CS305 Online Conference Project Report

Xu Siting,11911635, Ma Chengqi, 11910337, and Wang Zeyu, 11910538

Computer Science and Engineering Department of Southern University of Science and Technology of China
e-mail: 11911635@mail.sustech.edu.cn
e-mail: 11910337@mail.sustech.edu.cn
e-mail: 11910538@mail.sustech.edu.cn

Date: December 27, 2021

ABSTRACT

Implement a conference platform that can support multiple online meetings, with functions of creating, joining, leaving, and closing a meeting, real-time video and voice transmission, screen share, desktop control, and obtaining meeting members in real time, etc. Made a user-friendly interface, the server and client have exception handling mechanisms.

Key words. online conference – client-server model – real-time transmission

Contents

1. Background & Survey

1.1. Background of the project

Affected by the COVID-19 pandemic, people have to change their traditional way of working from offline to online. Online meetings provide an effective way for people to communicate face-to-face with friends or clients without leaving home. Currently, there are two common network environments. One is circuit-switched network, such as ISDN, DDN, and PSTN. The other is network based on packet switching, such as ATM, IP, frame relay, etc. Currently, commonly used online meeting software includes video conference, text chat, audio conference, shared desktop, file transfer, and meeting permission management, and generally uses protocols such as RTP, RTCP, RTSP, SIP, and SDP.

1.2. Survey of related network protocol

Before we started our project, we made several surveys on existing protocols concerning online conference. RTP, SIP and RDP are the main protocols we researched, and the following is a brief introduction of the survey.

1.2.1. RTP

The *Real-Time Transport Protocol* (also called RTP) is a Transport layer Protocol used for multimedia data flows over the Internet. The RTP protocol specifies the standard packet format for transmitting audio and video over the Internet. It is commonly used in streaming media systems (with RTCP), video conferencing, and Push to Talk systems (with H.323 or SIP), making it the technical foundation of the IP telephony industry.

Sequential transmission is implemented in RTP. Serial numbers in RTP allow the receiver to rearrange the sender's packet sequence and can also be used to determine the appropriate packet location. For example, in video decoding, sequential decoding is not required. Our network conference referred to THE RTP protocol and made some improvements to the video transmission and decoding.

1.2.2. SIP

A SIP session uses up to four main components: a SIP user agent, a SIP registry server, a SIP proxy server, and a SIP redirection server. These systems complete the SIP session by transmitting messages that include the SDP protocol, which defines the content and characteristics of the message. The following provides an overview of each SIP component and its role in this process.

SIP User Agents (UAs) are end-user devices, such as mobile phones, multimedia handheld devices, PCS, PDAs, and so on, used to create and manage SIP sessions. The user agent client emits a message. The user agent server responds to the message.

The SIP registry server is the database that contains the location of all the user agents in the domain. In SIP communication, these servers retrieve the IP address and other relevant information of the party and send it to the SIP proxy server.

The SIP proxy server receives session requests from the SIP UA, queries the SIP registration server, and obtains the ADDRESS information of the UA. It then forwards the session invitation information directly to the receiver UA (if it is in the same domain) or to the proxy server (if the UA is in another domain).

The SIP redirection server allows the SIP proxy server to direct SIP session invitations to external domains. The SIP redirection server can reside on the same hardware as the SIP registration server and SIP proxy server.

Here is a typical SIP session: Fig.1

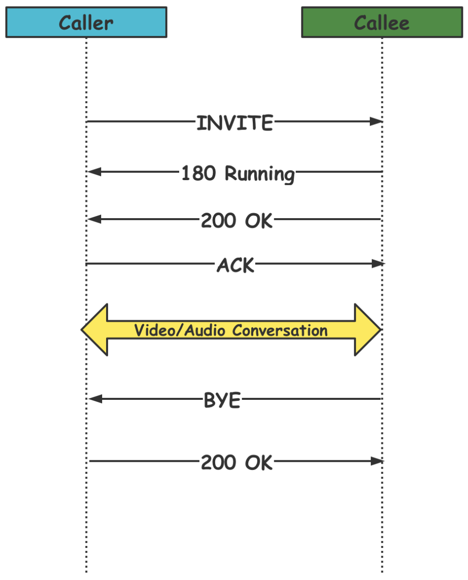


Fig. 1. a typical SIP session

1.2.3. RDP

Remote Desktop Protocol (RDP) is a mutil-channel Protocol that allows a user (client or "local computer") to connect to a computer (server or "Remote computer") that provides Microsoft terminal services.

Network connection layer: RDP is based on TCP/IP. As a large amount of data is transmitted, a network connection layer is defined at the bottom layer of the protocol. It defines a complete RDP data logic package to avoid data loss due to network packet length being broken up.

ISO data layer: On top of the network connection layer is the ISO data layer, which represents the normal connection communication of RDP data. Virtual channel layer: On top of ISO data layer, RDP defines a virtual channel layer to split data that identifies different virtual channels, speeding up client processing speed and saving time occupying network interfaces.

Encryption and decryption layer: RDP defines a data encryption and decryption layer on top of the virtual channel layer. This layer is used to encrypt and decrypt all functional data.

Functional data layer: above the encryption and decryption layer is functional data, screen information, local resource conversion, sound data, print data and other functional data information are processed in this layer. In addition, according to the different data types, these data have different levels of segmentation, their internal hierarchy will be explained in each functional module.

2. An overall of system

2.1. System Structure

Our system is mainly built based on clientserver model, which is composed of a server and a number of clients, which as follows: each end have four sockets including video socket, audio socket, message socket and desktop socket. The sturctures are as Figure 2,3

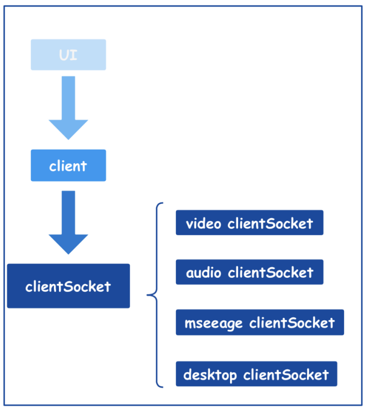


Fig. 2. The structure of client

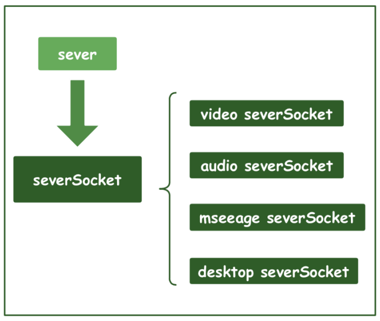


Fig. 3. The structure of sever

2.2. System Protocol

We use a custom socket protocol based on TCP/IP. The reason why do we need to customize the Socket protocol is that, In theory, the communication between the client and the server is asynchronous, and both can send and receive data at will. But in reality, they should work together.

for message socket: when the client sends or receives a specific signal, the server accepts and respond accordingly or sends a message ,which means whenever the client sends or receives data, we need to call the corresponding function.

The conference creator sends a create Conference message to the server. After receiving the message, the server creates a conference, randomly generates the conference ID, and sets the creator to the host of the conference. The client sends a Join message and the conference ID to the server. If the conference ID is correct, the server adds the client to the conference. When leaving a meeting, the client sends an exit message to the server, and the server moves the client out of the current meeting. When closing a meeting, the client sends a request to the server to close the meeting. If the client is the host of the current meeting, the server closes the current meeting. Otherwise, the operation is invalid.

for video/audio/desktop socket: The client continuously sends or receives data and processes the received data accordingly, and the server broadcasts the received data to other clients in a unified conferencewhich means we only need to turn on the receive and send functions when we enter the meeting.

After the client enters a meeting, it registers the connection with the server and continuously receives video and audio data from the server. When the client sends video and audio data to the server, the server forwards the data to all members in the meeting. When the client decides to turn off data such as video and audio, it no longer sends data to the server, but it can still receive data forwarded from the server.

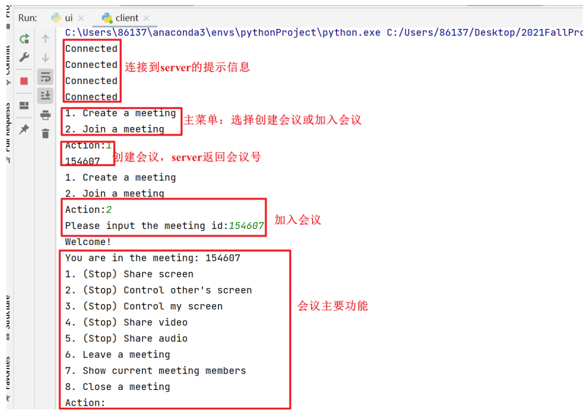
3. Testing & Verifying

3.1. System Testing Results

Figures below are the testing screenshots of our program. This section is divided into two parts - console part and ui part.

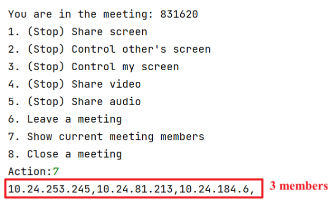
3.1.1. Console Part

Figures 4,5,6,7,8,9 are the figures of our console part.



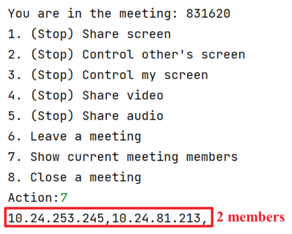
```
C:\Users\86137\anaconda3\envs\pythonProject\python.exe C:/Users/86137/Desktop/2021FallPrc
Connected
Connected
Connected
Connected
1. Create a meeting
2. Join a meeting
Action:1
154607
1. Create a meeting
2. Join a meeting
Action:2
Please input the meeting id:154607
Welcome!
You are in the meeting: 154607
1. (Stop) Share screen
2. (Stop) Control other's screen
3. (Stop) Control my screen
4. (Stop) Share video
5. (Stop) Share audio
6. Leave a meeting
7. Show current meeting members
8. Close a meeting
Action:
```

Fig. 4. The structure of console meeting



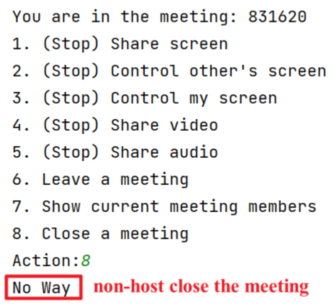
```
You are in the meeting: 831620
1. (Stop) Share screen
2. (Stop) Control other's screen
3. (Stop) Control my screen
4. (Stop) Share video
5. (Stop) Share audio
6. Leave a meeting
7. Show current meeting members
8. Close a meeting
Action:7
10.24.253.245,10.24.81.213,10.24.184.6,
```

Fig. 5. At first, three of us were all in the meeting



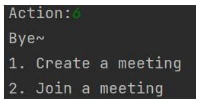
```
You are in the meeting: 831620
1. (Stop) Share screen
2. (Stop) Control other's screen
3. (Stop) Control my screen
4. (Stop) Share video
5. (Stop) Share audio
6. Leave a meeting
7. Show current meeting members
8. Close a meeting
Action:7
10.24.253.245,10.24.81.213,
```

Fig. 6. One of us left the meeting, there were only two members left



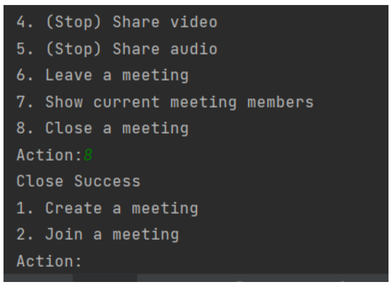
```
You are in the meeting: 831620
1. (Stop) Share screen
2. (Stop) Control other's screen
3. (Stop) Control my screen
4. (Stop) Share video
5. (Stop) Share audio
6. Leave a meeting
7. Show current meeting members
8. Close a meeting
Action:8
No Way
```

Fig. 7. When someone(not host) tried to close the meeting, a hint-"no way" will be printed on the console



```
Action:6
Bye~
1. Create a meeting
2. Join a meeting
```

Fig. 8. Someone leave the meeting



```
4. (Stop) Share video
5. (Stop) Share audio
6. Leave a meeting
7. Show current meeting members
8. Close a meeting
Action:8
Close Success
1. Create a meeting
2. Join a meeting
Action:
```

Fig. 9. Host administrator closed the meeting-close success

3.1.2. UI Part

Figures 10,11,12,13,14,15,16 are the figures of our UI part.

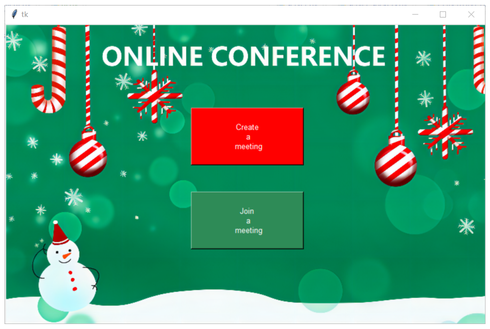


Fig. 10. This is the main window of our UI

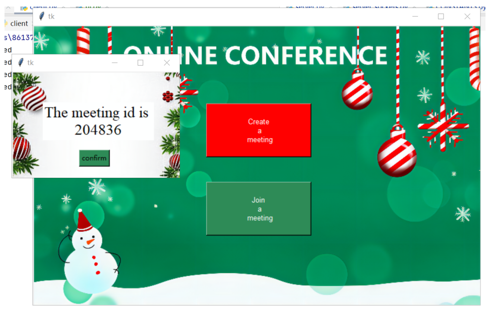


Fig. 11. Create a meeting

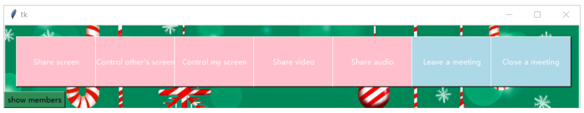


Fig. 12. This is the function of meeting window

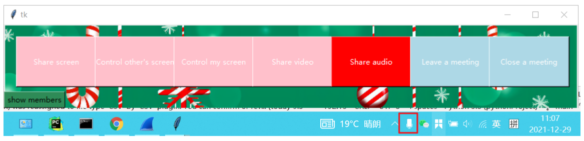


Fig. 13. Share audio - Tk is using my microphone

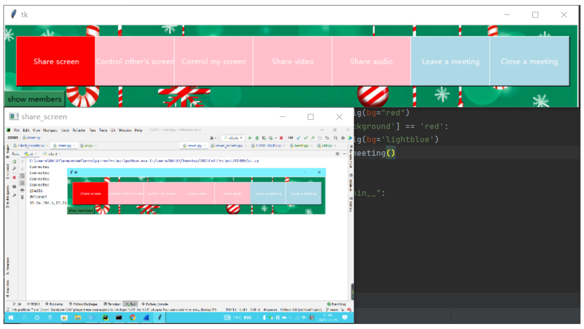
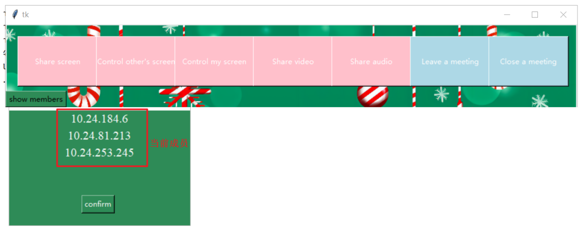


Fig. 14. Share Screen

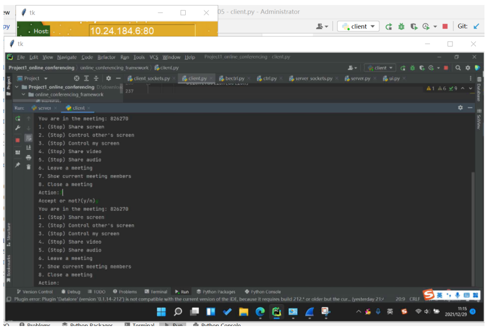


```
10.24.184.6
10.24.81.213
10.24.253.245
```

Fig. 15. Showing members in the meeting

3.2. Packet Capture Analysis

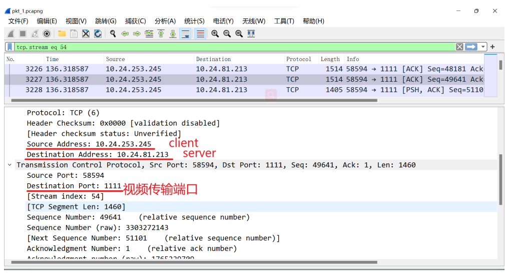
Figures 17,18,19,20,21,22 are the packet captured using wireshark. IP address of the server is 10.24.81.213. and IP address of the client is 10.24.253.245. We captured packets between client and server, which are of the port 1111-video, 1112-audio, 1113-message, 1114-desktop sharing and 80-desktop control. We also found the request and reply packet between client and server.


tk
Host: 10.24.184.6:80
client.py - Administrator
Project1_online_conferencing
online_conferencing_framework
client.py
server
client
11:15
2021/12/29

```
You are in the meeting: 826270
1. (Stop) Share screen
2. (Stop) Control other's screen
3. (Stop) Control my screen
4. (Stop) Share video
5. (Stop) Share audio
6. Leave a meeting
7. Show current meeting members
8. Close a meeting
Action:
Accept or not?(y/n)
You are in the meeting: 826270
1. (Stop) Share screen
2. (Stop) Control other's screen
3. (Stop) Control my screen
4. (Stop) Share video
5. (Stop) Share audio
6. Leave a meeting
7. Show current meeting members
8. Close a meeting
Action:
```

```
Plugin error: Plugin 'Datalore' (version '0.1.14-212') is not compatible with the current version of the IDE, because it requires build 212.* or older but the cur... (yesterday 21:
```

Fig. 16. Control others' screen

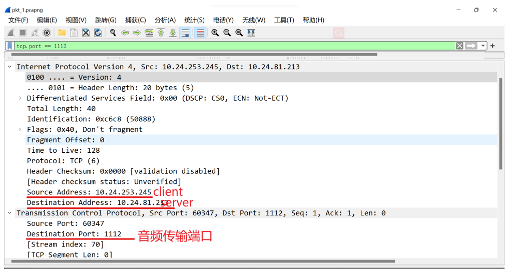


```
tcp.stream eq 54

No.     Time          Source          Destination    Protocol  Length  Info
   3226 136.318587    10.24.253.245   10.24.81.213   TCP       1514 58594 → 1111 [ACK] Seq=48181 Ack
   3227 136.318587    10.24.253.245   10.24.81.213   TCP       1514 58594 → 1111 [ACK] Seq=49641 Ack
   3228 136.318587    10.24.253.245   10.24.81.213   TCP       1405 58594 → 1111 [PSH, ACK] Seq=5110

    Protocol: TCP (6)
    Header Checksum: 0x0000 [validation disabled]
    [Header checksum status: Unverified]
    Source Address: 10.24.253.245
    Destination Address: 10.24.81.213
  Transmission Control Protocol, Src Port: 58594, Dst Port: 1111, Seq: 49641, Ack: 1, Len: 1460
    Source Port: 58594
    Destination Port: 1111
    [Stream index: 54]
    [TCP Segment Len: 1460]
    Sequence Number: 49641    (relative sequence number)
    Sequence Number (raw): 3303272143
    [Next Sequence Number: 51101    (relative sequence number)]
    Acknowledgment Number: 1    (relative ack number)
```

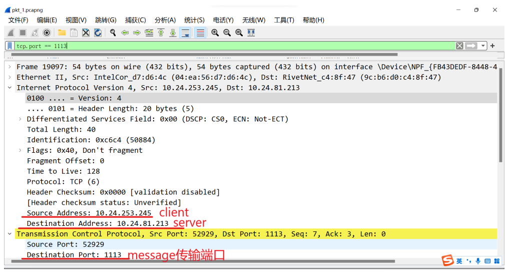
Fig. 17. packet of video, with port 1111



```
tcp.port == 1112

  Internet Protocol Version 4, Src: 10.24.253.245, Dst: 10.24.81.213
    0100 .... = Version: 4
    .... 0101 = Header Length: 20 bytes (5)
  > Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
    Total Length: 40
    Identification: 0xc6c8 (50888)
  > Flags: 0x40, Don't fragment
    Fragment Offset: 0
    Time to Live: 128
    Protocol: TCP (6)
    Header Checksum: 0x0000 [validation disabled]
    [Header checksum status: Unverified]
    Source Address: 10.24.253.245
    Destination Address: 10.24.81.213
  Transmission Control Protocol, Src Port: 60347, Dst Port: 1112, Seq: 1, Ack: 1, Len: 0
    Source Port: 60347
    Destination Port: 1112
    [Stream index: 70]
    [TCP Segment Len: 0]
```

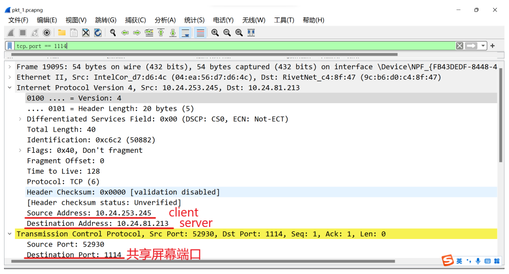
Fig. 18. packet of audio, with port 1112



```
tcp.port == 1113

> Frame 19097: 54 bytes on wire (432 bits), 54 bytes captured (432 bits) on interface \Device\NPF_{FB43DEDF-8448-4
> Ethernet II, Src: IntelCor_d7:d6:4c (04:ea:56:d7:d6:4c), Dst: RivetNet_c4:8f:47 (9c:b6:d0:c4:8f:47)
  Internet Protocol Version 4, Src: 10.24.253.245, Dst: 10.24.81.213
    0100 .... = Version: 4
    .... 0101 = Header Length: 20 bytes (5)
  > Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
    Total Length: 40
    Identification: 0xc6c4 (50884)
  > Flags: 0x40, Don't fragment
    Fragment Offset: 0
    Time to Live: 128
    Protocol: TCP (6)
    Header Checksum: 0x0000 [validation disabled]
    [Header checksum status: Unverified]
    Source Address: 10.24.253.245
    Destination Address: 10.24.81.213
  Transmission Control Protocol, Src Port: 52929, Dst Port: 1113, Seq: 7, Ack: 3, Len: 0
    Source Port: 52929
    Destination Port: 1113
```

Fig. 19. packet of message, with port 1113



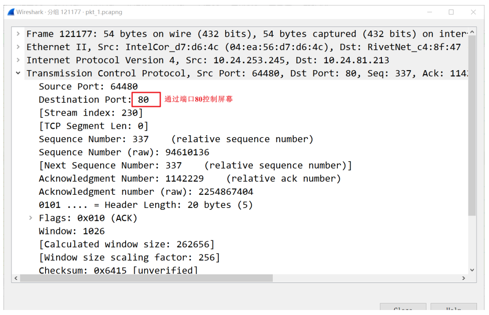
```
tcp.port == 1114

> Frame 19095: 54 bytes on wire (432 bits), 54 bytes captured (432 bits) on interface \Device\NPF_{FB43DEDF-8448-4
> Ethernet II, Src: IntelCor_d7:d6:4c (04:ea:56:d7:d6:4c), Dst: RivetNet_c4:8f:47 (9c:b6:d0:c4:8f:47)
  Internet Protocol Version 4, Src: 10.24.253.245, Dst: 10.24.81.213
    0100 .... = Version: 4
    .... 0101 = Header Length: 20 bytes (5)
  > Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
    Total Length: 40
    Identification: 0xc6c2 (50882)
  > Flags: 0x40, Don't fragment
    Fragment Offset: 0
    Time to Live: 128
    Protocol: TCP (6)
    Header Checksum: 0x0000 [validation disabled]
    [Header checksum status: Unverified]
    Source Address: 10.24.253.245
    Destination Address: 10.24.81.213
  Transmission Control Protocol, Src Port: 52930, Dst Port: 1114, Seq: 1, Ack: 1, Len: 0
    Source Port: 52930
    Destination Port: 1114
```

Fig. 20. packet of screen sharing, with port 1114

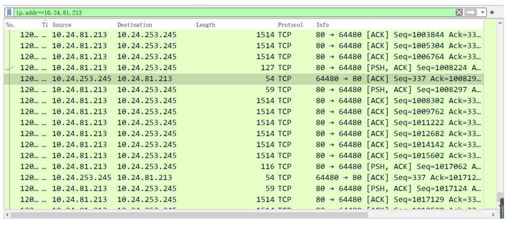
4. Summary & Future works

In this project, we built a network meeting based on TCP/IP based on the Client – Sever model. The main functions are: video



```
> Frame 121177: 54 bytes on wire (432 bits), 54 bytes captured (432 bits) on inter
> Ethernet II, Src: IntelCor_d7:d6:4c (04:ea:56:d7:d6:4c), Dst: RivetNet_c4:8f:47
> Internet Protocol Version 4, Src: 10.24.253.245, Dst: 10.24.81.213
v Transmission Control Protocol, Src Port: 64480, Dst Port: 80, Seq: 337, Ack: 114
    Source Port: 64480
    Destination Port: 80
    [Stream index: 230]
    [TCP Segment Len: 0]
    Sequence Number: 337    (relative sequence number)
    Sequence Number (raw): 94610136
    [Next Sequence Number: 337    (relative sequence number)]
    Acknowledgment Number: 1142229    (relative ack number)
    Acknowledgment number (raw): 2254867404
    0101 .... = Header Length: 20 bytes (5)
  > Flags: 0x010 (ACK)
    Window: 1026
    [Calculated window size: 262656]
    [Window size scaling factor: 256]
    Checksum: 0x6415 [unverified]
```

Fig. 21. packet of control screen, with port 80



```
ip.addr==10.24.81.213
No.     Ti Source          Destination     Length   Protocol  Info
120… …  10.24.81.213   10.24.253.245   1514 TCP  80 → 64480 [ACK] Seq=1003844 Ack=33…
120… …  10.24.81.213   10.24.253.245   1514 TCP  80 → 64480 [ACK] Seq=1005304 Ack=33…
120… …  10.24.81.213   10.24.253.245   1514 TCP  80 → 64480 [ACK] Seq=1006764 Ack=33…
120… …  10.24.81.213   10.24.253.245    127 TCP  80 → 64480 [PSH, ACK] Seq=1008224 A…
120… …  10.24.253.245  10.24.81.213      54 TCP  64480 → 80 [ACK] Seq=337 Ack=100829…
120… …  10.24.81.213   10.24.253.245     59 TCP  80 → 64480 [PSH, ACK] Seq=1008297 A…
120… …  10.24.81.213   10.24.253.245   1514 TCP  80 → 64480 [ACK] Seq=1008302 Ack=33…
120… …  10.24.81.213   10.24.253.245   1514 TCP  80 → 64480 [ACK] Seq=1009762 Ack=33…
120… …  10.24.81.213   10.24.253.245   1514 TCP  80 → 64480 [ACK] Seq=1011222 Ack=33…
120… …  10.24.81.213   10.24.253.245   1514 TCP  80 → 64480 [ACK] Seq=1012682 Ack=33…
120… …  10.24.81.213   10.24.253.245   1514 TCP  80 → 64480 [ACK] Seq=1014142 Ack=33…
120… …  10.24.81.213   10.24.253.245   1514 TCP  80 → 64480 [ACK] Seq=1015602 Ack=33…
120… …  10.24.81.213   10.24.253.245    116 TCP  80 → 64480 [PSH, ACK] Seq=1017062 A…
120… …  10.24.253.245  10.24.81.213      54 TCP  64480 → 80 [ACK] Seq=337 Ack=101712…
120… …  10.24.81.213   10.24.253.245     59 TCP  80 → 64480 [PSH, ACK] Seq=1017124 A…
120… …  10.24.81.213   10.24.253.245   1514 TCP  80 → 64480 [ACK] Seq=1017129 Ack=33…
```

Fig. 22. request and reply packet between client and server

conference, voice conference, desktop sharing, desktop control and conference control mechanism. In addition, we also made a beautiful UI interface. The whole project lasted three weeks and we were satisfied with the final result. In this process, we have a deeper understanding of the message sending and receiving between client and sever, as well as Multi-threaded programming in Python, and also learned how to deal with different kinds of data.

But the only fly in the ointment is that our error control mechanism is not perfect, for example in the control of desktop, control the client sends a request to the server and send the controlled the IP to the server, the server to being sent from the control request, control the control side is received and then to the server sends a reply before the back of the program. During this process, all sending and receiving is blocked and the process cannot continue if input errors are made on the control side. We can work on that in the future, and we can also try to implement more protocols.

And in the project itself, camparing to existing products on the Internet, it wasn't a user-friendly product. For us being unfamiliar with tkinter in python, the switching mechanisms were not well enough. There exists waiting for responding stage while running the control function for one send a request for desktop control and the becontroller has to reply to let the whole process continue, otherwise the meeting in the controller will stop and wait for becontroller's reply. To let it be more user-friendly, we need to work on tkinter UI design to let it be more humanizative so that different functions can work more smoothly when switching.

References

[1] https://www.cnblogs.com/qpanda/p/4331782.html
[2] https://github.com/TomPrograms/Python-Voice-Chat
[3] https://github.com/theintencity/p2p-sip
[4] https://www.lanqiao.cn/courses/672
[5] https://blog.csdn.net/xdwyyan/article/details
[6] https://blog.csdn.net/a997013919/article/details