CS305 · Computer Networking

何泽安 (He Zean)
12011323

PAssignment 2

Apr. 24, 2022

1 Design of System

1.1 HTTP

Environment This program was written under *Python 3.8*, exclude the buildin packages, it also uses asyncio 3.4.3 and flask 2.1.1. Since everyone should has asyncio, you may only check or install flask via

```
pip install Flask
```

I threw away the skeleton provided, and use *Flask* to provide the basic operations needed by designing a REST api. The server provides five endpoints:

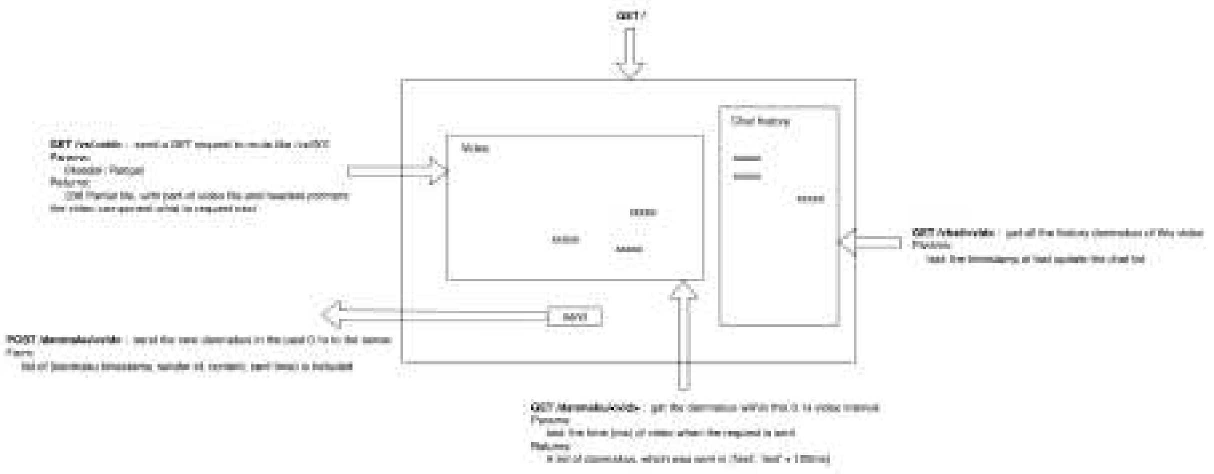
```python
@app.route('/', methods=['GET'])
def get_page():
    pass

@app.route('/vs/<vid>', methods=['GET'])
def get_video_part(vid):
    pass

@app.route('/danmaku/<vid>', methods=['GET'])
def get_danmakus(vid):
    pass

@app.route('/chat/<vid>', methods=['GET'])
def get_chat_history(vid):
    pass

@app.route('/danmaku/<vid>', methods=['POST'])
def post_danmaku(vid):
    pass
```

As mentioned in the document, the requests sent to route "/" (aka. http://localhost:8765/) will get a response, whose body is *danmu.html*, with two params filled by render_template (including the url for the background, and the client id assigned by the server).

The danmakus service uses "/danmaku" endpoint. Since we have multiple videos, we need to store the danmakus for different video separately, thus we put the video id into the url, other constraints are placed into parameters, as the REST style. All *POST* requests are handled by *post_danmaku* function, and the *GET* requests are handled by *get_danmakus* function.

Since I implemented the down streaming for the video, the *get_video_part* listening GET requests to "/vs". Each response contains part of the video file, and the video component in html5 will auto request the next part of file until EOF. Also, hope you enjoy the three fantastic Apple ads selected as the resource :)

Different from the danmakus show flowing above the video, which uses the relative time of a danmaku was sent in a video as the filter, the chat board displaying in the right side cares more about the absolute time of one danmaku was sent.



Note that for the bonus part, the live mode is integrated into the video mode, since the document mentioned "don't need to focus on the live contents", I use the videos as live, aka display a chat board that display the talking person nearly real-time (with some polling delay).

1.2 WebSocket

Rather than the HTTP version, WebSocket is quiet easy, and the provided skeleton is good enough to implement the danmakus. Since this protocol is *full duplex*, the server can not only receive messages from clients (of course), but also send message to client, or broadcast message to clients.

As the comments in the code said, the *websockets.serve* listens and handles connections to localhost:8765. When a new client is connected (after opening the html file in the browser, the js command *let ws = new*

WebSocket('ws://localhost:8765') will create a new client socket and connect to the server socket), a new async function reply will be called and keep running in background, until a ConnectionClosed is raised to stop this thread.

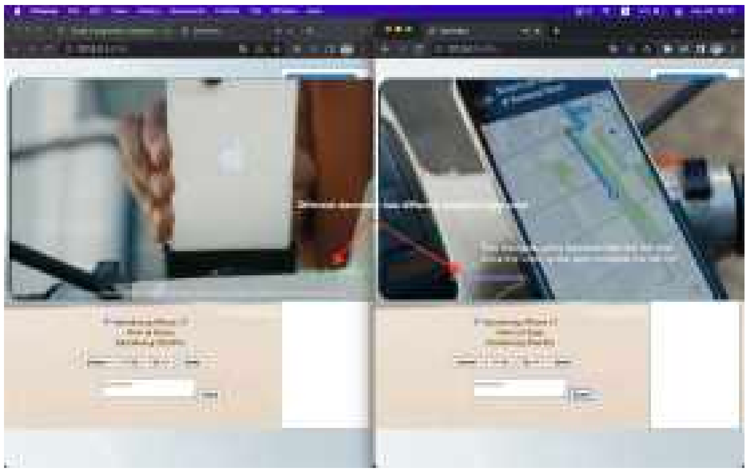
Assume that there are n websites opening (normally they should keep connecting with the server), they are considered as n client sockets and all added into the clients list of a DanmakuServer instance. In all the time, there are $n + 1$ threads, one holds the main, and other n threads each contains a reply. In the most of time, the function is blocked by the async for, once a client sends message to the server, the corresponding thread's function will step into a for loop, in which the server broadcast the message to the list of clients.

2 Running Results

2.1 HTTP

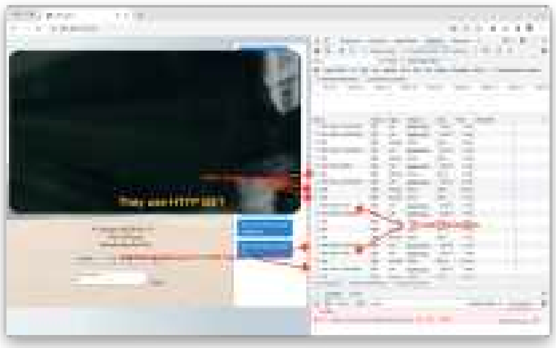
Since the logic seems correct for me, the HTTP version can support infinite users theoretically, and the display of danmakus has no missing or repeat, and low latency. However, since we have to use polling for HTTP, which will cause a lot of CPU and memory cost, I've tested opening 8 webpages at one time and the fan began running fast.

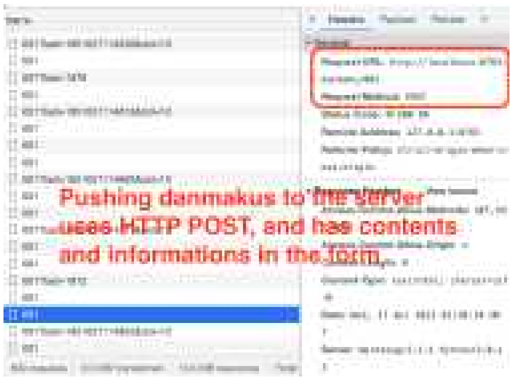
The *video mode* shows danmakus not in real time, but bases on the time of video, as shown below. But you may also notice that the chat board floating in the right side of webpage shows danmakus in the realtime, base on the time it was sent, but not about the time in the video, that's for the *live mode.*



* The page has been beautified, such as adding a background picture, and changing the color of danmakus.
** Please use Chrome if the video component looks strange or cannot click play/pause in other browsers.

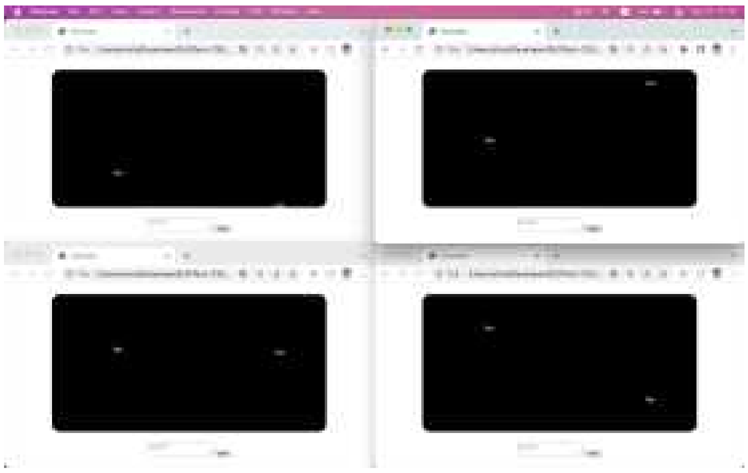
Packages can be caught by Wireshark (Loopback: lo0, set the display filter as tcp.port == 8765) or F12 Network watcher.



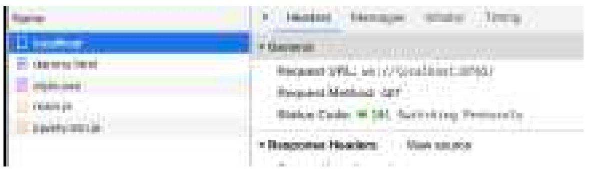

Pushing danmakus to the server uses HTTP POST, and has contents and informations in the form

3 WebSocket

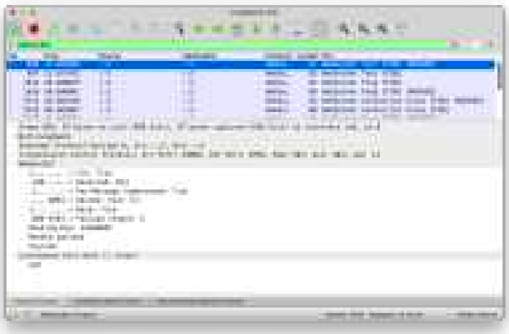
The ws version is just the baseline, without any additional method. As below shows, opening many clients in the same time has the correct effect as we expected. Moreover, since ws is full duplex, we need no stupid and CPU/memory/IO costing pollings. We can easily open 10+ webpages without adding much workload.



Chrome's developer tool doesn't support capture packages under WebSocket protocol, as below shows. But we can capture the packages by using Wireshark, setting the display filter as *websocket*.



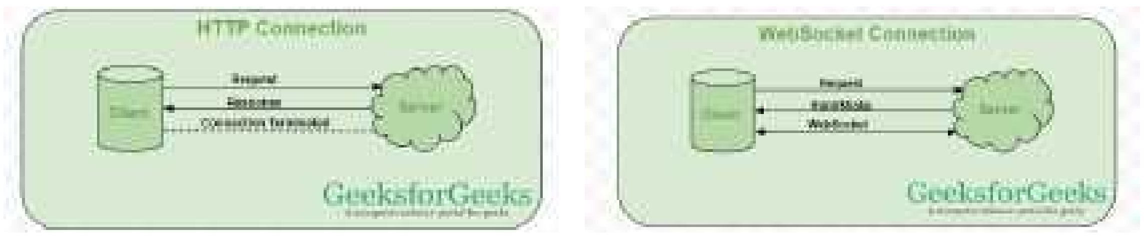
Request URL: ws://localhost:[illegible]/
Request Method: GET
Status Code: 101 Switching Protocols



4 Analysis

4.1 Design Diagram

The biggest different between HTTP and WebSocket is that, HTTP is unidirectional while WebSocket is bidirectional and full-duplex. When a client sends an HTTP request to the server, a TCP connection is open between the client and server, after getting the response, the TCP connection will be terminated, each HTTP request opens a separate TCP connection to the server. Here are two cons for the danmaku scenario: ① The server cannot send message to a client initiatively, since the server doesn't remember the IP address and the port number of the clients that need (subscribe?) the service. Under this situation, the client has to send requests to get the response initiatively and regularly. ② The polling mechanism can receive new danmakus, but not good enough. The latency may get even larger when the server needs to serve many clients that set the polling time interval larger. Also, the cost of performance of the server grows fast as the users are getting more and more. Now back to WebSocket, it easily solve the above two problems, yet the cost of keeping many connections between the server and each client may be appreciable.



src: www.geeksforgeeks.org/what-is-web-socket-and-how-it-is-different-from-the-http

4.2 Network Traffic

As we mentioned in the last section, the client using HTTP protocol has to send many request packages over the time, unfortunately, in the most of time there's no new message for the client, which means the most part of request packages sent by client (polling) contribute nothing for the service, but the resources are wasted.

Besides, since HTTP is stateless, the extra information must be carried in the header fields in a package, while the server under the WebSocket protocol remember the state of clients, and the packages are "lighter", in a word, the ratio of valid data, defined as data carrying info / (data + headers), of WebSocket is greater than the one of HTTP.

4.3 Upgrade Efficiency

Trivially, the latency of WebSocket is much less than the latency of HTTP. Since WebSocket can "broadcast" messages, or initiative send messages to client, the latency is the time to do operations on data and the time for packages to be transmit to the client, while the HTTP version's latency is "large or small" latency of polling (depends on the polling's frequency and the time of message is generated in a polling interval) and the time for doing operations and an even larger transmit time, since a new TCP connection needs to be established (handshaking) and the number of bits of headers are larger then the ws's headers.

4.4 Comparison Summary

We've notice that HTTP is stateless, in the scenarios that requires in-time data, that perform is awful, yet the server need not to keep the connections, which means it's more expandable (for the large amount of scenarios like browsing the net, watching videos, HTTP is fair enough and more fit to the public server that need to serve a large amount of users). If some serves need the message get just in time, WebSocket is much better than HTTP, such as the machines in labs, theating as servers, that may want to send message initiative to the endpoints acting as client, and the number of client won't be so large.