CS305 CDN Project Report

Developers and Work Division

SID	Developer	Work Division
12012530	张力宇	Compile and Output, Proxy, Danmakus System
12011436	李佳纯	Log File, Proxy, Bit Rate Adaption, Graph Analysis
12012919	廖铭骞	DNS, Round-Robin, Interface Beautification, Report

Proxy Implementation

Arguments

The function of arguments of the proxy.py is like belows.

```
        argv[0] is the python file name.
        argv[1] is <log>, the path of file that logs the messages.
        argv[2] is <alpha>, [0, 1], in throughput estimate.
        argv[3] is <listen-port>, the TCP port your proxy should
listen on for accepting                    connections from your
browser.
        argv[4] is <dns-port>, UDP port DNS server listens on.
        argv[5] is [<default-port>], proxy should accept an optional
argument specifying the                port of the web server
               from which it should request video chunks. If this
argument is not present,
               proxy should obtain the web server's port by querying
DNS server.
```

Forward Request

The forwarding instances code is shown as belows. Take the StrobeMediaPlayback.swf request forwarding as an example.

```
 @staticmethod
    @app.route('/StrobeMediaPlayback.swf')
    def query_strobe():
        cookie = request.cookies
        cur_client_id = int(cookie.get("ID"))
        current_client = ClientSelf.dict_client.get(cur_client_id)
        return Response(requests.get(current_client.getURL() +
"/StrobeMediaPlayback.swf",      headers=ClientSelf.headerSelf))
```

Strategy to Exit Proxy

Our strategy to exit the proxy is by entering "quit" into danmakus text field. If the danmakus system receives the "quit" message, it will trigger a process to exit the proxy. The code is shown as belows.

```
danmu = request.stream.read().decode('utf-8')
if danmu == "quit":
    shutdown_func = request.environ.get('werkzeug.server.shutdown')
    if shutdown_func is None:
        raise RuntimeError('Not running werkzeug')
        shutdown_func()
        return "Shutting down..."
```

Video Bit rate Adaptation

The goal of bit rate adaptation is to choose the best rate for each video chunk. The choice is based on throughput estimation.

The throughput can be calculated as belows. Assume the B is the length of the video chunk and t_s and t_f is the start time and end time of the process of receiving the chunk from the servers.

$$T = \frac{B}{t_f - t_s}$$

The current of the throughput of estimate is shown as belows.

$$T_{current} = \alpha T_{new} + (1 - \alpha) T_{current}$$

The code is shown as belows, and the method is explained in the comment.

```
@staticmethod
    @app.route('/vod/<int:rate>Seg<int:segment>-Frag<int:frag>')
    def forward_segment(rate, segment, frag):
        """
            Here you should change the requested bit rate according to
your computation of throughput.
            And if the request is for big_buck_bunny.f4m, you should
instead request big_buck_bunny_nolist.f4m
            for client and leave big_buck_bunny.f4m for the use in
proxy.
        """

        cookie = request.cookies
        cur_client_id = int(cookie.get("ID"))
        current_client = ClientSelf.dict_client.get(cur_client_id)

        # to select a proper bitrate
        bound_bitrate = current_client.throughput / 1.5
        print(bound_bitrate)
        selected_bitrate = 10

        for bps in current_client.bitrate_list:
            if bps <= bound_bitrate:
                selected_bitrate = bps

        print(selected_bitrate)

        start_time = time.time()
        response =
requests.get(f'{current_client.getURL()}/vod/{selected_bitrate}Seg{seg
ment}-Frag{frag}', headers=ClientSelf.headerSelf)
```

```
        end_time = time.time()
        KB = float(response.headers['Content-Length']) * 8 / 1024
        # record the log message
        duration = end_time - start_time
        new_throughput = KB / duration

        current_client.throughput = ClientSelf.alpha * new_throughput
+ (1-ClientSelf.alpha) * current_client.throughput
        chunkname = str(segment) + "-" + str(frag)
        ClientSelf.record_log(start_time, duration, new_throughput,
current_client.throughput, selected_bitrate,
current_client.server_port, chunkname)
        return Response(response)
        # record the log message
```

DNS Implementation

Read File

Since the DNS should know which port is open before answering the request, the DNS.py will first read servers file from command line as parameters.

```
file_name = sys.argv[1]
    with open(file_name, 'r') as f:
        for line in f.readlines():
            port_list.append(line.strip())
```

Then, the $port\_list$ will contains all the open ports.

At first, I am not sure whether the port in the file will open for certain, so we also write a method to examine it. The method is shown as belows.

```
# 判断端口是否被占用
def is_open(port, host = '127.0.0.1') -> bool:
    s = None
    try:
        s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        s.settimeout(1)
        s.connect((host, int(port)))
        return True
    except socket.error:
        return False
    finally:
        if s:
            s.close()
```

But after the test, we found that when using `python3 ./netsim.py servers start -s servers/2servers` to launch servers, the port in the port file will be open, so it is no need using the method to check whether the port is open. So we delete the method at last.

Round-Robin

The principle of Round-Robin algorithm is an arrangement of choosing all elements in a port group equally in some rational order, usually from the front of a list and then starting again from the front of the list and so on. A simple way to think of round robin is that it is about "taking turns." Used as an adjective, round robin becomes "round-robin."

In our implementation, Round-Robin is like the iteration, each time will return the next element of the port group. When the pointer which indicates the previous port reach the tail of the port group, then it will begin from the front of the group again.

The implementation code is shown as belows.

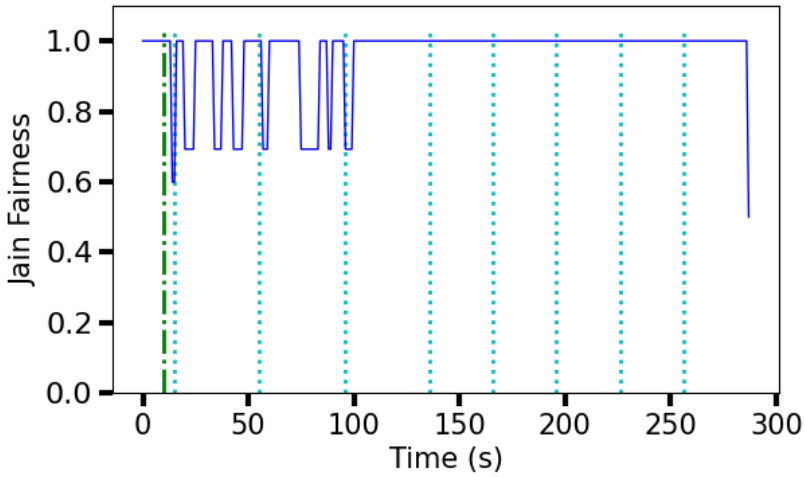
```
@staticmethod
    @app.route('/dnsRequest')
    def makeDnsRerurn():
        global pre_port
        # choose file according to the input filename
        pre_port = (pre_port + 1) % len(port_list)
        return make_response(str(port_list[pre_port]))
```

Graph and Analysis

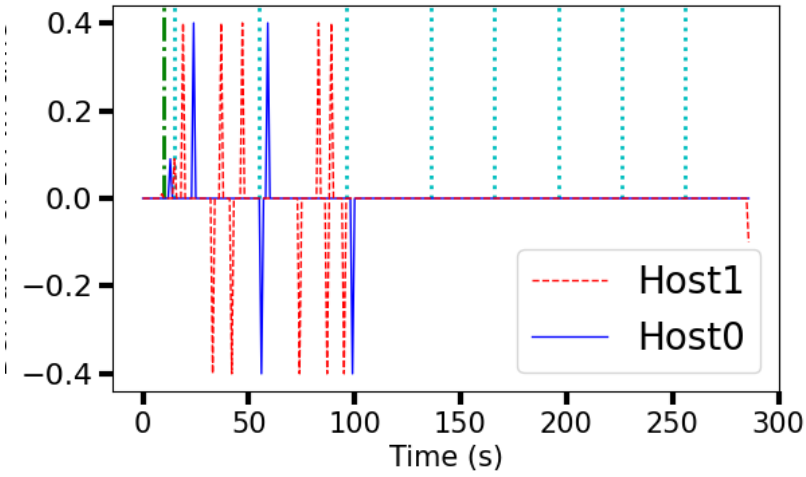
The graphs we drew and the related analysis are shown as belows.

onelink, two servers, alpha = 0.6:

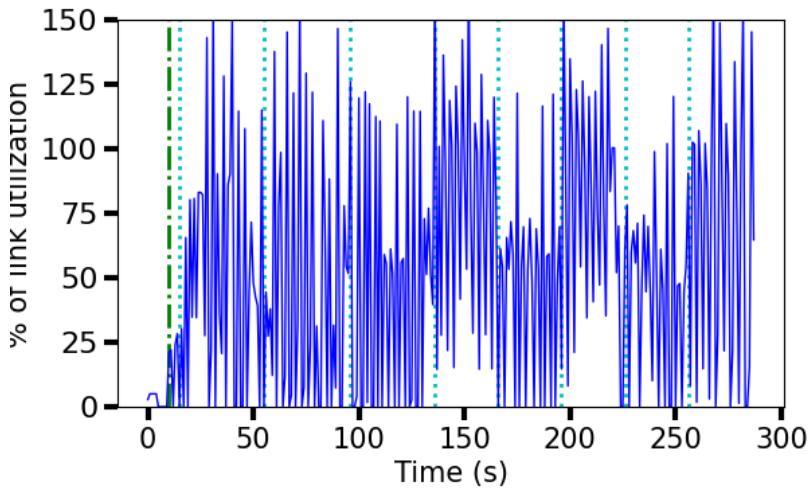
Fairness:



Smoothness:



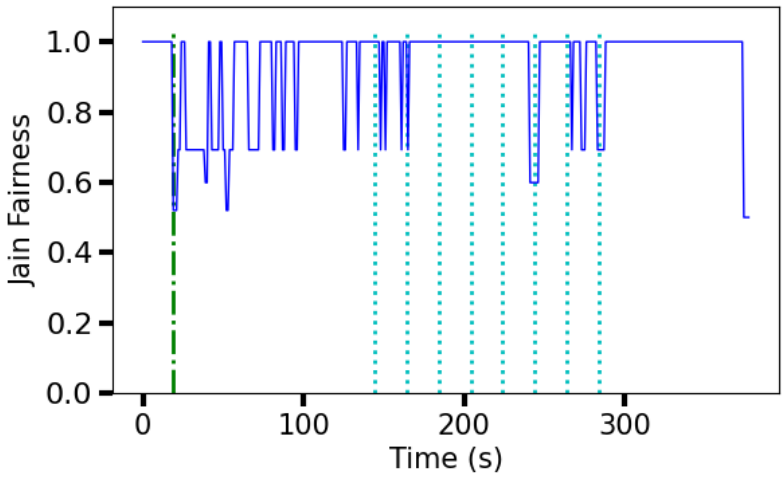
Utilization:



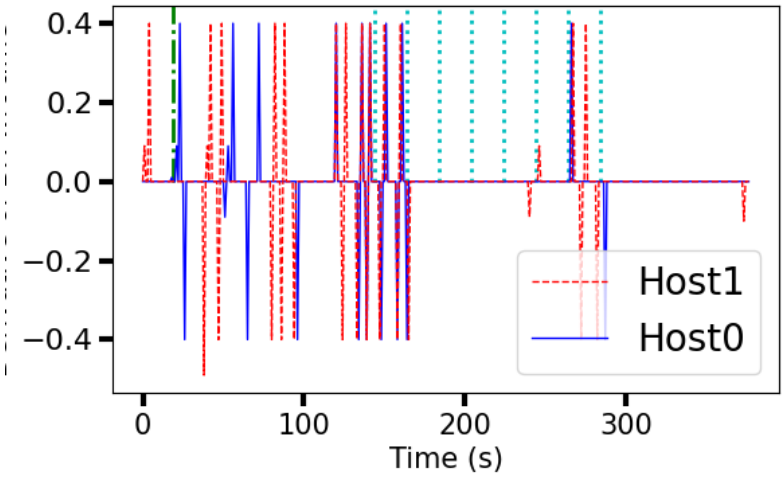
Analysis: During the first 3 events, the fairness presents a state of fluctuation. We can see that the fairness is jumping from 1.0 to 0.7 and from 0.7 to 1.0, which means the network is always trying to keep the fairness. And then the network keeps fairness at 1.0. It is strictly fair. In the smoothness graph, we can also see that during the first 3 events, the graph presents a state of fluctuation. This is the same state as fairness, which is normal. The utilization graph is strange when we first look at it. After analyzing, I think it is reasonable. Any utilization point larger than 100% is the result of decreasing the link bit rate suddenly. As our utilization is depending on the duration and the bit rate, when our duration is not small, the duration*bit rate will not decreases so quickly that makes the point larger than 100%.

sharelink, two servers, alpha = 0.6:

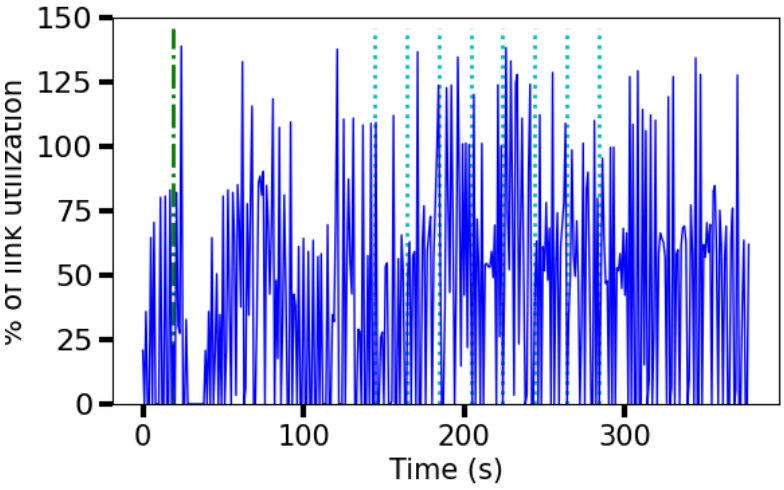
Fairness:



Smoothness:



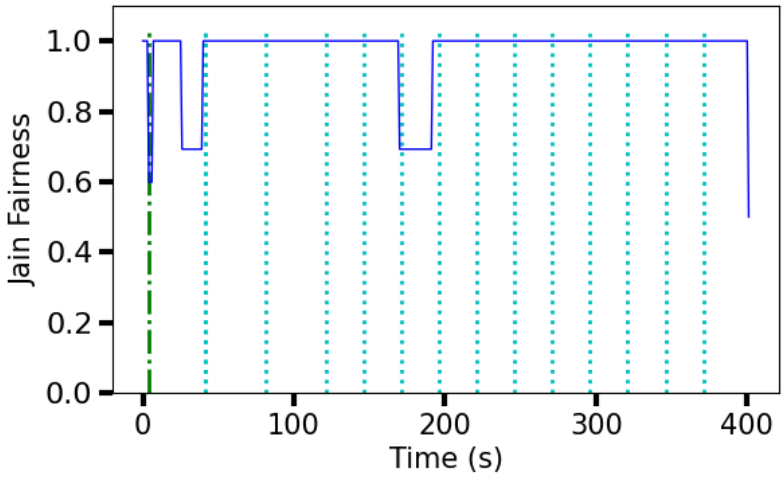
Utilization:



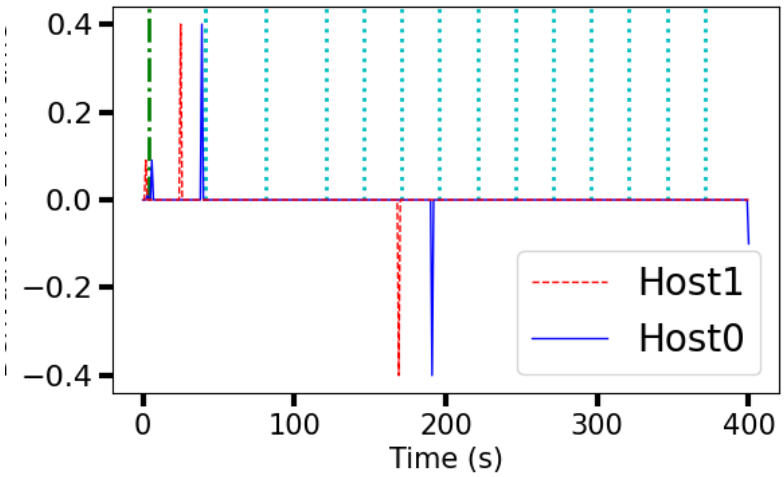
Analysis: Similar to onelink, the network is always trying to make the network fair. The fluctuation in the fairness graph and smoothness graph are mutually verifiable. The utilization is reasonable as I wrote above.

twolink, two servers, alpha = 0.1:

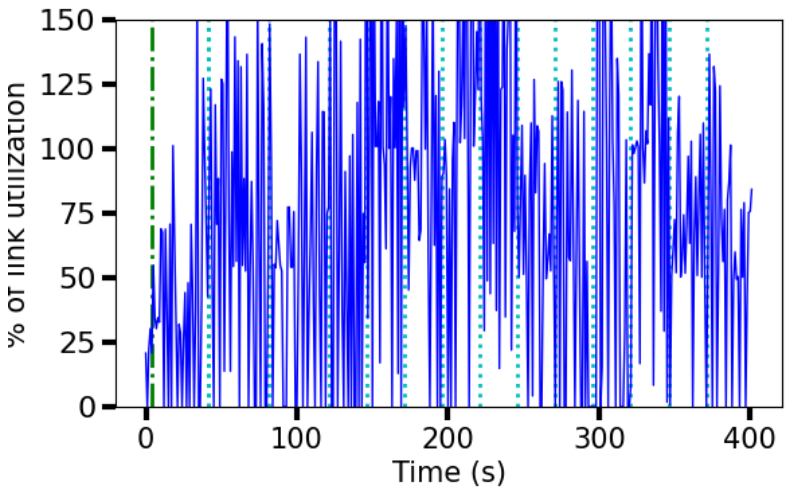
Fairness:



Smoothness:

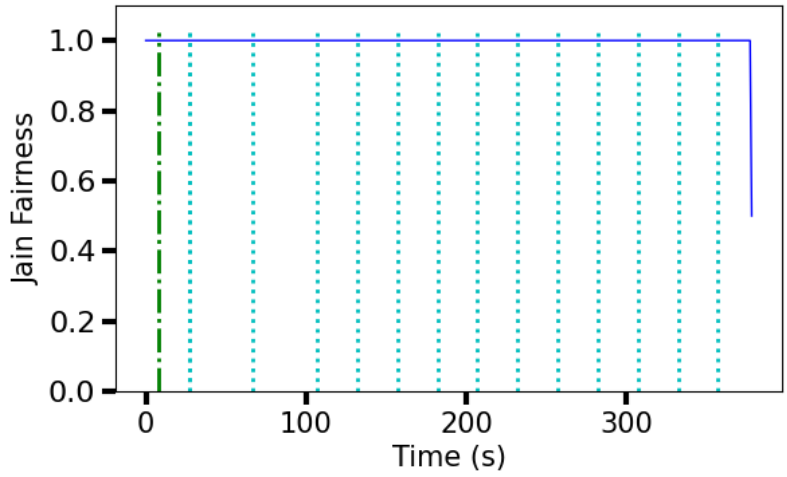


Utilization:

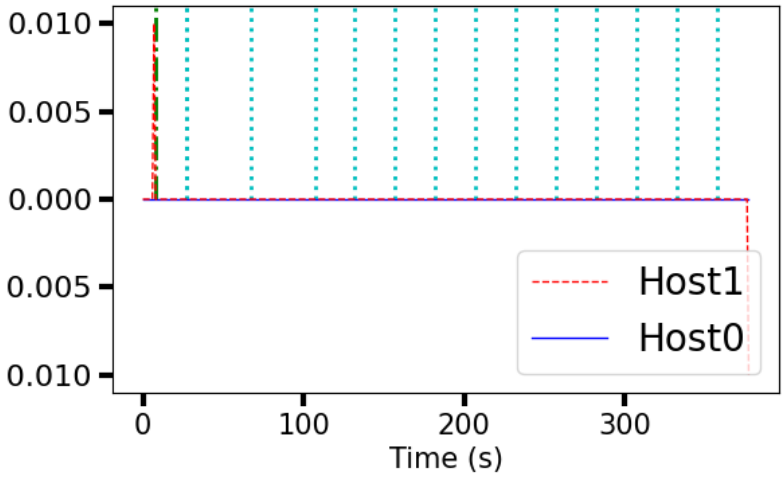


twolink, two servers, alpha = 0.5:

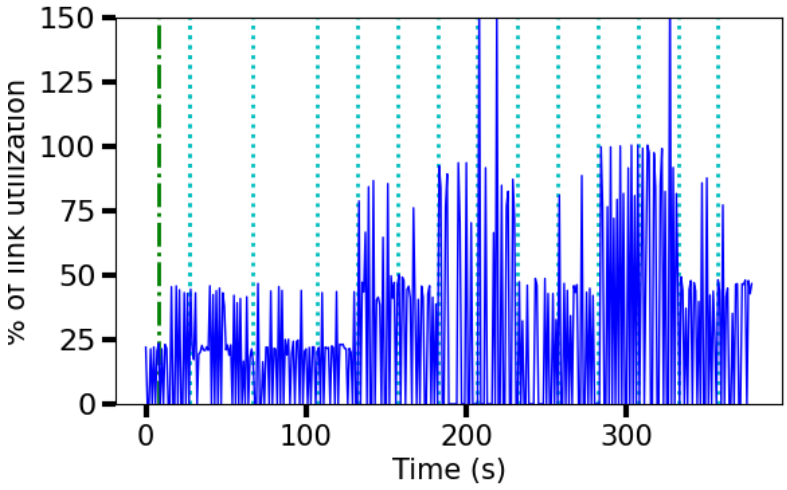
Fairness:



Smoothness:

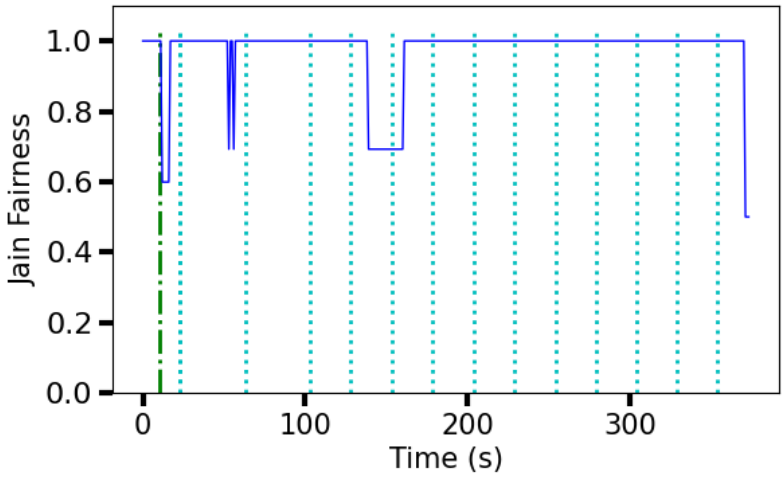


Utilization:

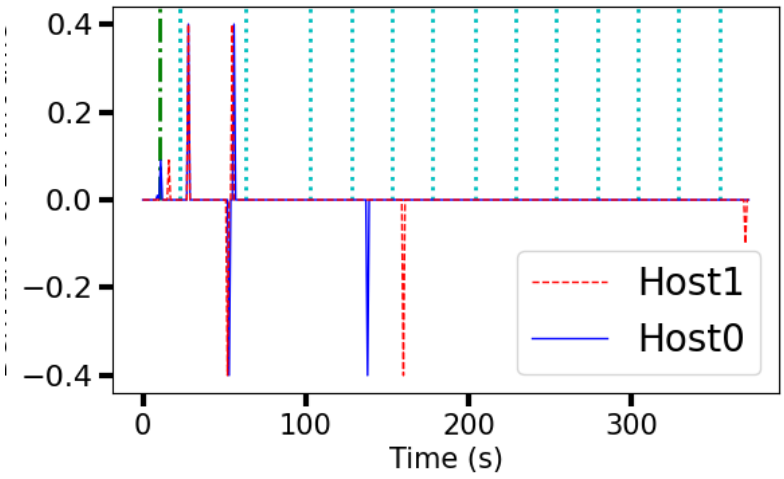


twolink, two servers, alpha = 0.9:

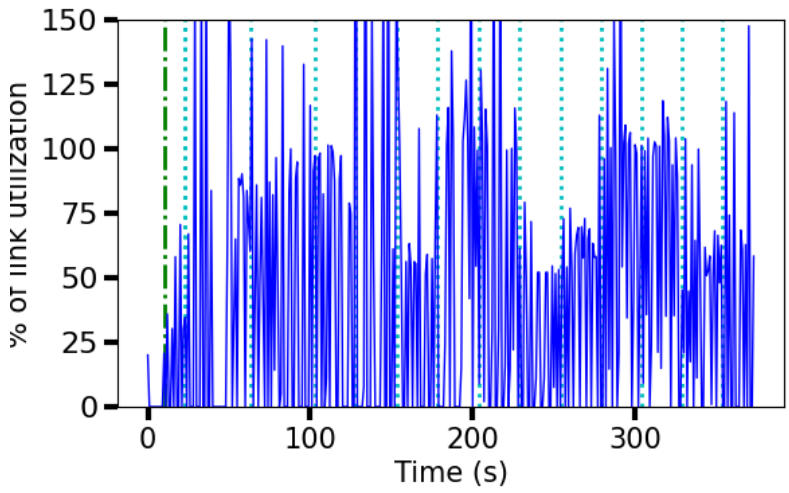
Fairness:



Smoothness:



Utilization:



Analysis: We can see the 3 situation when alpha = 0.1, 0.5, and 0.9. We can see that when alpha equals to 0.5, the fairness, smoothness and utilization are all perfect. We think alpha == 0.5 is a trade-off value. At this value, the network can predict the network condition well. When the value is 0.1, the network is too slow to respond to sudden bit rate changes. So when we suddenly change the bit rate, the network badly responds. When the value is 0.9, the network is too sensitive to the suddenly changed bit rate, which makes the network inclined to fluctuate a lot. The robustness of the network decreases.

Danmakus System Implementation (Bonus)

By adding the danmakus system, we can add some danmakus into the video.

The Danmakus system code is shown as belows.

```
@staticmethod
    @app.route('/qdanmu')
    def qdanmu():
        cookie = request.cookies
        cur_client_id = int(cookie.get("ID"))
        current_client = ClientSelf.dict_client.get(cur_client_id)
        to_return = ""
        for ele in current_client.danmulist_topush:
            to_return = to_return + str(ele) + "\r\n"
        current_client.danmulist_topush = []
        return make_response(to_return)
```

History danmakus

When a danmakus is sent by users, then the danmakus will be recorded in a list named `history_danmu`

The code is shown as belows.

```
ClientSelf.history_danmu.append(danmu)
```

Interface Beautiication (Bonus)

Changing the HTML code to make the color and size of danmakus to change in random. Then the users will find it comfortable to watch the danmakus.

```
function createDanmaku(text) {
    var timestamp = Date.parse(new Date());
    timestamp = timestamp % 255;
    var time_str_r = timestamp + "";
    var time_str_g = (timestamp + Math.random() * 100) % 255 + "";
    var time_str_b = (timestamp + Math.random() * 200) % 255 + "";
    const jqueryDom = $("<div class='bullet'>" + text + "</div>");
    const fontColor = "rgb(" + time_str_r + "," + time_str_g + "," +
time_str_b + ")";
    var fsize = (Math.random() * 20 + 20) + "";
    const fontSize = fsize + 30 + "px";
    let top = Math.floor(Math.random() * 400) + "px";
    const left = $(".screen_container").width() + "px";
    jqueryDom.css({
        "position": 'absolute',
        "color": fontColor,
        "font-size": fontSize,
        "left": left,
        "top": top,

    });
```

And We also change the margin color and add background picture, etc to make the system more beautiful.

```
body {
    position: fixed;
    top: 0;
    bottom: 0;
    left: 0;
    width: 100%;
    min-width: 1000px;
```

```
    background-image: url("https://img-qn-
3.51miz.com/preview/element/00/01/18/57/E-1185774-
ED1A979F.jpg!/quality/90/unsharp/true/compress/true/format/jpg/fh/260"
);
    background-repeat: no-repeat;
    background-size: cover;
    background-attachment: fixed;
    -webkit-background-size: cover;
    -o-background-size: cover;
    background-position: center center;
  }
```

The result page is shown as belows.