Introdunction to Computer Networking Final Project

電機三 吳倉永
電機三 黃平瑋
電機三 張問寬

❑ Abstract

In the daily life, we usually watch video streaming on the Internet; however, most of the platforms on the Internet share a high level of similarity and do not perform as well as we expect. As a result, in this project, we strived to provide a livestream platform with some attractive, interesting functions and modes. In order to achieve this goal, we introduced some techniques we learnt on the class, such as HTTP, socket programming, TCP, IP, and so on. In addition, we introduce some special algorithm such as

It is worth mentioning that modules and packages designed by others take a small proportion in our project, only some standard libraries of Python are utilized. Instead, we handcrafted the main architecture and algorithms by the knowledge we learn.

❑ Introduction

○ Interface

■ Screen

The main part of this platform. The streamer and watchers can see the live video recorded by the webcam of the streamer.

■ Donate & Quality Setting

If users press the Donate Button, users will get something in return, and the button will show ”Donate Success!!”.

On the other hand, users can adjust the quality they want by pressing the other three buttons. Once one of them is pressed, the mode of transmission will be changed.

If **High Quality** is specified, the stream will be transmitted in HQ(Shown in Fig. 2).

If **Medium Quality** is specified, the stream will be transmitted in MQ(Shown in Fig. 3).

If **Low Quality** is specified, the stream will be transmitted in LQ(Shown in Fig. 4).

This part is one of our special designs. **Users can adjust quality independently, that is, when one user adjust quality, the streaming quality of others won’t be affected.** (In the beginning, the quality will be affected by others, we will state the way we fix this problem in the problem encounter section.)

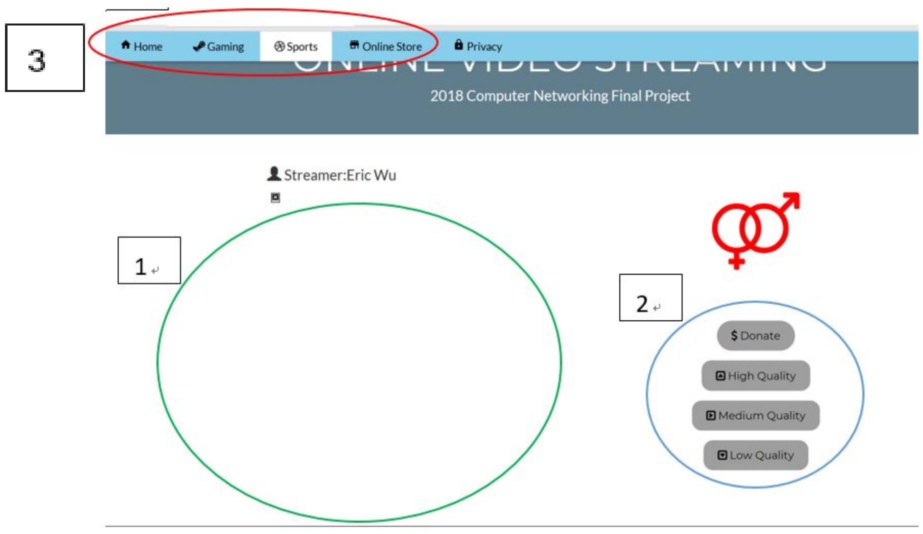


Fig.1 The interface of our livestream platform

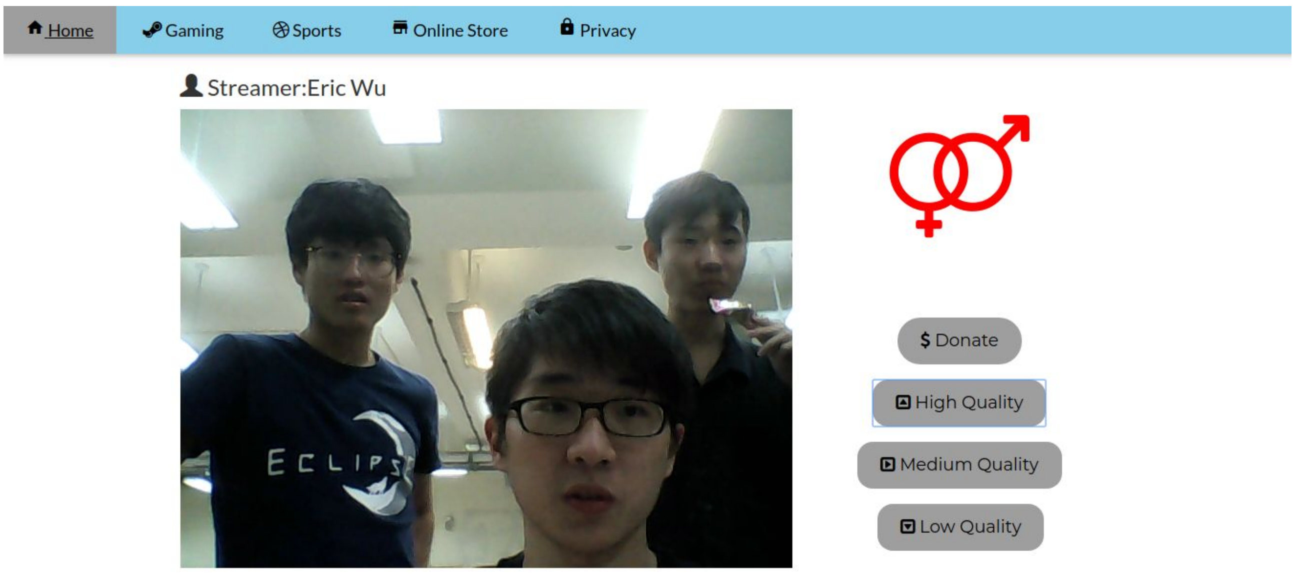


Fig. 2 High Quality

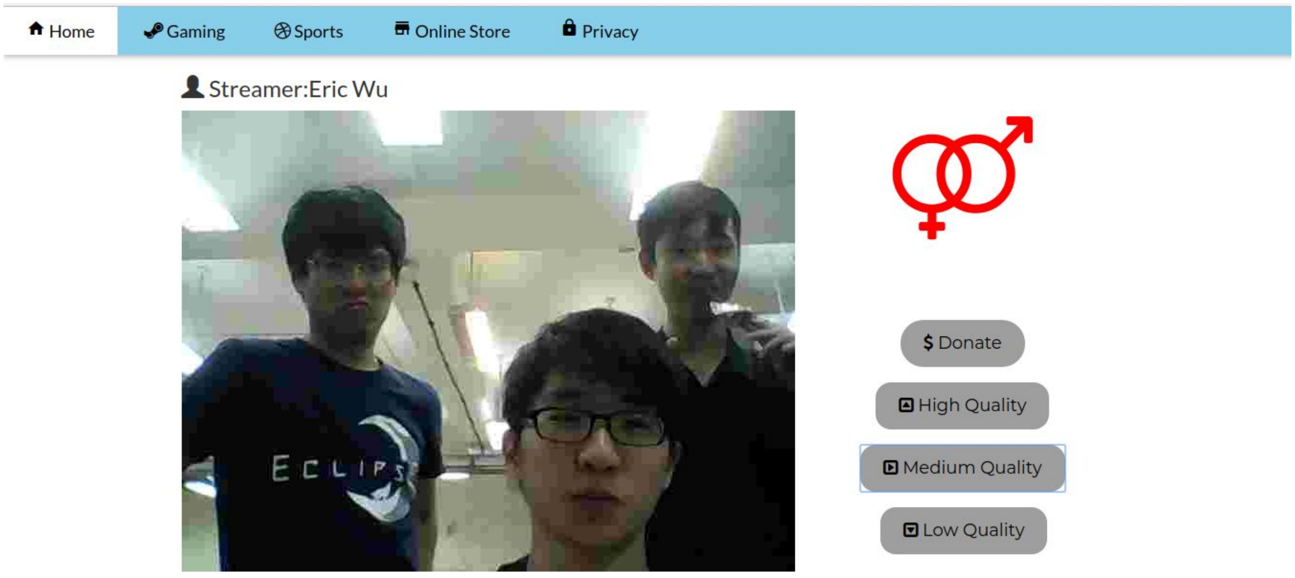


Fig. 3 Medium Quality

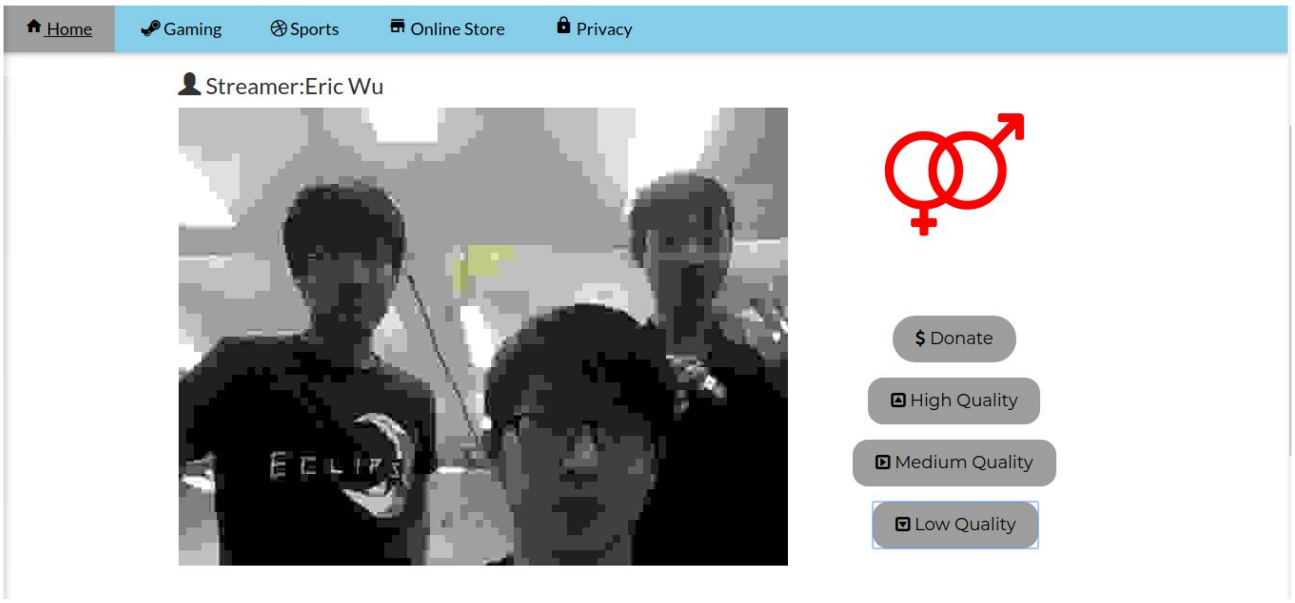


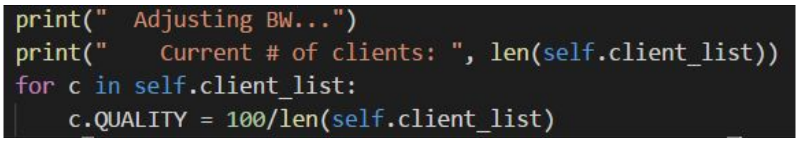
Fig. 4 Low Quality

- ■ Mode Setting

  Users can change modes between the Home button and the Gaming button. If Home Mode is specified, user can alter quality as mentioned above. On the other hand, if Gaming Mode is specified, which is our special design as well, user can't alter quality anymore. Instead, quality will change with the motion of the streamer, that is, **quality gets better as long as the streamer is moving, and gets worse as long as the streamer is stationary.** How we implement and solve this problem will be shown below in the backend implementation section.

- ○ Features
  - ■ Automatic Bandwidth Adjustment

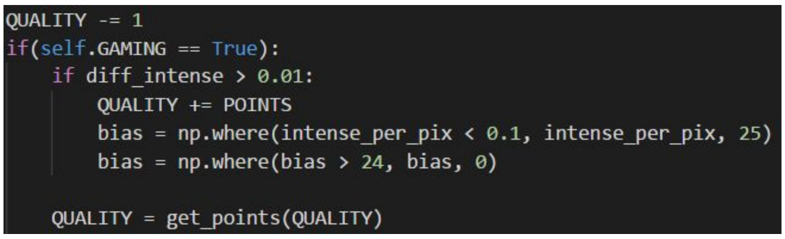
    Server will automatically adjust bandwidth for user by estimating the number of clients in use.



```
print("  Adjusting BW...")
print("    Current # of clients: ", len(self.client_list))
for c in self.client_list:
    c.QUALITY = 100/len(self.client_list)
```

  - ■ Motion Detection

    As mentioned above, if Gaming Mode is specified, quality will change by the level of the streamer's motion.



```
QUALITY -= 1
if(self.GAMING == True):
    if diff_intense > 0.01:
        QUALITY += POINTS
        bias = np.where(intense_per_pix < 0.1, intense_per_pix, 25)
        bias = np.where(bias > 24, bias, 0)

    QUALITY = get_points(QUALITY)
```

❑ Achievement

○ Implementation

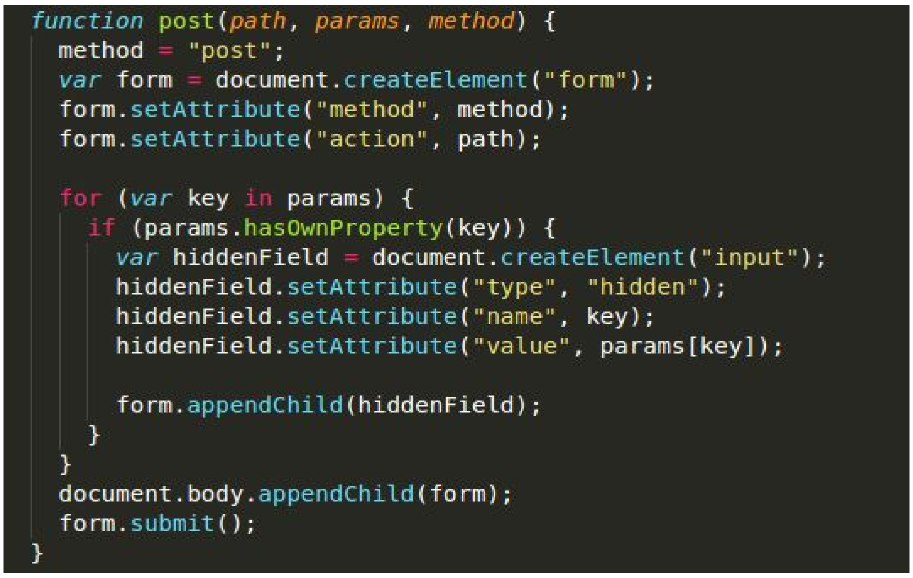
■ Frontend

Frontend are used to present server's data at client's browser, combining with HTML, CSS, and JavaScript.

1. **HTML**
   HTML(HyperText Markup Language) is a set of specifications to describe the elements of your webpage, create the basic structure and content of a webpage.

2. **CSS**
   Layout of the web page, where everything should be placed and define the style, such as font, color, padding, of Html element.
   Below is the CSS in our frontend.

3. **JavaScript**
   Interactive elements of a webpage that help to engage users.

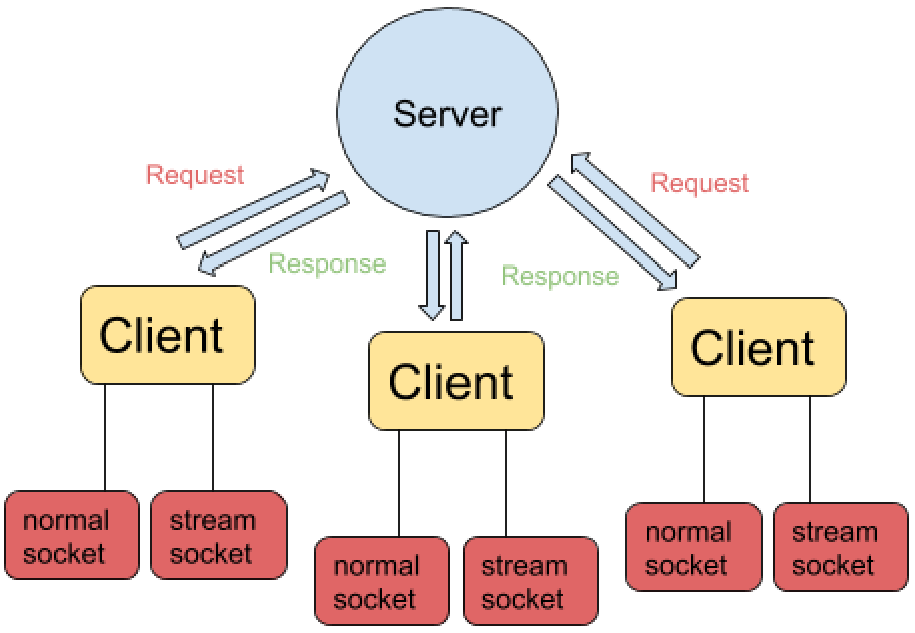


```javascript
function post(path, params, method) {
  method = "post";
  var form = document.createElement("form");
  form.setAttribute("method", method);
  form.setAttribute("action", path);

  for (var key in params) {
    if (params.hasOwnProperty(key)) {
      var hiddenField = document.createElement("input");
      hiddenField.setAttribute("type", "hidden");
      hiddenField.setAttribute("name", key);
      hiddenField.setAttribute("value", params[key]);

      form.appendChild(hiddenField);
    }
  }
  document.body.appendChild(form);
  form.submit();
}
```

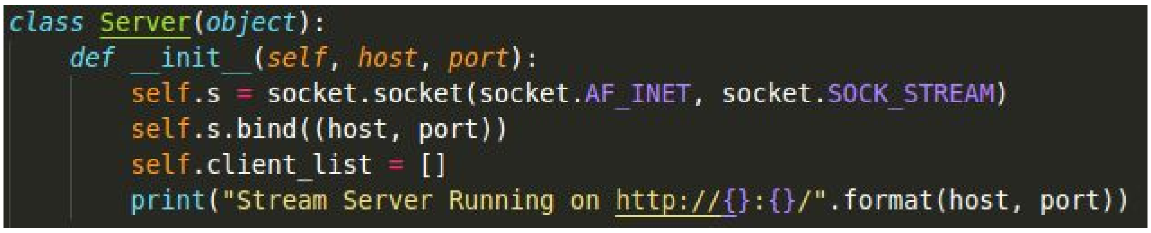
In this work, when we need to change the mode or quality, we need the help of JavaScript to interact with the backend. For example, if I want change to low quality, clicking the button will trigger the function "**post**", and this function will add parameter into the POST request sended to server. When server receive the request, it will adjust the quality as a response.

- **Backend**

The block diagram below shows how the server interacts with clients



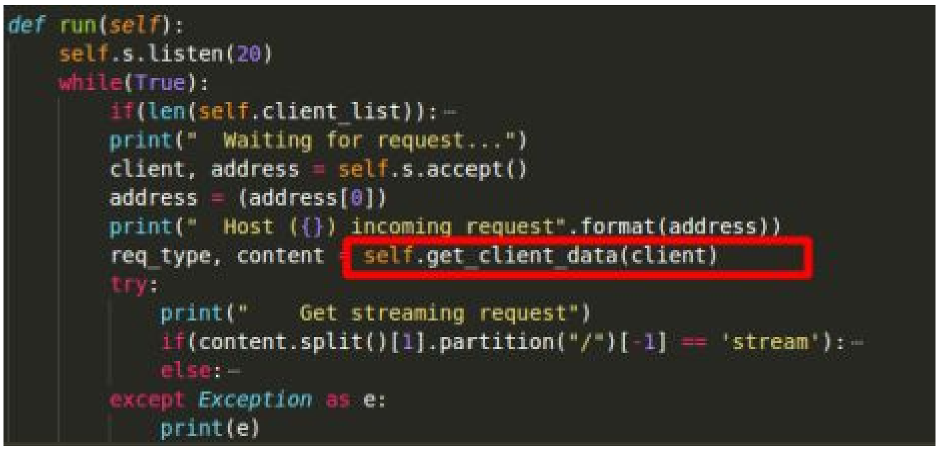
1. Server



```python
class Server(object):
    def __init__(self, host, port):
        self.s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        self.s.bind((host, port))
        self.client_list = []
        print("Stream Server Running on http://{}:{}/".format(host, port))
```

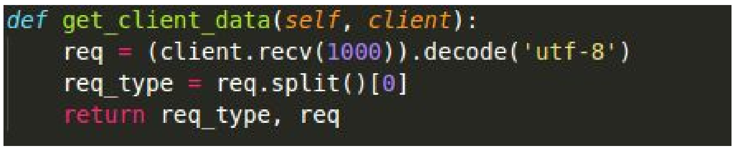
We run our server via TCP socket programming, and record unique client in a list Server can receive three kinds of request from the client, and deal with different action.

- New client sending GET request: adding new client to client_list
- Existing client sending GET request: use get_handler() in ClientHandler
- Existing client sending POST request: use post_handler() in ClientHandler



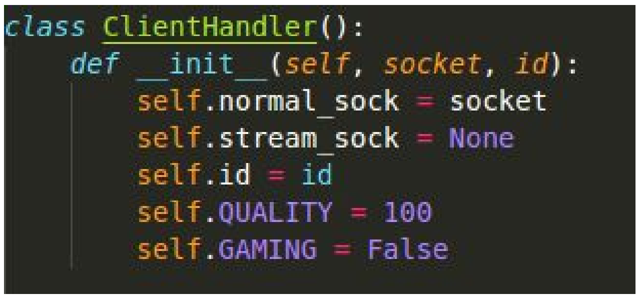
```
def run(self):
    self.s.listen(20)
    while(True):
        if(len(self.client_list)):-
        print("  Waiting for request...")
        client, address = self.s.accept()
        address = (address[0])
        print("  Host ({}) incoming request".format(address))
        req_type, content = self.get_client_data(client)
        try:
            print("    Get streaming request")
            if(content.split()[1].partition("/")[-1] == 'stream'):-
            else:-
        except Exception as e:
            print(e)
```

The main flow of the server waiting for requests from clients, using get client data.



```
def get_client_data(self, client):
    req = (client.recv(1000)).decode('utf-8')
    req_type = req.split()[0]
    return req_type, req
```

2. ClientHandler
   For each unique client, we use a ClientHandler to handle it. Thus, Server can manage the mode and quality of streaming between each client easily.

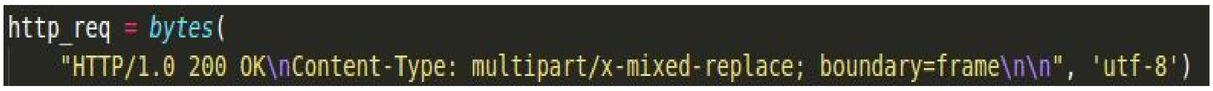


```
class ClientHandler():
    def __init__(self, socket, id):
        self.normal_sock = socket
        self.stream_sock = None
        self.id = id
        self.QUALITY = 100
        self.GAMING = False
```

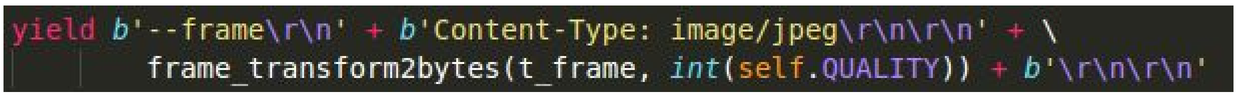
There are two main function in the ClientHandler, **get_handler()** and **post_handler()**.

Get_handler can handle two types of GET request, and generate corresponding header for response.

a. **Stream:**



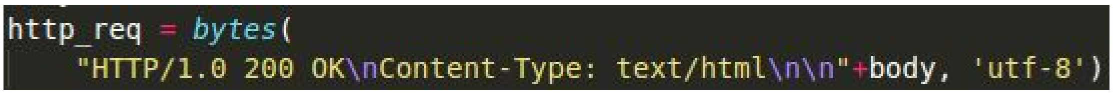
```
http_req = bytes(
    "HTTP/1.0 200 OK\nContent-Type: multipart/x-mixed-replace; boundary=frame\n\n", 'utf-8')
```



```
yield b'--frame\r\n' + b'Content-Type: image/jpeg\r\n\r\n' + \
        frame_transform2bytes(t_frame, int(self.QUALITY)) + b'\r\n\r\n'
```

To realize the function of streaming, we use a generating function continuously send frames to client via TCP socket.

b. HTML:



```
http_req = bytes(
    "HTTP/1.0 200 OK\nContent-Type: text/html\n\n"+body, 'utf-8')
```

This will send “index.html”, namely our frontend, to the client. “Index.html”, containing HTML, CSS and JavaScript, will render the streaming as background.

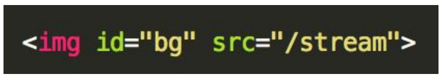
Post_handler can handle POST request from client, when client want to adjust their streaming quality or mode.

3. Multi-threading:
   In order to realize multiple sockets within a single client handler, we need to use multi-threading to handle it, otherwise different sockets will interfere with each other. There are 2 sockets in a ClientHandler, **normal socket** and **stream socket**, stream socket is used for frame transmission , while normal socket is used for receiving POST request, and it’s idle most of the time.

○ Difficulties we Encountered and Solution

Since the whole system was handcrafted by our own, in the developing stage, we have encountered two major problems which are all the backbone to our system.

1. **Streaming via HTML**:
   We utilized HTTPS protocol as our main transmission method, so we have to hardcode the header content. However, we cannot have the HTML rendered as our background with images continuously refreshing, since the browser should request the server for image source. Then, we have no idea how to implement it because we do not know how to keep request the server after the HTML request had been handled (no other triggering options). Finally, we change our image src as



```
<img id="bg" src="/stream">
```

   and it worked. After the HTML is rendered, it would request the image source “/stream” to our server.

2. **Multi-threading**:
   To fulfill multiple clients handling, we import threading module as our toolkit. But we always failed to accept for more than two clients, prompting with “Broken pipe” error. Since every time the server is requested, a new client socket would construct for it. We found that if we did not close the idle sockets, the server would not be able to receive any request at all, forced to break the pipe. Thus the client socket must be handle properly by a class and keep track of it, hence we have to update the new socket to the existing client and close the old one.

❑ Readme

- **Environment Setting**
  - First, ensure all the devices connect to the same LAN.
  - Then, get the IP address of server's (streamer's) sevice.(By **ifconfig** or other method)
- **Program Running**
  - After the environment setting, run the program by the command
    python app.py <port_num> .
  - Finally, open the website **IP_address:port_num** in client's devices
    (e.g. http://172.20.10.6:8081/)

❑ Conclusion

In this final project, we reflected on the **HW2**, using **socket programming** as our main server structure, which helped us integrating what we learnt in the class, and implement it to reality. To deal with multiple clients issue, with "**threading**" python module, we developed our own schematic to handle different user, forming a **server-client hierarchy**, creating **ClientHandler** class. Seeing the drawback of the multiple users, **self-adjusting bandwidth** is on the shelf to provide great user experience.

More importantly, we **handcrafted** a **LAN streaming server** with HTTPs protocol. To summarize our works, we utilize the knowledge taught by Professor, and do it by ourselves!

❑ Future Work

To make our streaming platform more interesting, we want to add more features to the platform, such as filtering on the stream, chatting room. It may require both frontend and backend be reviese. Moreover, we can mimic the youtube or facebook platform with stop button as our streaming platform to be more user friendly. To sum up, due to handcrafting, we still have a lot to be learnt!