

BÀI THỰC HÀNH SỐ 2

CÁC HỆ THỐNG PHÂN TÁN VÀ ỨNG DỤNG

CHƯƠNG 2: TIỀN TRÌNH VÀ LUỒNG TRONG HỆ PHÂN TÁN

Họ và tên: Nguyễn Duy Khánh

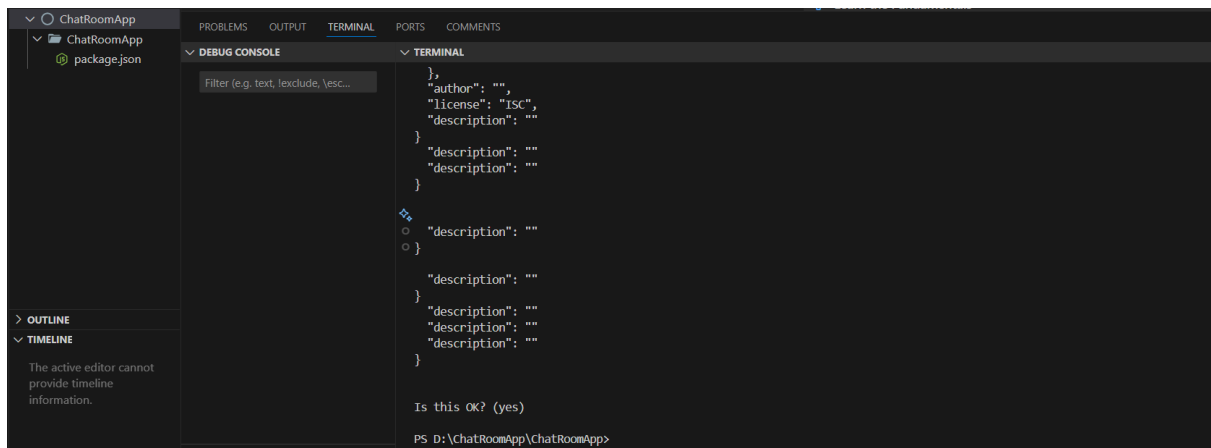
Mã lớp: 157542

MSSV: 20225019

Mã học phần: IT4611

1. Xây dựng một ChatRoom sử dụng Socket.io

Câu hỏi 1: Tập nào vừa xuất hiện trong thư mục ChatRoomApp? Nó được sử dụng để làm gì?



Tập vừa xuất hiện trong thư mục ChatRoomApp là package.json.

File package.json chứa các trường thông tin “name”, “version”, “description”, “main”, “scripts”, “author”, “license”, “dependencies”,...

File package.json để thực hiện các việc sau:

- Chỉ rõ phiên bản của project và mô tả project (tên project, tác giả, bản quyền...)
- Liệt kê các packages mà project phụ thuộc (các thư viện mà project sử dụng)
- Dễ dàng chia sẻ project giữa các developers, giúp project có thể được sử dụng lại như một thư viện.

Tạo 1 file app.js mới (nó dùng để chạy server và tất cả các gói) và đưa vào đoạn mã sau:

```
const express = require('express')
const app = express()
//set the template engine ejs
app.set('view engine', 'ejs')
//middlewares
app.use(express.static('public'))
//routes
```

```
app.get('/', (req, res) => {  
  res.send('Hello world')  
})  
//Listen on port 3000  
server = app.listen(3000)
```

Câu hỏi 2: Mở trình duyệt và gõ vào đó địa chỉ *http://localhost:3000*, bạn sẽ nhận được thông điệp gì?



Mở trình duyệt và gõ vào đó địa chỉ <http://localhost:3000>, nhận được thông điệp **Hello world**

Thêm các đoạn mã sau vào file *app.js*:

```
//socket.io instantiation  
const io = require("socket.io")(server)  
//listen on every connection  
io.on('connection', (socket) => {  
  console.log('New user connected')  
})
```

Câu hỏi 3: Bạn hãy thử reload (Ctrl-R) lại trình duyệt. Bạn có nhìn thấy gì mới xuất hiện trên cửa sổ không? Nếu không có gì xuất hiện hết thì là vì sao?

Không có gì mới xuất hiện ở cửa sổ cả.

Nguyên nhân: Vì ta mới chỉ cài đặt socket.io cho server chứ chưa cài đặt socket.io cho client.

Trong thư mục ChatRoomApp của bạn, hãy tạo 2 thư mục con mới tên là *public* và *views*.

```
>mkdir public
```

```
>mkdir views
```

Trong thư mục views, hãy tạo tệp tên là index.ejs Trong thư mục views, hãy tải file index.ejs từ đường link: <https://github.com/anhth318/ChatRoomApp/blob/master/views/index.ejs>

Trong thư mục *public*, hãy tạo file *chat.js* với nội dung sau:

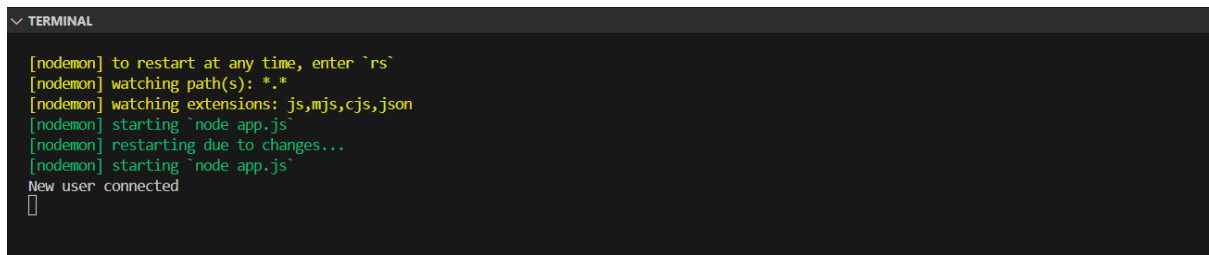
```
$ (function () {  
    //make connection  
    var socket = io.connect('http://localhost:3000')  
});
```

(chú ý là nếu bạn làm việc với 2 máy tính thì hãy thay localhost bằng địa chỉ IP của server).

Tải file *style.css* và đặt nó trong thư mục *public*:

<https://github.com/anhth318/ChatRoomApp/blob/master/public/style.css>

Câu hỏi 4: Refresh trang localhost:3000, bạn nhìn thấy thông điệp nào?



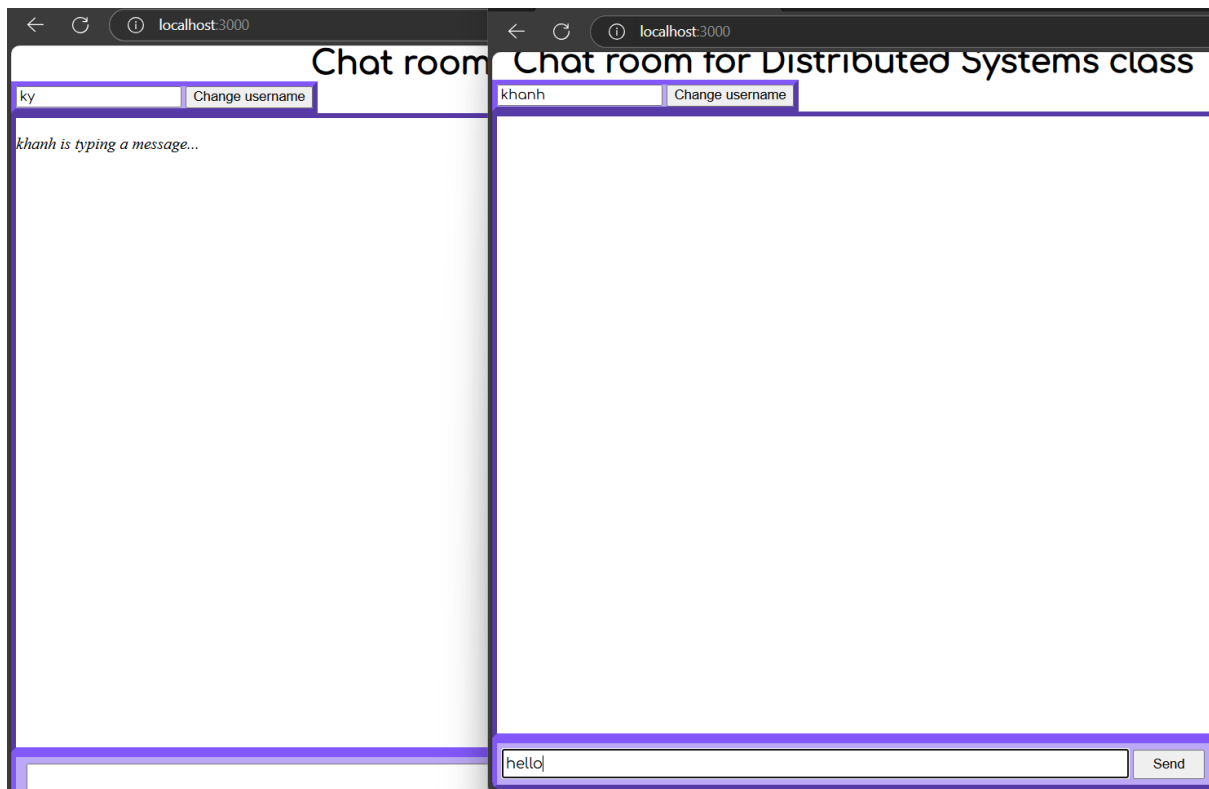
```
▼ TERMINAL  
[nodemon] to restart at any time, enter `rs`  
[nodemon] watching path(s): *.*  
[nodemon] watching extensions: js,mjs,cjs,json  
[nodemon] starting `node app.js`  
[nodemon] restarting due to changes...  
[nodemon] starting `node app.js`  
New user connected  
█
```

Ở cửa sổ console/terminal sẽ xuất hiện dòng chữ “New user connected”

Ở phía client, bây giờ bạn muốn làm chủ một số hành động khác như: gửi tin nhắn, nghe tin nhắn mới, và gửi đi username.

Bây giờ bạn có thể thử refresh lại cửa sổ trình duyệt của bạn để thử nghiệm chat room.

Câu hỏi 5: Bây giờ bạn hãy thử gõ gì đó lên một tab. Cùng lúc đó, nhìn sang tab khác của người dùng khác, bạn thấy gì?



Khi đó, ở bên giao diện bên người dùng khác sẽ thấy dòng chữ “(ten_username) is typing a

message”.

Còn khi tin nhắn được gửi đi, phía bên kia sẽ nhận được tin nhắn có username và nội dung tin nhắn người gửi

2. Phát triển hệ thống RPC sử dụng RabbitMQ

Hãy tạo 1 thư mục làm việc mới, copy hết 3 files trên vào đó. Hệ thống RPC mà chúng ta sắp xây dựng sẽ được mô tả như sau:

Client gửi yêu cầu đến hàng đợi *rpc_queue* và tạo ra một hàng đợi riêng của nó để chờ kết quả trả về từ Server. Sau khi nhận yêu cầu từ Client, Server sẽ xử lý yêu cầu và sau đó gửi trả về câu trả lời cho Client vào hàng đợi tương ứng. Ở bài thực hành này, Server sẽ vận hành để tính chuỗi Fibonacci.

Bây giờ bạn đã có thể sẵn sàng để xây dựng Client và Server.

Client:

Vào thư mục làm việc mà đang có 3 files vừa rồi. Tạo một lớp `RPCClient` bằng cách tạo file `RPCClient.java`.

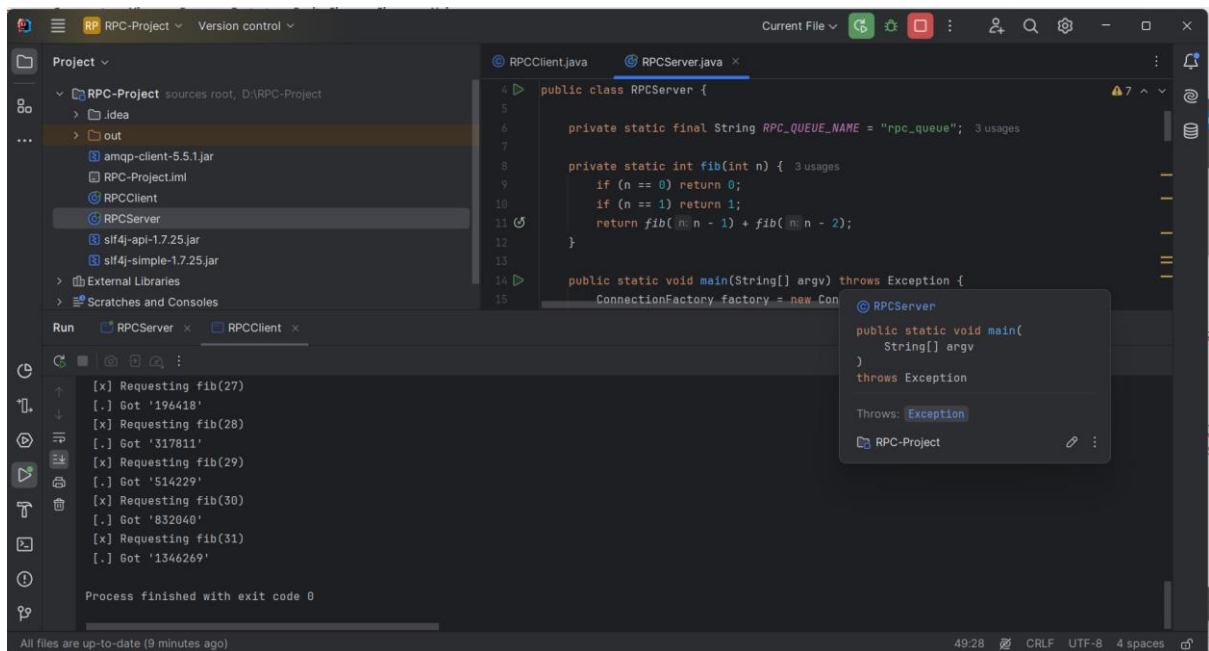
Client sẽ gửi 32 lần lên Server để yêu cầu 32 giá trị của chuỗi số Fibonacci.

Server:

Tạo lớp `RPCServer` bằng cách tạo file `RPCServer.java` ở trong thư mục làm việc. Thêm đoạn code sau và tự mình hoàn thiện phương thức `fib` (để sinh ra số thứ *n* của chuỗi fibonacci)

Đoạn code để hoàn thiện phương thức `fib`:

```
private static int fib(int n) {  
    if (n == 0) return 0;  
    if (n == 1) return 1;  
    return fib(n - 1) + fib(n - 2);  
}
```



Câu hỏi 6: Đây là đoạn code mà Server gán *correlationID* vào câu trả lời?

Server cần gán *correlationId* vào phản hồi để Client có thể xác định đúng yêu cầu nào được phản hồi. Đoạn code này nằm ở phần tạo *replyProps* trong *deliverCallback*:

Đoạn *delivery.getProperties().getCorrelationId()* lấy *correlationId* từ yêu cầu của Client. Sau đó, tạo *replyProps* mới với *correlationId* giống yêu cầu ban đầu. Khi Server gửi kết quả về hàng đợi phản hồi (*replyTo*), nó sẽ gán *correlationId* vào *replyProps*.

```

28 DeliverCallback deliverCallback = (consumerTag, delivery) -> {
29     AMQP.BasicProperties replyProps = new AMQP.BasicProperties
30         .Builder()
31         .correlationId(delivery.getProperties().getCorrelationId())
32         .build();

```

Câu hỏi 7: Dựa vào cả code của Client và Server để giải thích đây là đoạn code mà Client gửi yêu cầu lên cho Server thông qua hàng đợi *rpc_queue* và tạo ra một hàng đợi mới để chờ câu trả lời của Server.

Đoạn code mà Client gửi yêu cầu lên Server (*rpc_queue*) nằm ở *RPCClient.java*:

```

49 channel.basicPublish( s: "", requestQueueName, props, message.getBytes( charsetName: "UTF-8"));

```

với *requestQueueName* = "*rpc_queue*" là hàng đợi mà Client gửi yêu cầu lên.

Đoạn code tạo hàng đợi tạm thời để nhận phản hồi (Client chờ Server):

```

42 String replyQueueName = channel.queueDeclare().getQueue();

```

Channel.queueDeclare().getQueue() sẽ tạo một hàng đợi tạm thời để chờ phản hồi từ Server. Đoạn code này dùng để phía Client nhận phản hồi từ hàng đợi tạm thời *channel.basicConsume(replyQueueName, true, deliverCallback, consumerTag -> {})*;

Dịch và chạy chương trình:

Bây giờ chúng ta sẽ dịch và chạy chương trình. Đầu tiên, hãy xây dựng chương trình Client và Server bằng cách sử dụng classpath (-cp) gắn với file thư viện client mà bạn vừa tải về lúc

bạn đầu:

```
>javac -cp amqp-client-5.5.1.jar RPCClient.java
```

RPCServer.java

Mở thêm một cửa sổ command và chạy Server. Hãy chắc rằng bạn đã thêm được biến môi trường *CP* như ở trên có hướng dẫn:

```
>java -cp %CP% RPCServer
```

(ghi chú: trong trường hợp không muốn tạo biến môi trường *CP* như trên thì có thể thay thế *CP* bằng đoạn giá trị đó).

Mở một cửa sổ tách biệt và chạy Client:

```
>java -cp %CP% RPCClient
```

```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS D:\RPC-Project> javac -cp "amqp-client-5.5.1.jar;slf4j-api-1.7.25.jar;slf4j-simple-1.7.25.jar" RPCClient.java RPCServer.java
PS D:\RPC-Project> java -cp "amqp-client-5.5.1.jar;slf4j-api-1.7.25.jar;slf4j-simple-1.7.25.jar;." RPCServer
[x] Awaiting RPC requests
[.] fib(0)
[.] fib(1)
[.] fib(2)
[.] fib(3)
[.] fib(4)
[.] fib(5)
[.] fib(6)
[.] fib(7)
[.] fib(8)
[.] fib(9)
[.] fib(10)
[.] fib(11)
[.] fib(12)
[.] fib(13)
[.] fib(14)
[.] fib(15)
[.] fib(16)
[.] fib(17)
[.] fib(18)
[.] fib(19)
[.] fib(20)

Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS D:\RPC-Project> java -cp "amqp-client-5.5.1.jar;slf4j-api-1.7.25.jar;slf4j-simple-1.7.25.jar;." RPCClient
[x] Requesting fib(0)
[.] Got '0'
[x] Requesting fib(1)
[.] Got '1'
[x] Requesting fib(2)
[.] Got '1'
[x] Requesting fib(3)
[.] Got '2'
[x] Requesting fib(4)
[.] Got '3'
[x] Requesting fib(5)
[.] Got '5'
[x] Requesting fib(6)
[.] Got '8'

Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS D:\RPC-Project> java -cp "amqp-client-5.5.1.jar;slf4j-api-1.7.25.jar;slf4j-simple-1.7.25.jar;." RPCServer
[x] Awaiting RPC requests
[.] fib(0)
[.] fib(1)
[.] fib(2)
[.] fib(0)
[.] fib(3)
[.] fib(1)
[.] fib(4)
[.] fib(0)
[.] fib(2)
[.] fib(5)]
```

Câu hỏi 8: Bây giờ hãy thử thêm một chút delay vào chương trình Server bằng cách thêm vào đoạn code sau ở dưới dòng:

```
response += fib(n);
try {
    Thread.sleep(2000);
} catch (InterruptedException _ignored) {
```

```

        Thread.currentThread().interrupt();
    }

```

Chương trình Server sẽ ngủ 2s đối với mỗi request. Hãy dịch lại chương trình Server và chạy nó.

Mở cùng lúc nhiều cửa sổ command và chạy nhiều chương trình Client trên đó cùng lúc.

Cùng lúc đó mở một cửa sổ command khác và chạy dòng lệnh sau:

```

>rabbitmqctl.bat list_queues name messages_ready
messages_unacknowledged

```

Bạn nhận được kết quả hiển thị gì? Giải thích!

Giả sử cho 4 Client gửi yêu cầu lên Server đồng thời:

Tại thời điểm sau đó, ta thu được kết quả sau:

```

PS D:\RPC-Project> rabbitmqctl.bat list_queues name messages_ready messages_unacknowledged
Timeout: 60.0 seconds ...
Listing queues for vhost / ...
name      messages_ready  messages_unacknowledged
amq.gen-HWLy_oc6dNoCWEJ_nKN_ow  0      0
rpc_queue      3      1
amq.gen-ciKg5P5lg4jUiFTSk3XgPw  0      0
amq.gen-_cpMGwbKWZHNtZrw161X8w  0      0
amq.gen-53N6J0xrrg5h145QfFmnsA  0      0
PS D:\RPC-Project> |

```

Cột name: Tên của hàng đợi trong RabbitMQ.

Cột messages_ready: Số lượng tin nhắn đang chờ xử lý

Cột messages_unacknowledged: Số lượng tin nhắn đã được nhận bởi Server nhưng chưa xác nhận (ACK)

Ở rpc_queue: Có 3 yêu cầu từ Client đang chờ được xử lý, Có 1 yêu cầu đã được Server nhận nhưng chưa xử lý xong. Vì server có độ trễ do ta đã cài đặt Thread.sleep(2000) nên nó chưa xử lý xong yêu cầu đầu tiên

Các hàng đợi tạm thời có messages_ready = 0 và messages_unacknowledged = 0 tức là Client đã nhận phản hồi từ Server, không có tin nhắn đang chờ

Khi Server xử lý xong công việc cho một client thì số lượng hàng đợi sẽ giảm xuống.

3. Phân tích ảnh hưởng của các thông số QoS đến dịch vụ

Hai máy sử dụng trong bài thực hành có thể là 2 máy ảo (tạo bởi virtualbox) hoặc là 2 máy thật của 2 bạn trong nhóm. Cài phần mềm VLC lên 2 máy bằng câu lệnh sau:

```

$ sudo apt-get install vlc

```

Đầu tiên phải đảm bảo 2 máy có thể kết nối với nhau bằng cách dùng công cụ *ping* để thử.

Câu hỏi 9: Hãy thực hiện gõ những lệnh tương ứng như trên với 3 dịch vụ còn lại:


```
khanhbkit1@khanhskii04-VirtualBox: ~  
khanhbkit1@khanhskii04-VirtualBox:~$ ip a  
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000  
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00  
    inet 127.0.0.1/8 scope host lo  
        valid_lft forever preferred_lft forever  
    inet6 ::1/128 scope host noprefixroute  
        valid_lft forever preferred_lft forever  
2: enp0s3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000  
    link/ether 08:00:27:e9:31:3e brd ff:ff:ff:ff:ff:ff  
    inet 10.0.2.15/24 brd 10.0.2.255 scope global dynamic noprefixroute enp0s3  
        valid_lft 86271sec preferred_lft 86271sec  
    inet6 fd00::9323:d67b:a793:47c3/64 scope global temporary dynamic  
        valid_lft 86273sec preferred_lft 14273sec  
    inet6 fd00::a00:27ff:fee9:313e/64 scope global dynamic mngtmpaddr  
        valid_lft 86273sec preferred_lft 14273sec  
    inet6 fe80::a00:27ff:fee9:313e/64 scope link  
        valid_lft forever preferred_lft forever  
3: enp0s8: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000  
    link/ether 08:00:27:5a:31:32 brd ff:ff:ff:ff:ff:ff  
    inet 192.168.56.102/24 brd 192.168.56.255 scope global dynamic noprefixroute enp0s8
```

IP Server: 192.168.56.101

IP Client: 192.168.56.102

Sử dụng lệnh `ip a` để lấy địa chỉ IP của các máy

Sau đó sử dụng lệnh `ping + IP_Address` để thực hiện ping

Câu hỏi 10: Bạn đã xem được video trên máy client chưa? Đánh giá chất lượng video mà bạn xem trên máy client.

Tải một số video về máy Server từ website <http://www.open-video.org/> Bây giờ chúng ta sẽ sử dụng VLC để truyền dòng video từ server đến client. Để làm được điều đó thì ở máy server chúng ta cho chạy dòng lệnh sau:

```
$vlc -vvv input_stream --sout  
'#standard{access=http,mux=ogg,dst=SERVER_IP:8080}'
```

Trong đó:

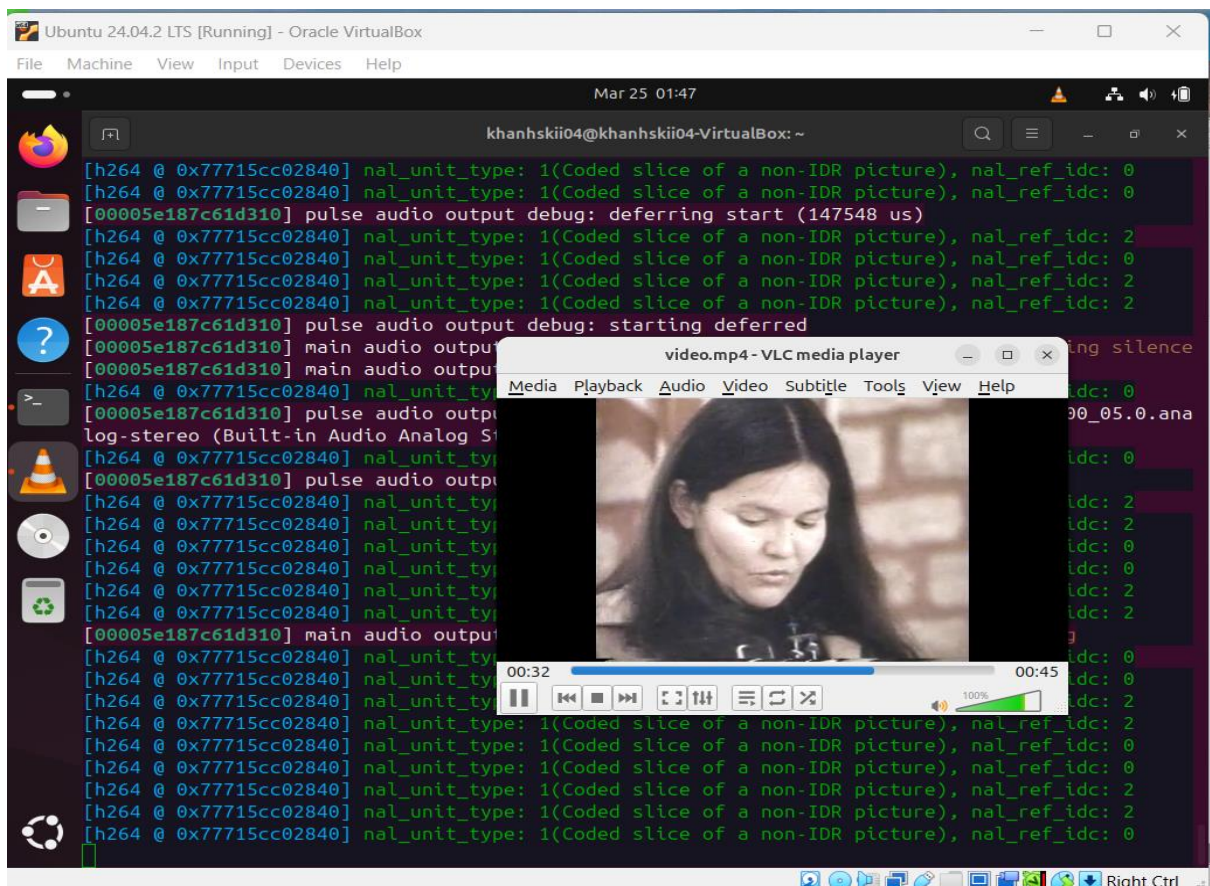
`input_stream` là tên file video của bạn muốn phát

`SERVER_IP` là địa chỉ IP của Server.

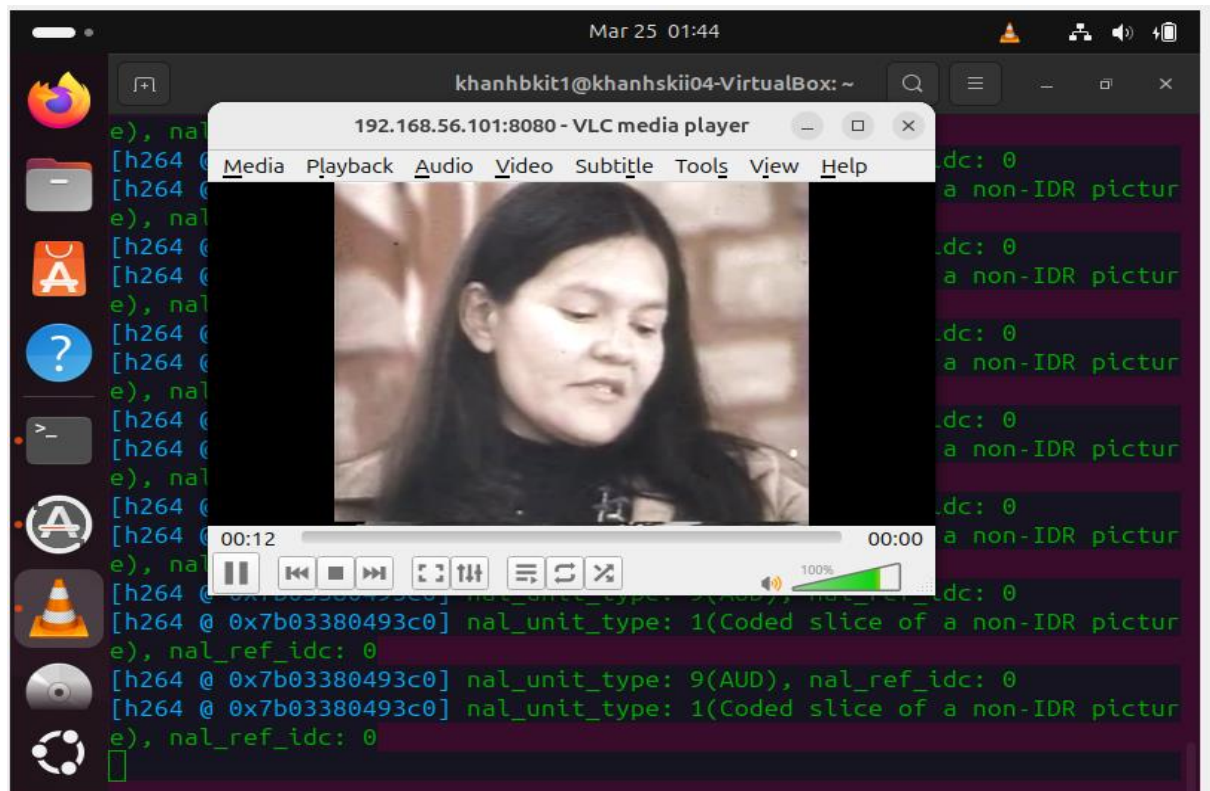
Ở máy Client thì chạy lệnh sau:

```
$vlc http://SERVER\_IP:8080
```


Đây là video tải về trực tiếp từ Website:



Đây là video xem được trên máy Client



Trên máy Client đã xem được video stream từ Server. Ta có thể kết luận chất lượng video trên máy Client gần tương đương với chất lượng video gốc.

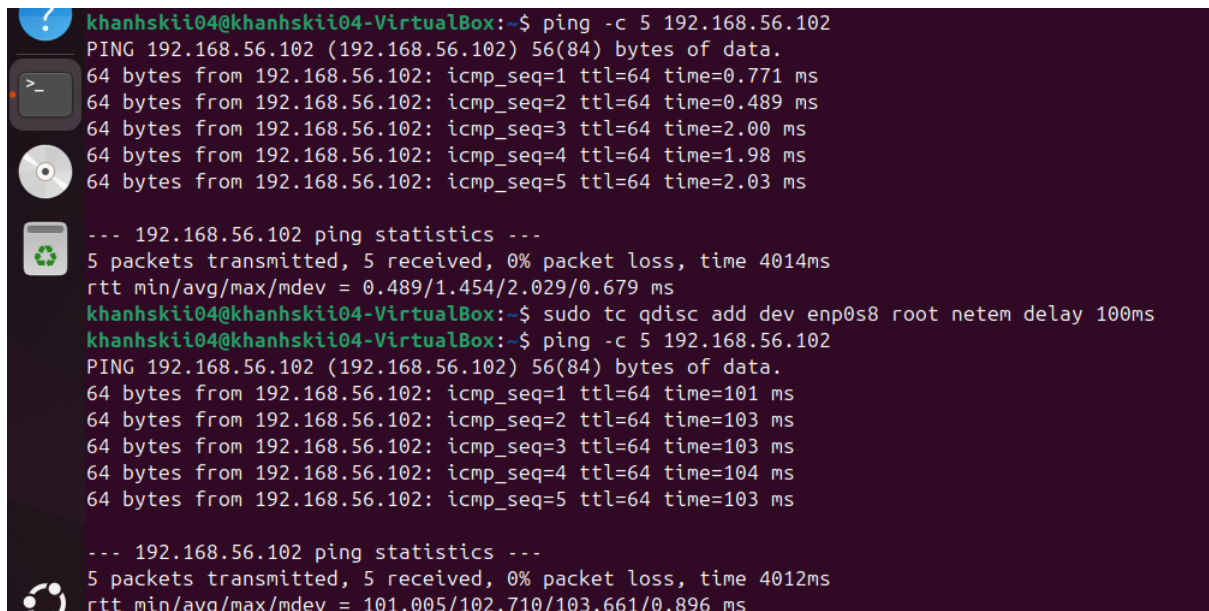
Bây giờ trong quá trình truyền dòng video, chúng ta sẽ chủ động thay đổi các thông số QoS để thay đổi chất lượng mạng, từ đó quan sát chất lượng video ở máy client. Để thực hiện điều đó thì chúng ta sẽ sử dụng công cụ NetEm.

Cài đặt NetEm trên máy Server như sau:

```
$sudo apt-get install iproute $sudo modprobe sch_netem
```

Bây giờ chúng ta sẽ tiến hành các câu lệnh để thay đổi các thông số QoS. Các câu lệnh tiếp theo đây sẽ thực hiện ở máy Server và có thể gõ song song với quá trình đang truyền video.

Câu hỏi 11: Kết quả nhận được sau lệnh ping là gì? Bạn có thấy độ trễ đã tăng 100ms không?



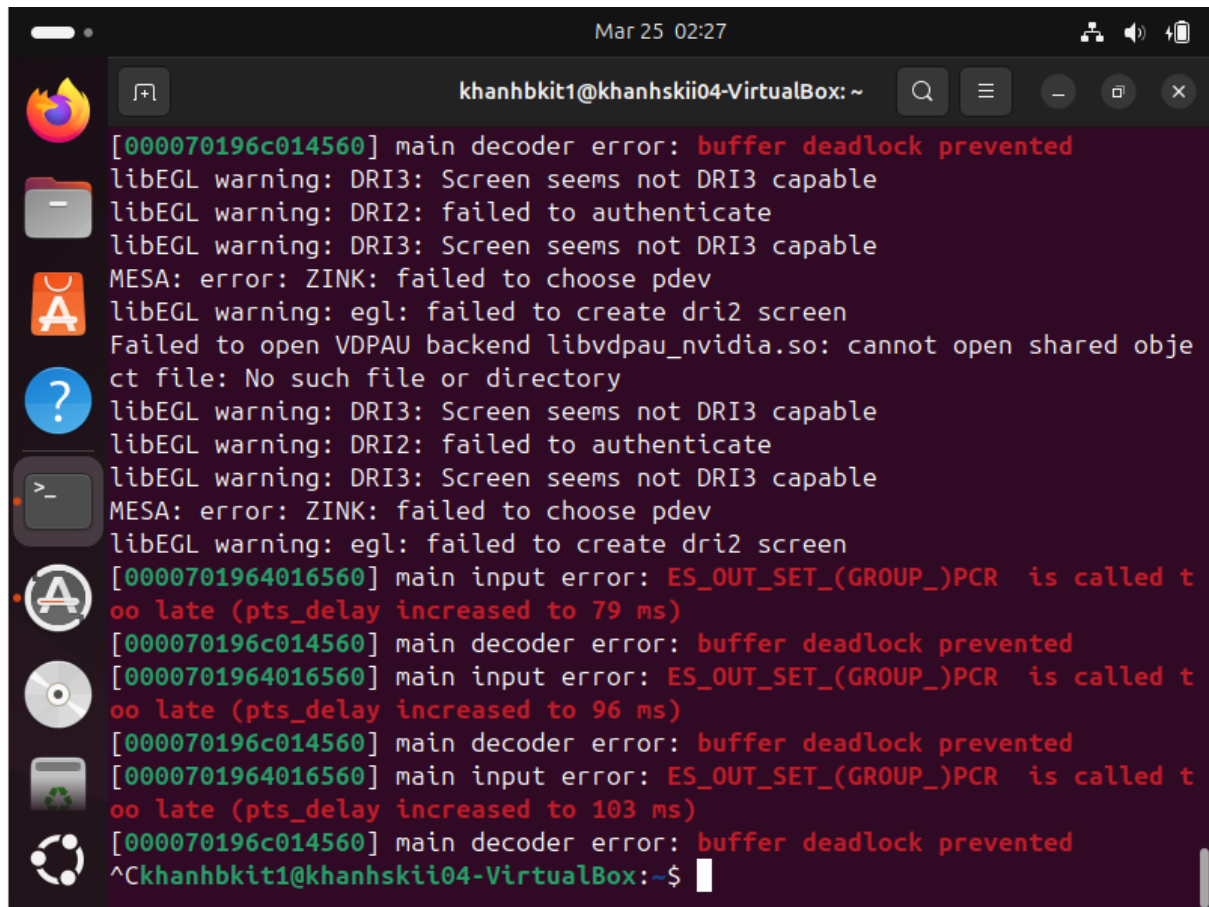
```
khanhskii04@khanhskii04-VirtualBox:~$ ping -c 5 192.168.56.102
PING 192.168.56.102 (192.168.56.102) 56(84) bytes of data:
64 bytes from 192.168.56.102: icmp_seq=1 ttl=64 time=0.771 ms
64 bytes from 192.168.56.102: icmp_seq=2 ttl=64 time=0.489 ms
64 bytes from 192.168.56.102: icmp_seq=3 ttl=64 time=2.00 ms
64 bytes from 192.168.56.102: icmp_seq=4 ttl=64 time=1.98 ms
64 bytes from 192.168.56.102: icmp_seq=5 ttl=64 time=2.03 ms

--- 192.168.56.102 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4014ms
rtt min/avg/max/mdev = 0.489/1.454/2.029/0.679 ms
khanhskii04@khanhskii04-VirtualBox:~$ sudo tc qdisc add dev enp0s8 root netem delay 100ms
khanhskii04@khanhskii04-VirtualBox:~$ ping -c 5 192.168.56.102
PING 192.168.56.102 (192.168.56.102) 56(84) bytes of data:
64 bytes from 192.168.56.102: icmp_seq=1 ttl=64 time=101 ms
64 bytes from 192.168.56.102: icmp_seq=2 ttl=64 time=103 ms
64 bytes from 192.168.56.102: icmp_seq=3 ttl=64 time=103 ms
64 bytes from 192.168.56.102: icmp_seq=4 ttl=64 time=104 ms
64 bytes from 192.168.56.102: icmp_seq=5 ttl=64 time=103 ms

--- 192.168.56.102 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4012ms
rtt min/avg/max/mdev = 101.005/102.710/103.661/0.896 ms
```

Trước khi thực hiện áp dụng NetEm, mạng có độ trễ rất thấp, khoảng 1,5ms (avg = 1.454ms), sau khi áp dụng NetEm với delay 100ms, tất cả gói tin đều có độ trễ tăng hơn hẳn (avg = 102.710ms), độ trễ đã tăng lên khoảng 100ms.

Câu hỏi 12: Hãy tắt chức năng sử dụng bộ đệm ở máy Client. Sau đó hãy đánh giá chất lượng của video nhận được ở máy Client. Bạn kết luận thế nào về ảnh hưởng của delay với dịch vụ truyền dòng video?



```
Mar 25 02:27
khanhbkit1@khanhskii04-VirtualBox: ~
[000070196c014560] main decoder error: buffer deadlock prevented
libEGL warning: DRI3: Screen seems not DRI3 capable
libEGL warning: DRI2: failed to authenticate
libEGL warning: DRI3: Screen seems not DRI3 capable
MESA: error: ZINK: failed to choose pdev
libEGL warning: egl: failed to create dri2 screen
Failed to open VDPAU backend libvdpau_nvidia.so: cannot open shared object file: No such file or directory
libEGL warning: DRI3: Screen seems not DRI3 capable
libEGL warning: DRI2: failed to authenticate
libEGL warning: DRI3: Screen seems not DRI3 capable
MESA: error: ZINK: failed to choose pdev
libEGL warning: egl: failed to create dri2 screen
[0000701964016560] main input error: ES_OUT_SET_(GROUP_)PCR is called too late (pts_delay increased to 79 ms)
[000070196c014560] main decoder error: buffer deadlock prevented
[0000701964016560] main input error: ES_OUT_SET_(GROUP_)PCR is called too late (pts_delay increased to 96 ms)
[000070196c014560] main decoder error: buffer deadlock prevented
[0000701964016560] main input error: ES_OUT_SET_(GROUP_)PCR is called too late (pts_delay increased to 103 ms)
[000070196c014560] main decoder error: buffer deadlock prevented
^Ckhanhbkit1@khanhskii04-VirtualBox:~$
```

Nó hiển thị lỗi như bên trên, PCR (Program Clock Reference) là một tham số giúp đồng bộ hóa video/audio. Khi độ trễ cao, VLC nhận gói tin video quá muộn, nên nó tăng pts_delay để cố đồng bộ. Nếu pts_delay tăng liên tục, video sẽ bị giật, mất khung hình. Vì vậy, khi tắt bộ đệm phía Client và để delay ở phía Server 100ms, ta sẽ thấy hình ảnh bị giật lag, có thể bị khựng lại, khiến chất lượng video giảm.

Các lệnh tiếp tới đây chúng ta sẽ chỉ thay thế các giá trị chứ không khởi động lại chương trình *qdisc* (nghĩa là chỉ dùng option *change* thay vì *add*).

Chúng ta biết rằng ở môi trường mạng thật thì các thông số mạng không cố định như vậy, vì vậy chúng ta sẽ thêm vào một giá trị biến đổi cho chúng:

```
$ sudo tc qdisc change dev eth0 root netem delay 100ms 10ms
```

Tác dụng của lệnh trên là tạo ra độ trễ thay đổi $100\text{ms} \pm 10\text{ms}$. Sự biến đổi giá trị delay của mạng không hoàn toàn là ngẫu nhiên, vì vậy chúng ta sẽ thêm vào một giá trị tương quan:

```
$ sudo tc qdisc change dev eth0 root netem delay 100ms 10ms 25%
```

Câu hỏi 13: Cũng như câu hỏi 7, hãy quan sát video ở Client và đưa ra đánh giá và kết luận về ảnh hưởng của độ biến đổi delay lên chất lượng dịch vụ truyền video?

Delay cao làm video bị giật, méo tiếng hoặc mất khung hình. Nếu độ trễ biến đổi hoàn toàn ngẫu nhiên thì chất lượng video có lúc tốt, lúc tệ, còn khi thêm giá trị tương quan và giới hạn phạm vi thay đổi thì chất lượng sẽ ổn định hơn

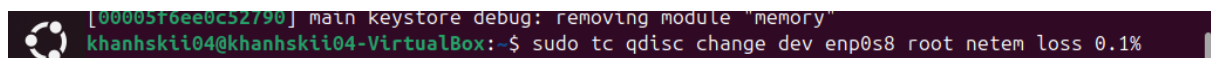
Mất gói tin (Packet loss):

Giá trị mất mát gói tin ngẫu nhiên được xác định trong lệnh tc dựa trên đơn vị phần trăm (%). Giá trị dương nhỏ nhất là 0.0000000232%

Bây giờ hãy thử cho 1/1000 gói tin bị mất ngẫu nhiên:

```
$ sudo tc qdisc change dev eth0 root netem loss 0.1%
```

Câu hỏi 14: Hãy xem video ở client và đánh giá về độ ảnh hưởng của packet loss lên chất lượng dịch vụ truyền video. Thử tăng giá trị của tỷ lệ mất gói tin lên để thấy độ ảnh hưởng rõ nét hơn



```
[00005f6ee0c52790] main keystore debug: removing module "memory"
khanhskii04@khanhskii04-VirtualBox:~$ sudo tc qdisc change dev enp0s8 root netem loss 0.1%
```

Với tỉ lệ packet loss 0.1% thì chất lượng video hầu như không đổi, với tỉ lệ 1% thì rất hiếm khi video bị giật nhẹ, với packet loss 10% thì video có lúc bị đứng hình 3s, âm thanh bị mất, ta có thể kết luận tỉ lệ packet loss cao sẽ ảnh hưởng trực tiếp đến chất lượng video(mờ, âm thanh ngắt quãng...)

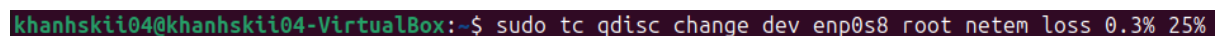
Giá trị tương quan có thể đưa vào để làm giảm tính ngẫu nhiên của việc mất gói tin:

```
$ sudo tc qdisc change dev eth0 root netem loss 0.3% 25%
```

Câu lệnh trên có nghĩa là 0.3% của các gói tin sẽ bị mất, và mỗi xác suất để gói tin kế tiếp sẽ phụ thuộc 25% vào xác suất mất của gói phía trước:

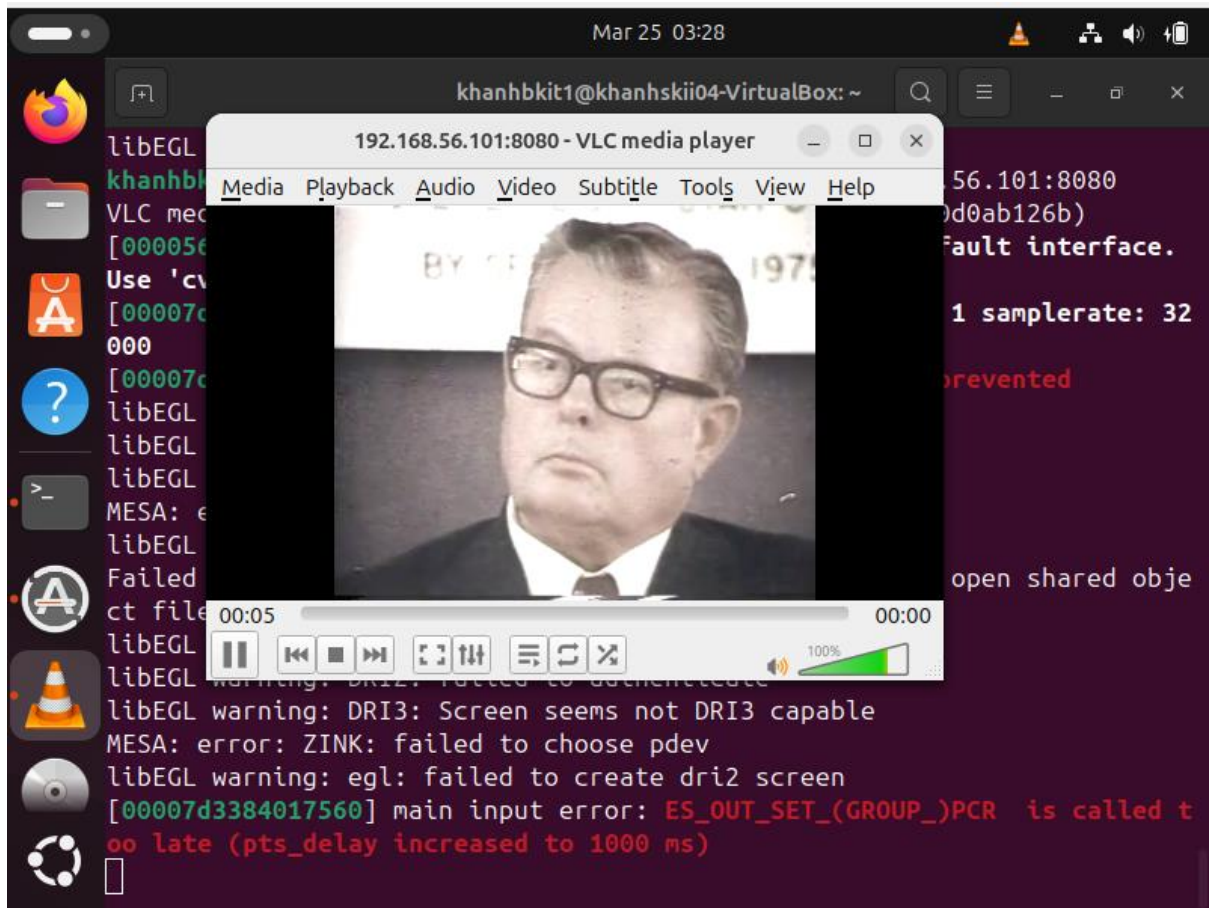
$$\text{Probn} = 0.25 * \text{Probn-1} + 0.75 * \text{Random}$$

Câu hỏi 15: Hãy xem video ở client và đánh giá về độ ảnh hưởng của packet loss lên chất lượng dịch vụ truyền video. Thử tăng giá trị của tỷ lệ mất gói tin lên để thấy độ ảnh hưởng rõ nét hơn



```
khanhskii04@khanhskii04-VirtualBox:~$ sudo tc qdisc change dev enp0s8 root netem loss 0.3% 25%
```

Với tỉ lệ mất gói dưới 1%, video gần như không bị ảnh hưởng, xem bằng mắt thường không phát hiện sự khác biệt, với tỉ lệ 5 – 10%, video có sự giật lag nhẹ, nhưng ít, với tỉ lệ mất gói ở mức 30% trở lên thì video không thể phát được, đứng hình nghiêm trọng như hình dưới



Lặp gói tin (Packet duplication)

Cơ chế lặp cũng tương tự cơ chế mất gói tin:

```
$ sudo tc qdisc change dev eth0 root netem duplicate 1%
```

Câu hỏi 16: Hãy xem video ở client và đánh giá về độ ảnh hưởng của packet loss lên chất lượng dịch vụ truyền video. Thử tăng giá trị của tỷ lệ mất gói tin lên để thấy độ ảnh hưởng rõ nét hơn

Lặp gói tin không ảnh hưởng đến việc mất khung hình, nhưng có thể gây ra mất đồng bộ hóa giữa âm thanh và hình ảnh.

Nếu tỷ lệ lặp gói cao, client có thể nhận dữ liệu dư thừa, dẫn đến việc âm thanh bị lặp lại hoặc méo tiếng. Nếu lặp gói quá cao, khoảng 25% - 30% video có thể bị lỗi nặng, hình và tiếng lệch quá nhiều

Các lỗi ngẫu nhiên có thể được làm giả lập như sau:

```
$ sudo tc qdisc change dev eth0 root netem corrupt 0.1%
```

Lỗi đảo thứ tự gói tin (Packet re-ordering)

Có 2 cách để thực hiện đảo thứ tự các gói tin. Cách đầu tiên là sử dụng chuỗi cố định và đảo gói tin thứ N. Ví dụ như câu lệnh sau:

```
$ sudo tc qdisc change dev eth0 root netem gap 5 delay 10ms
```

Có nghĩa cứ gói tin thứ 5 (thứ 10, 15, ...) sẽ được gửi ngay lập tức còn các gói tin khác thì bị delay 10ms.

Cách thứ 2 là thay đổi thứ tự một cách ngẫu nhiên, giống môi trường mạng thật hơn. Nó gây ra một số phần trăm nhất định của các gói tin bị đảo lộn thứ tự:

```
$ sudo tc qdisc change dev eth0 root netem delay 10ms reorder  
25% 50%
```

Ở câu lệnh trên thì 25% các gói tin (với độ tương quan là 50%) sẽ được gửi ngay lập tức, còn các gói tin khác thì bị chậm 10ms.

Câu hỏi 17: Hãy xem video ở client và đánh giá về độ ảnh hưởng của việc đảo thứ tự gói tin lên chất lượng dịch vụ truyền video.

```
khanhskii04@khanhskii04-VirtualBox:~$ sudo tc qdisc change dev enp0s8 root netem delay 100ms r  
eorder 25% 50%
```

Với việc đảo thứ tự gói tin thì sẽ gây nên hiện tượng âm thanh bị lệch so với hình ảnh, méo hình, nếu tỉ lệ đảo thứ tự cao thì video có thể không phát được do quá nhiều gói tin không đến đúng thứ tự, vì Video Streaming sử dụng giao thức UDP.