(self,ret,frame,encode_param,capture):
    while ret:
        # 首先对图片进行编码，因为socket不支持直接发送图片
        # '.jpg'表示把当前图片frame按照jpg格式编码
        # result, img_encode = cv2.imencode('.jpg', frame)
        # 压缩图像
        img_encode = cv2.imencode('.jpg', frame, encode_param)[1]
        # 转换为字节流
        bytedata = img_encode.tobytes()
        # 将输入数据转换为矩阵形式
        # bytearray返回一个新字节数组 读取了图片并显示出来，那么存储的数据格式必须为uint8
        data = numpy.asarray(bytearray(bytedata), dtype="uint8")
        # 解码进行显示
        # bool imencode(const String& ext, InputArray img, vector<uchar>& buf, const vector<int>& params=vector<int>())
        # ext-定义输出文件格式的扩展名 img-需要被编码的图像 buf-输出的缓存区，类型是vector parms-被编码的格式和压缩率,类型是vector 默认使用该种标识。
        # cv2.IMREAD_COLOR加载一张彩色图片，忽视它的透明度。
        img = cv2.imdecode(data, cv2.IMREAD_COLOR)
        # 载入的图片显示
        cv2.imshow("client_TCP", img)
        # cv2.waitKey(int delay)不断刷新图像，频率为delay，单位是ms，返回值为当前键盘按下的值，没有按键时返回-1.
        c = cv2.waitKey(1)
        stringData = data.tostring()
        # 生成标志数据
        # 首先发送图片编码后的长度
        self.TCPsock.send(str(len(stringData)).encode())

        ret, frame = capture.read()
        cv2.resize(frame, (640, 480))
```

此函数为断开视频连接的函数，作用防止一方在视频传输结束后存在一方的 socket 没有关闭因此可以手动关闭，一般来说由服务器一方按下 ESC 键关闭即可。至此使用 TCP 传输视频结束。

```
def Shut_Video(self):
    if self.TCPsock != None:
        self.ui.out_message.appendPlainText('TCP已断开连接')
        self.TCPsock.close()
    if self.UDPsock != None:
        self.ui.out_message.appendPlainText('UDP已断开连接')
        self.UDPsock.close()
    cv2.destroyAllWindows()
```

回到服务器，如果采用 UDP 传输视频则与 TCP 不同的地方在于 UDP 不需要连接，但是需要地址返回确认并让客户端计算延时和发送速率，在下面的函数会用到。此处 Video_UDP 为初始化设置，与上述原理相同。

```
209     def Video_UDP(self):
210         self.ui.out_message.appendPlainText('Server: Ready for the Transmission of video')
211         self.UDPsocket = socket(AF_INET,SOCK_DGRAM)
212         self.ui.out_message.appendPlainText('Server: Choose the UDP protocol')
213         # 将Socket（套接字）绑定到地址
214         self.UDPsocket.bind((IP, PORT))
215         self.ui.out_message.appendPlainText('Server: Binding the IP and PORT')
216         th = Thread(target=self.Video_star_UDP, args=())
217         th.setDaemon(True)
218         th.start()
219
```

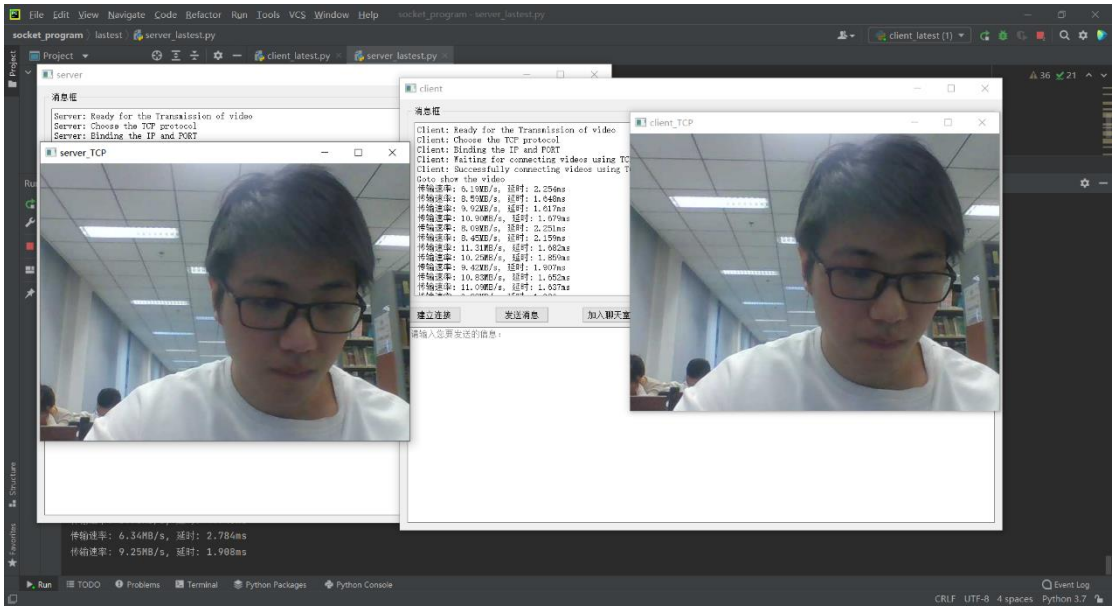
首先在进入死循环之后提示正在等待的信息，等到客户机返回图像的长度之后通过 recvfrom 得到客户机的地址，继续解码得到图像的数据，并返回确认信息，接收完成之后继续返回确认，之后开始解码显示图像，最后是退出的按钮。

```
227     def Video_star_UDP(self):
228         self.ui.out_message.appendPlainText('Waiting for the videos message using UDP...')
229         # 循环接收每一帧数据
230         while True:
231             length, addr = self.UDPsocket.recvfrom(1024)  # 首先接收来自客户端发送的大小信息
232             length = length.decode()
233
234             # 数据包小于60000，可以直接发送
235             if int(length) != 0:
236                 # 确认数据长度,回复ack
237                 self.UDPsocket.sendto('ACK'.encode(), addr)
238                 # 接收整张图片
239                 bytedata, self.addr = self.UDPsocket.recvfrom(BUFLEN)
240                 # 接收完一帧数据
241                 self.UDPsocket.sendto('ACK'.encode(), addr)
242                 data = numpy.asarray(bytearray(bytedata), dtype="uint8")
243                 # 解码进行显示
244                 img = cv2.imdecode(data, cv2.IMREAD_COLOR)
245                 cv2.imshow("server_UDP", img)
246                 if cv2.waitKey(1) == 27:
247                     self.Shut_Video()
248                     break
249
```

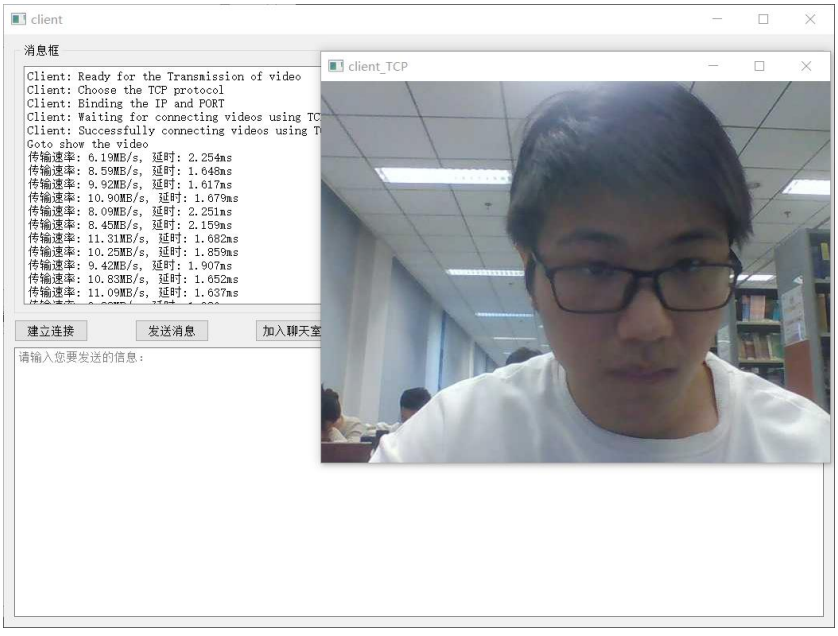
剩余客户机使用 UDP 传输视频的代码与使用 TCP 的类似，只是少了连接的步骤，send 函数换用 sendto 的函数，详细见代码附件，不再赘述。

三、实验结果与分析

在客户机代码中设置 IP 为 127.0.0.1，启动程序后在服务器端点击“接收视频/TCP”，然后在客户机窗口点击“打开视频/TCP”即可以实现使用回环地址的视频传输，具体界面如下所示，左侧是服务端观看的视频窗口，右侧是客户端打开视频的窗口，并在两个视频后的窗口中都由显示进行的步骤的详细提示。

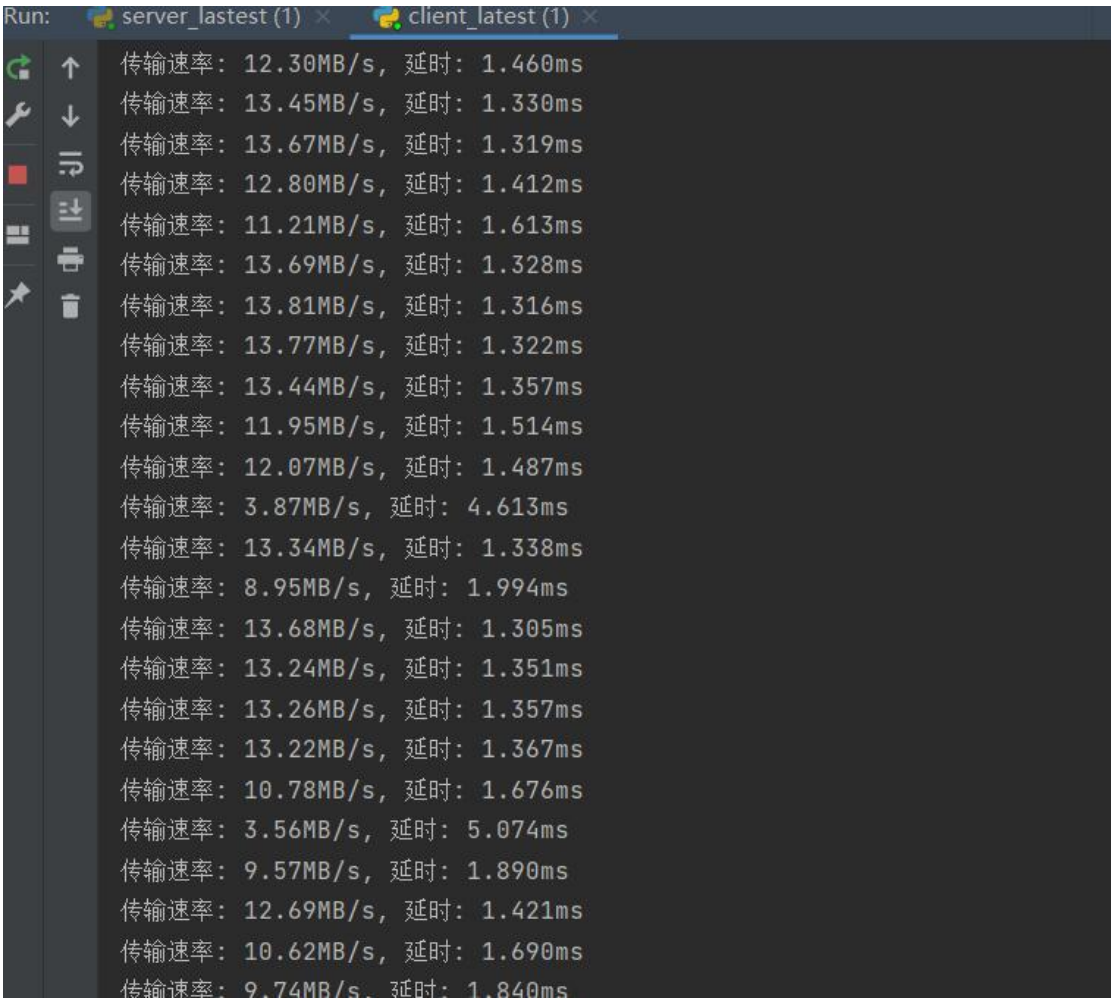


如下图所示是客户机的视频显示，消息框中弹出的即是在传输视频的过程中的传输速率和延时的显示，在代码中为了防止显示过快导致刷屏设置了每计算 20 次结果仅在窗口显示一次，但是在 pycharm 中完整显示。

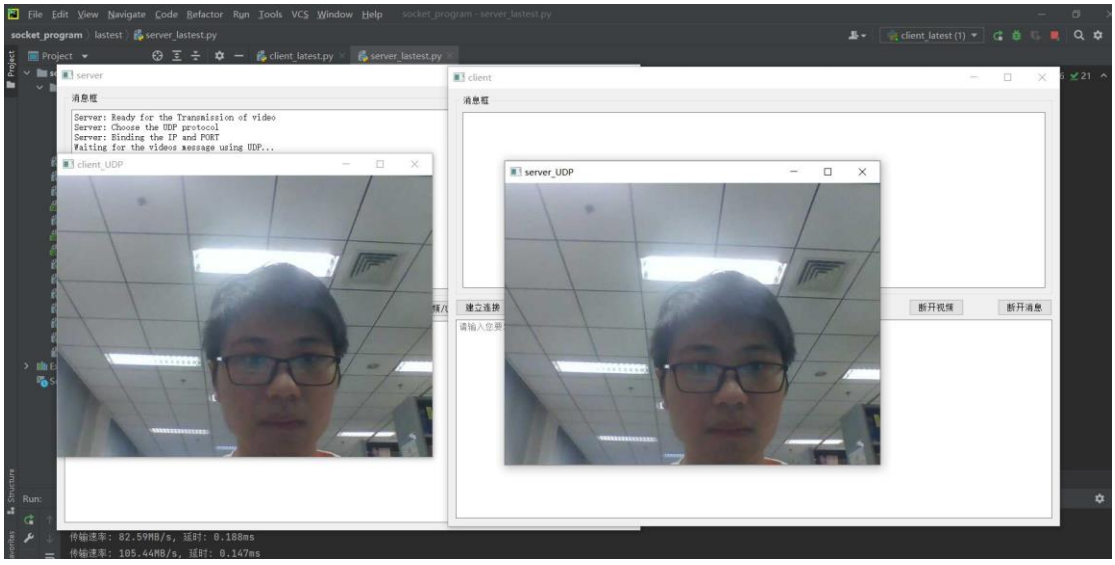


在进行完成视频传输后，服务器按下 ESC 按键退出该模式，并在服务器的消息框也提示 TCP 已经断开连接，客户机也会自动退出。

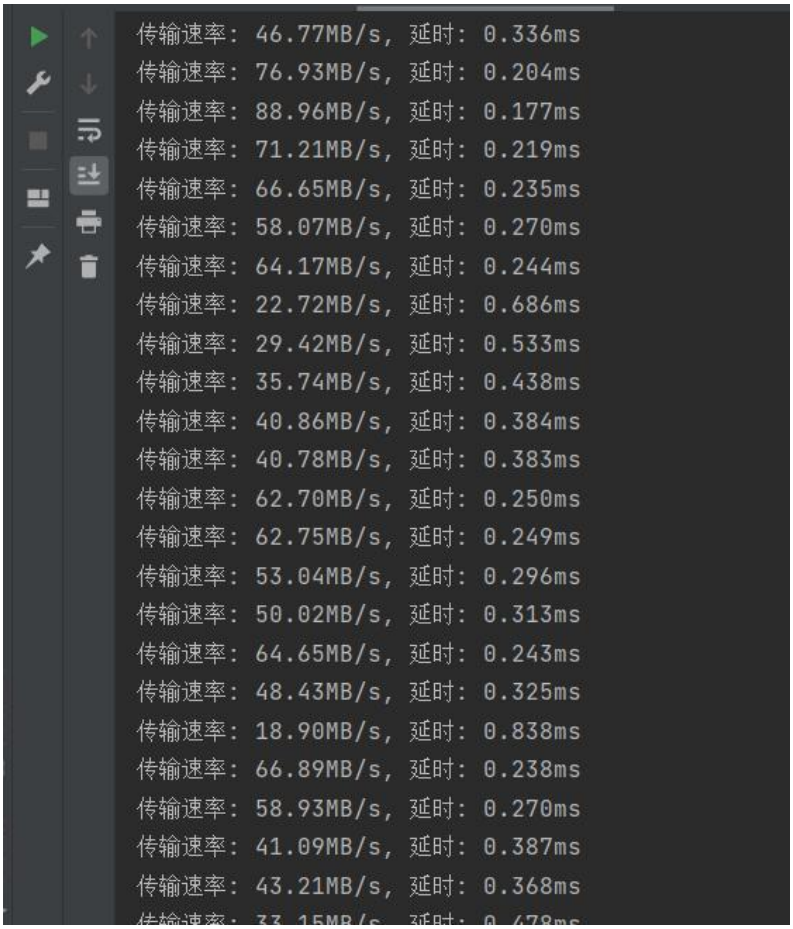
具体使用 TCP 的传输速率和延时的部分截图如下图所示：



使用 UDP 进行视频传输的界面与 TCP 相同，只是窗口名字做了区分。



使用 UDP 传输的速率和延时也如图所示，相比较来看 UDP 传输比 TCP 要很多。在传输视频结束后，如果程序中的 socket 完全关闭，则可以进一步点击别的按钮如换成 TCP 的视频传输、继续 UDP 视频传输、向服务器传输消息等操作，如果没有完全关闭可以通过手动点击按钮关闭，设计比较人性化。



四、讨论、感想

通过本次实验学习到了如何去使用 opencv 去实现 TCP 和 UDP 的实时视频传输，切实感受到了两者在传输速率上的差异，在出现 bug 调试时要多做一些 print 信息，即可以观察代码的运行情况，及时的找出出错的位置，又可以增加一些的运行过程中的体验感受。通过整个程序编写的过程中，还要注意 socket 编程的数据格式，比如在 send 的时候直接传输一系列的字符串或者数据会出现错误，只能先利用 python 拼接好一个完整的字符串再传输。整个实验完成之后，学习到了很多关于 python 编程和 socket 编程的知识，收获丰富，也体验了其中调试代码过程的无奈与心态崩溃，但是只能细心的检查，勤在浏览器上搜索问题总会得到问题的解答，计算机网络在生活中运用广泛，学习相关知识也可以帮助以后的就业选择或者就业方向，通过此次大作业初步入门 socket 编程也是体验相当不错，只是遗憾的是时间不够充裕，没有完善代码的其他功能，如传输文件、做完整的 socket 关闭和再启动、优化 ui 的图形界面等等。在实现聊天室功能时，自己与同学测试时使用手机热点测试成功，但是使用校园网时传输时服务器能看到大家发送的消息，但是客户机只能看到自己发送的消息，服务器未能完成消息的转发。

五、参考网站：

http://www.byhy.net/tut/py/etc/socket/

http://www.byhy.net/tut/py/gui/qt_08/

https://blog.csdn.net/DWei_2017_11_24/article/details/96314551?ops_request_misc=%257B%2522request%255Fid%2522%253A%252216388827516780274190116%2522%252C%2522scm%2522%253A%252220140713.130102334..%2522%257D&request_id=163888827516780274190116&biz_id=0&utm_medium=distribute.pc_search_result.none-task-blog-2~all~sobaiduend~default-2-96314551.first_rank_v2_pc_rank_v29&utm_term=python+socket%E4%BC%A0%E8%BE%93%E8%A7%86%E9%A2%91&spm=1018.2226.3001.4187