Ahmedabad
University

CSE342 Computer Networks

Project Report

Submitted to faculty: Shashi Prabh

Date of Submission: 09-05-2022

Group Member Details

Roll No.	Name of the Student	Name of the Program
AU1940212	Grishika Sharma	Btech CSE
AU1940268	Dhruv Bhatt	BTech CSE
AU1940183	Mithil Parmar	Btech CSE
AU1940172	Aayushi Chauhan	Btech CSE
AU1940272	Dhruvi Desai	Btech CSE

INTRODUCTION

Project Title: Multicasting Multimedia over IP

Language of Implementation-Python
Imported Libraries-

- Imutils
- Queue
- Threading
- Socket
- Numpy
- Pickle
- Wave
- Os
- Time
- Base64
- Struct
- openCV
- ThreadPoolExecutor (from concurrent.features)

Transferring of MP4 video stream files is done from Server to Client. The client connects to the server via a TCP socket and receives multimedia streams via a UDP socket. We have used the Python language. The server listens on port 9688 for requests of the stream file. The outgoing data rate should be close to the stream bit rate so we are using Imutils package to limit the frame size to 65536 bytes so buffer overflow can be avoided in the server. Also, a point to take into consideration is that we have converted the stream file to a format which can be played using ffmpeg -i

---

FEATURES & DETAILS

Server

We have created a Datagram based server that uses the **IPv4** address scheme. Here we have used a **Queue** data structure to store frames and transfers. Here the queue stores a certain number of frames when we run the server code and one frame is taken out and sent to the client and a new frame is added to the queue. As it is, addition and removal of frames happen continuously.

After we run the client code the video plays in a limited maximum frame size of **65536 bytes** which is also the maximum transferable buffer size. Here we have used SO_RCVBUF to set the receive buffer size to BUFF_SIZE in the setsockopt() function.

Threading is used to perform parallel tasks of sending stream files continuously as well as to detect connection requests from clients.

We then did basic socket binding code where we defined socket family, hostname and a socket address. The socket is listening at **port 9688** and **IP 127.0.0.1.**

The creation of a video capture object is done using cv2.VideoCapture which is a function of the **CV package**. This VideoCapturefuction returns a tuple which contains frames. To open the video file we have used the isOpened() function. If the video file cannot be opened then an Error message is displayed. we use the get() method to retrieve important metadata associated with the video stream.CAP_PROP_FPS retrieves frame rate.CAP_PROP_FRAME_COUNT counts the number of frames in the video file. **total time taken by the frames is given by the total no. of frames/frames per second.**

Using vid.read() **tuples** are returned. We resize the frame using **imutils**. We enqueue and dequeue frames as they are added or sent to another socket respectively.

To stream the frames of video we have created a window named **TRANSMITTING WINDOW**. **recvfrom()** method reads the number of bytes sent from a UDP socket. This method returns a byte read from a UDP socket and the address of the client socket as a tuple.

Items are managed in the queue using the **get()** method. The encode() method encodes the image into streamable data. Here the extension of the image is encoded to **.jpeg**. The JPEG image quality can be controlled with cv2.IMWRITE_JPEG_QUALITY parameter and its value (between 0 & 100) default value of the parameter is 95 but we have set it to **80.**

To keep the inherent frame rate of video while rendering, we stabilize the rendering **sample time (TS)** in the loop. by current frame rate (ie. fps) is higher or lower than the desired inherent frame rate(ie. FPS) and according to increment or decrement TS.

Client

In the first part of the program, we created a datagram socket. The setsockopt() function provides an application program with the means to control socket behaviour. We tried to lower the receiver's socket buffer size. We enter the hostname and assign the port number on which we have to connect. Then, we send the message to the server using the created UDP Socket. We constructed the function of video_stream for video streaming. In the function, the window name is 'RECEIVING VIDEO,' where the frame will be displayed. In the function, we implement a while loop with different parameters. In the functions, we created different codes for receiving a reply from the server, returning the decoded packet, constructing an array from the byte using NumPy, decoding the array, drawing a text string on any image, using the cv. imshow() method for displaying the frame in a window. cv2.waitkey() function binds the code with the keyboard, and anything we type will be by this function. It returns a 32-bit integer value. The critical input is in ASCII, which is an 8-bit integer value. So using bitwise AND, it can return the binary form of the input key. Then we closed all the windows.

We declared the number of frames in the buffer and the number of samples collected per sample. We were creating a client socket that would accept the connections. Then, we run a loop to send and receive data from the server simultaneously.

In the end, we obtained a video stream.

Some Screenshots:

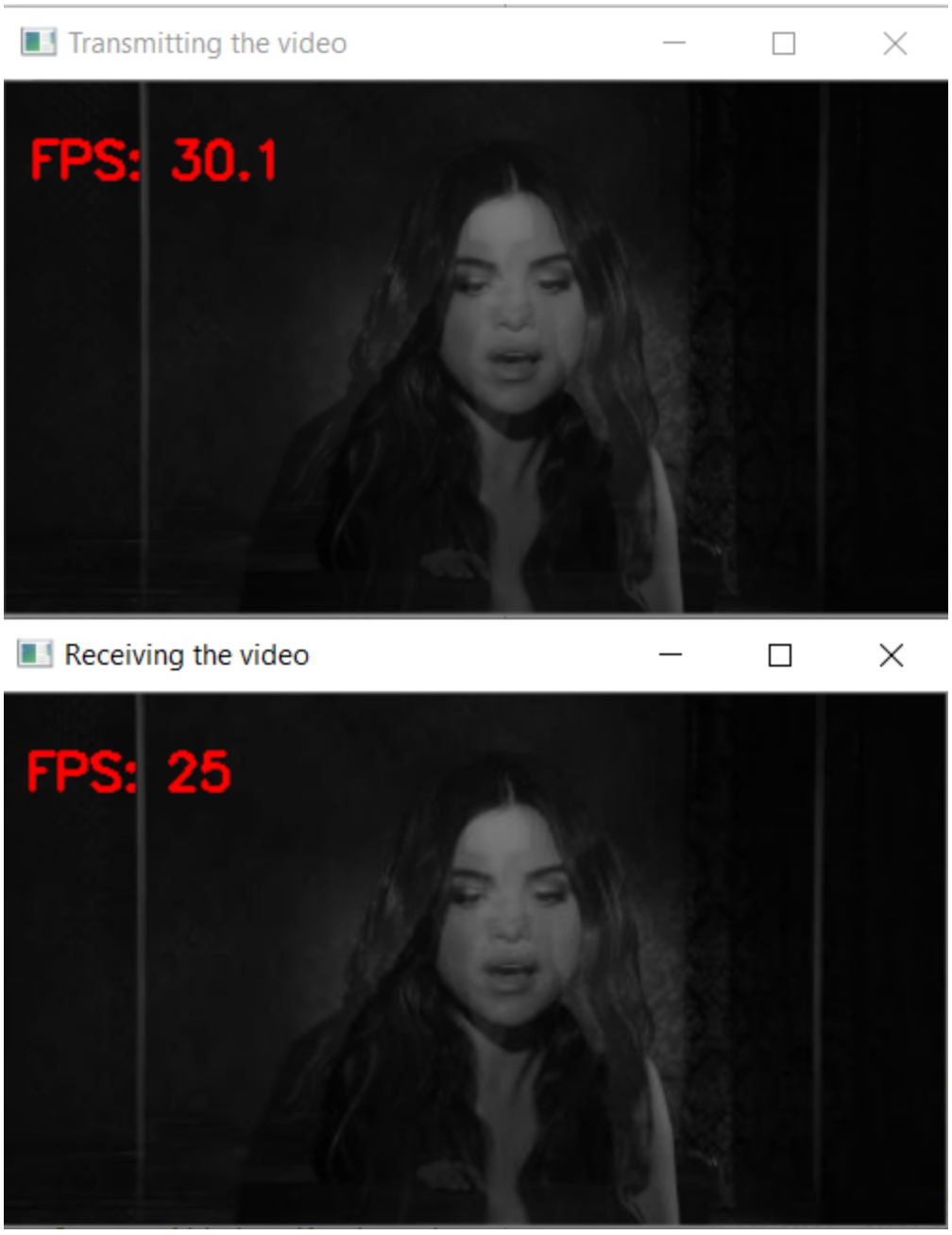
SERVER

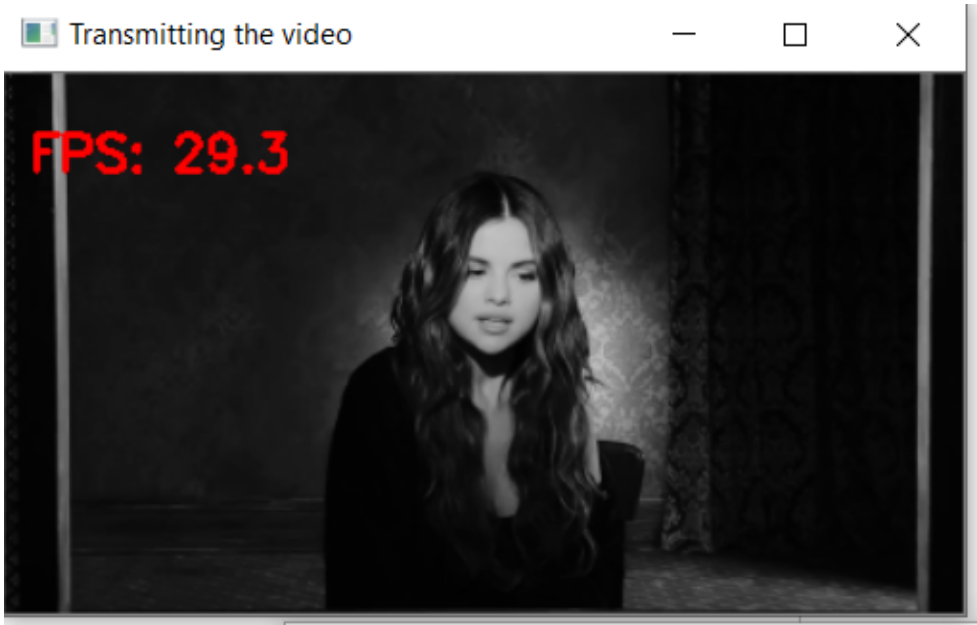
```
Python 3.7.4 (default, Aug  9 2019, 18:34:13) [MSC v.1915 64 bit
(AMD64)]
Type "copyright", "credits" or "license" for more information.

IPython 7.8.0 -- An enhanced Interactive Python.

In [1]: runfile('C:/Users/Dhruvi/Downloads/fsecondserver.py',
wdir='C:/Users/Dhruvi/Downloads')
127.0.0.1
Address where our socket server is Listening is: ('127.0.0.1',
9688)
The FPS is: 23.976023976023978 0.020854166666666667
206.91504166666664 0.0
We got the connection from :-  ('127.0.0.1', 54645)
```

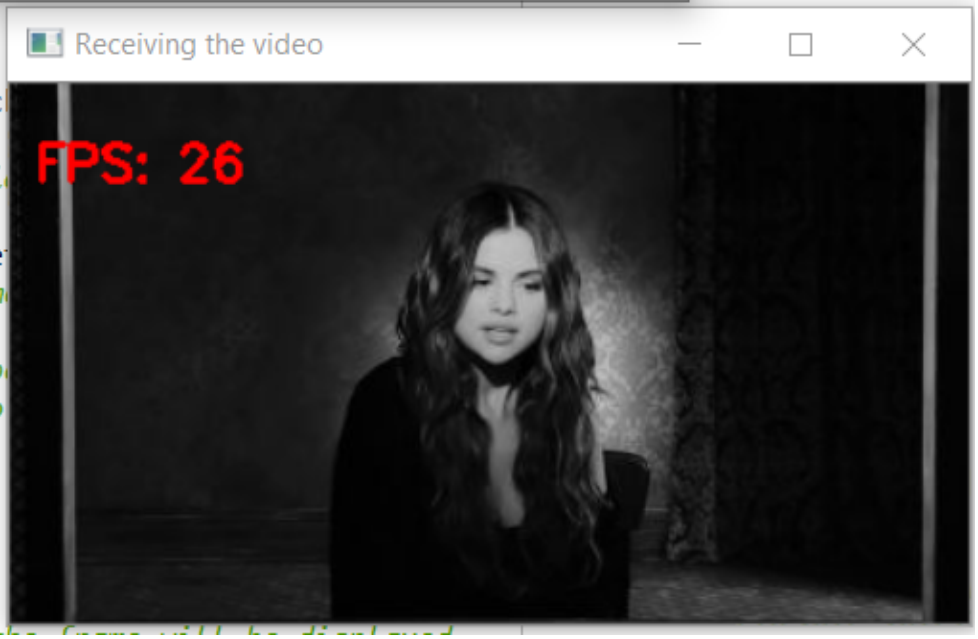
CLIENT

```
Python 3.7.4 (default, Aug  9 2019, 18:34:13) [MSC v.1915 64 bit
(AMD64)]
Type "copyright", "credits" or "license" for more information.

IPython 7.8.0 -- An enhanced Interactive Python.

In [1]: runfile('C:/Users/Dhruvi/Documents/secondclient.py',
wdir='C:/Users/Dhruvi/Documents')
127.0.0.1
server listening at ('127.0.0.1', 9687)
```

OUTPUT


Transmitting the video
FPS: 30.1
Receiving the video
FPS: 25






```
ket.AF_INET, soc

des an applicati
ket buffer size
OL_SOCKET, socke
ket.gethostbynam

igning port numb
d message 'Hello

P socket
st_ip, port))
```