# INTERSHIP REPORT

CNI

| | |
|---|---|
| SUBMITTED BY: | Krishna Prabhakar |
| DURATION: | 17 MAY 2021 - 2 AUG 2021 |

# 1. INTRODUCTION

**1.1 Objective:** Devise an algorithm to effectively adapt to varying network capacity while insuring real-time video data transmission.

## 1.2 Setup:

1. There will be a camera on the car to capture the road-scene and send it to encoder.
2. There will be a H.264 encoder on the car, which will encode the raw frames received from the camera. After Encoding the video packets will be send to control room. Wi-fi will be used as the data-link layer.
3. There will be a control room from which we will drive the vehicle.
4. There will be a H.264 decoder machine in the control room for receiving the video data and then decoding it.
5. There will be a screen in the control room on which we will see the scenes sent by the car machine.

6. For encoding and decoding we will be using x264 library which is an implementation of H.264 codec. For Transmission we will be using UDP as Transport layer protocol because we want real-time streaming of scenes.

## 1.3 Definitions:

**FFMPEG:** FFmpeg is a multimedia framework written in C and Assembly language. It is a cross-platform software that can run on Linux, Mac, Windows, Solaris, etc. It supports wide range of media codecs, but for our purpose we will use H.264 video-codec.

**H.264:** It is a video-codec standard. This standard specifies a set of rules to compress the raw video-frame and another set of rules to decompress the compressed video frames. There can be two types of compressed frame I-frame or P-frame.

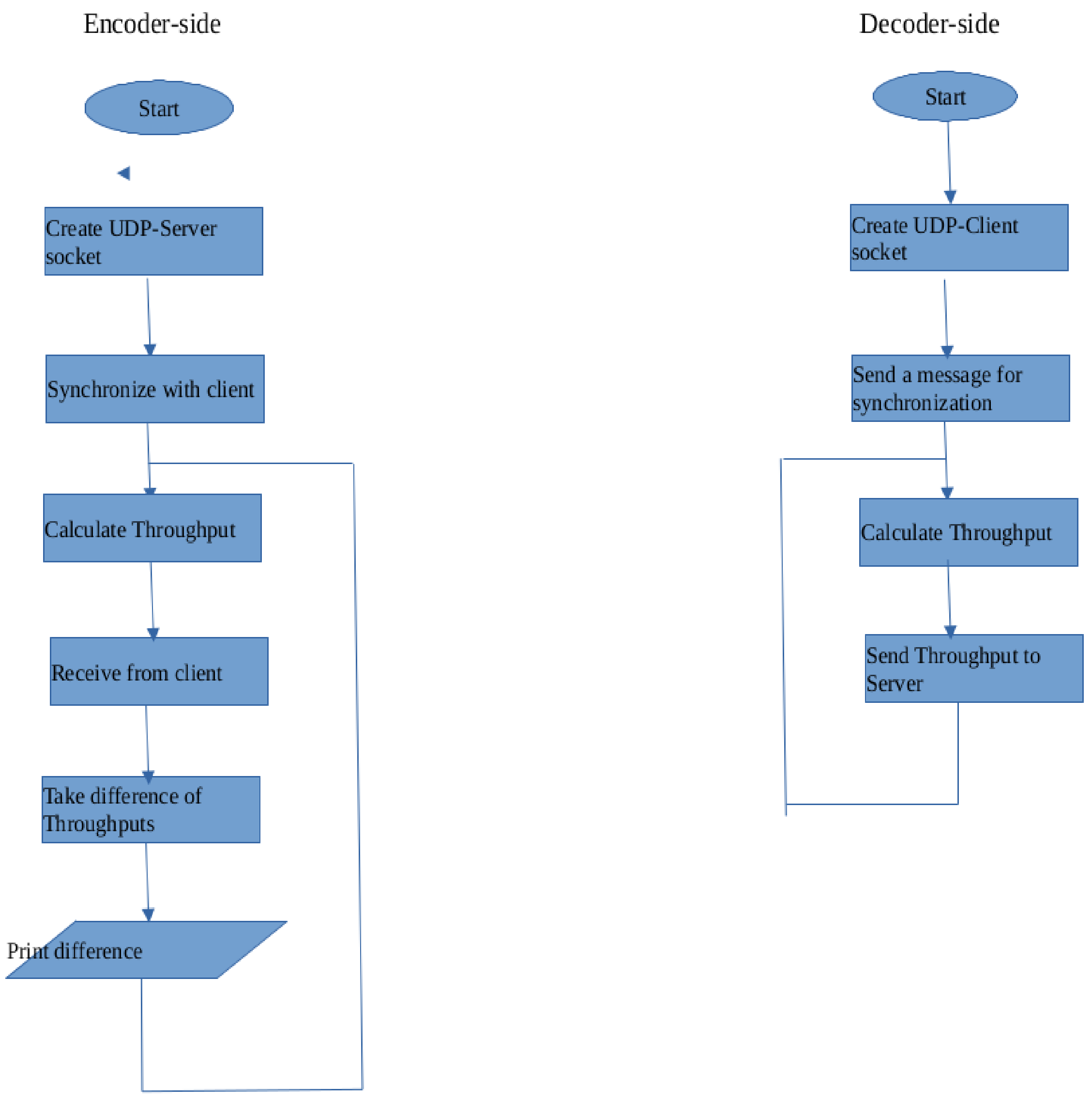
**X264:** It is an implementation of H.264 standards. Though it is not perfect but it is an optimized implementation of H.264 and can be used in real life applications.

# 2. MY CONTRIBUTIONS

## 2.1 Program for comparing Decoder-side throughput and Encoder-side Throughput in real-time:

The purpose of this programs was to see if the encoder-side and the decoder-side throughput conforms with each other.

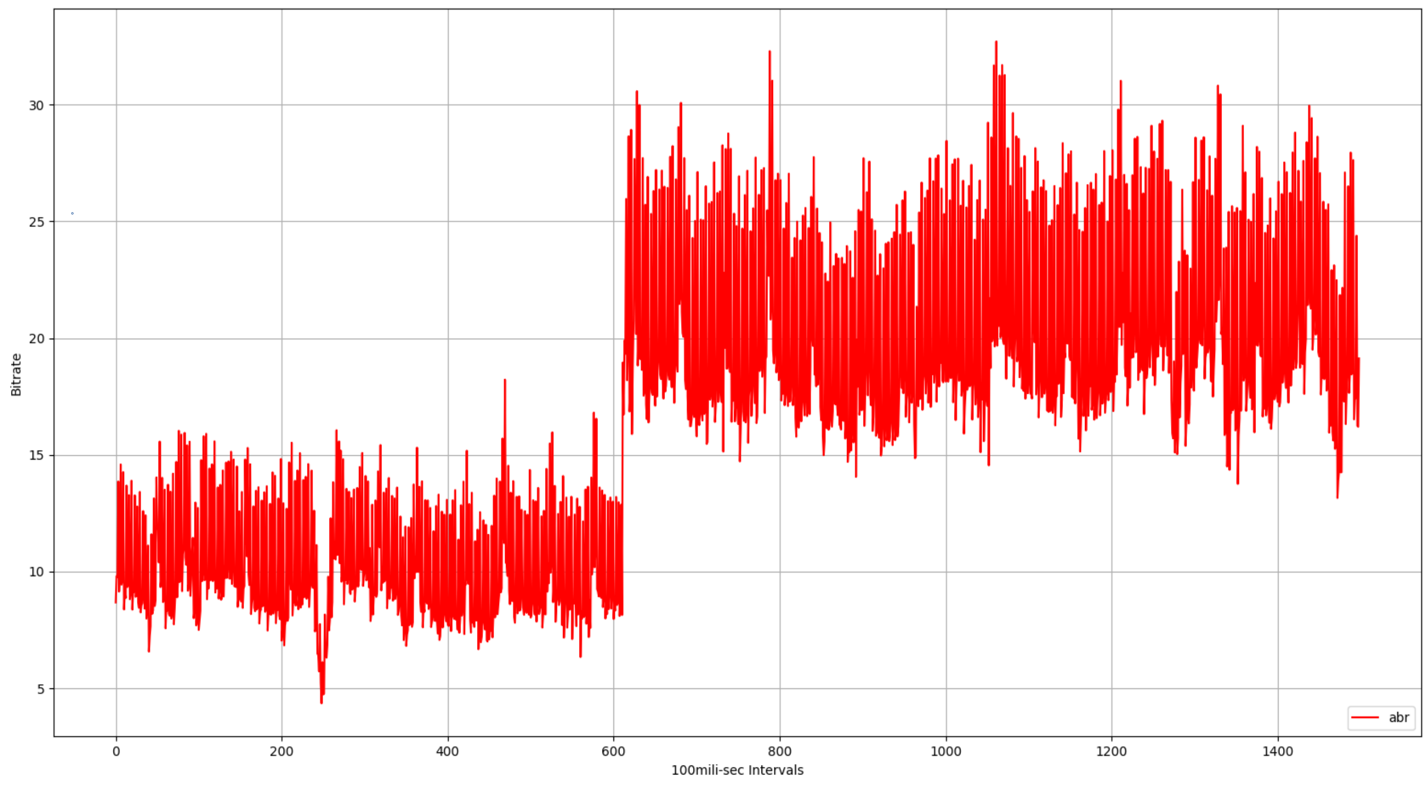
This program has two parts, one at the Encoder-side machine and another at the Decoder-side machine. It computes throughput for each 50 milli-second intervals. Both parts remain synchronized with each-other. The Decoder side part sends its throughput to Encoder side part using a UDP socket and then Encoder side part compares it with its own throughput and prints it in stdout.



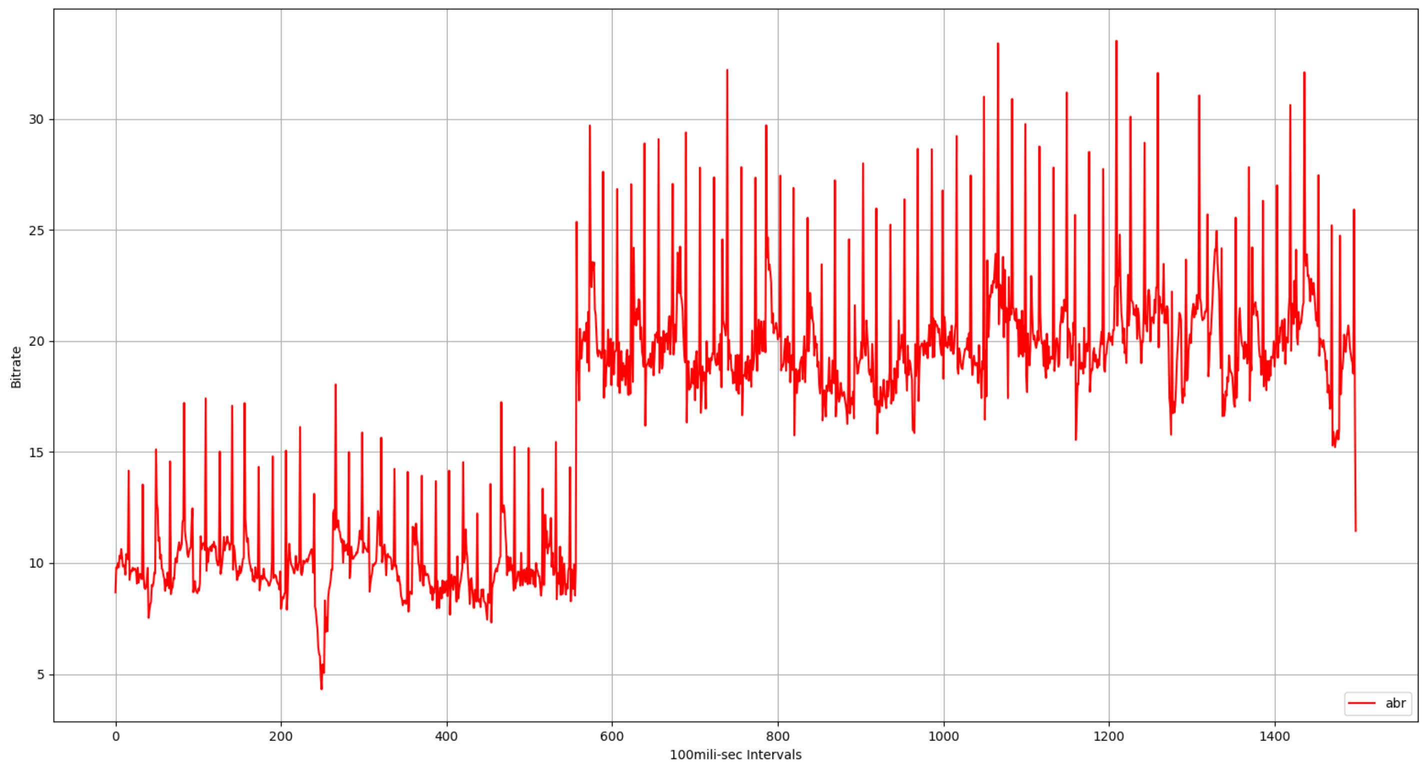
*Flow diagram*

## 2.2 Experiment with varying GOP size in FFmpeg:

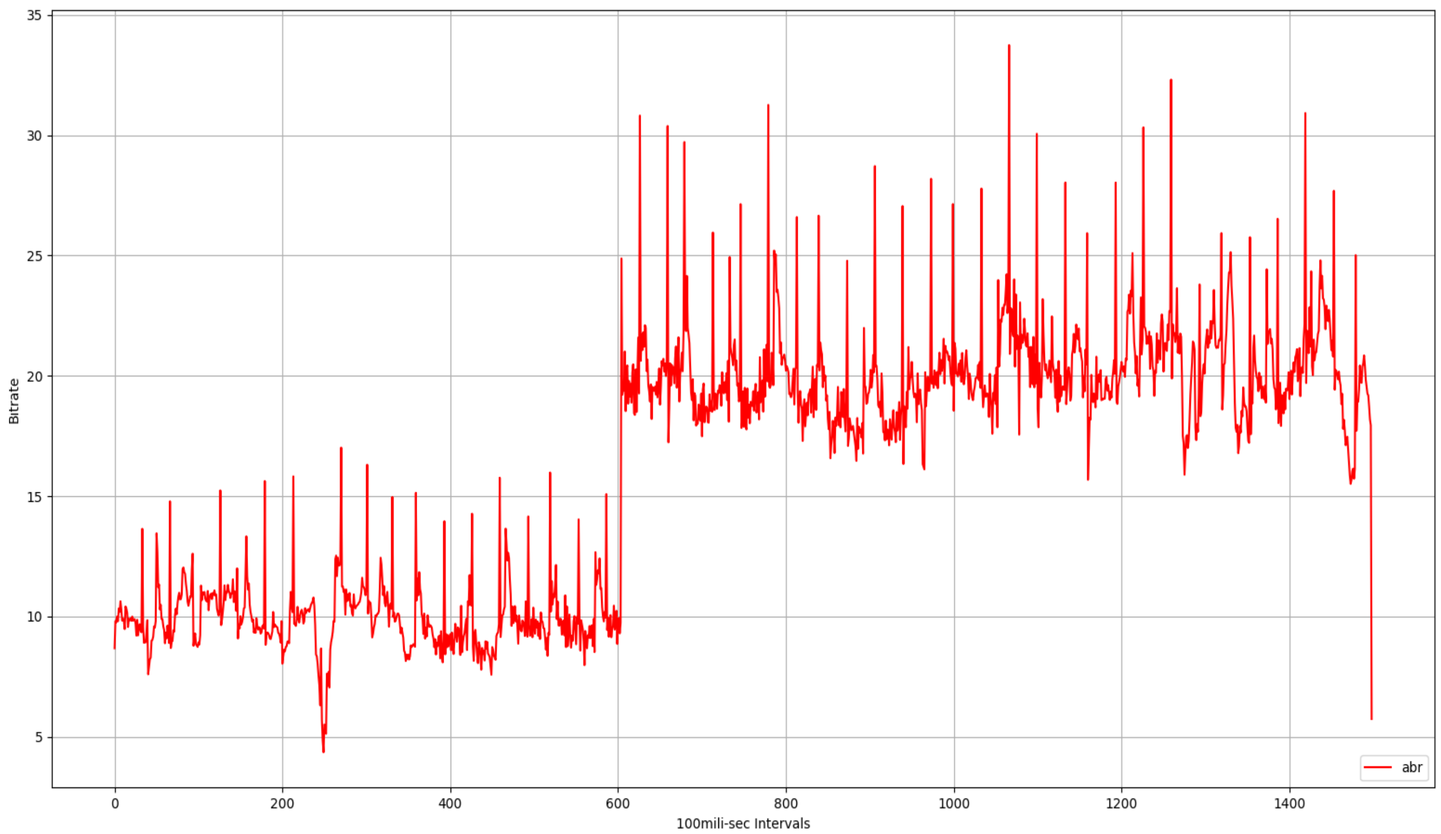
The Motive of this experiment was to observe the effect of GOP size on the throughput.
In our experiment we found that increase in GOP size corresponds to smoother variation of throughput. Also, it does not affects quantization factor significantly.



***GOP=10***



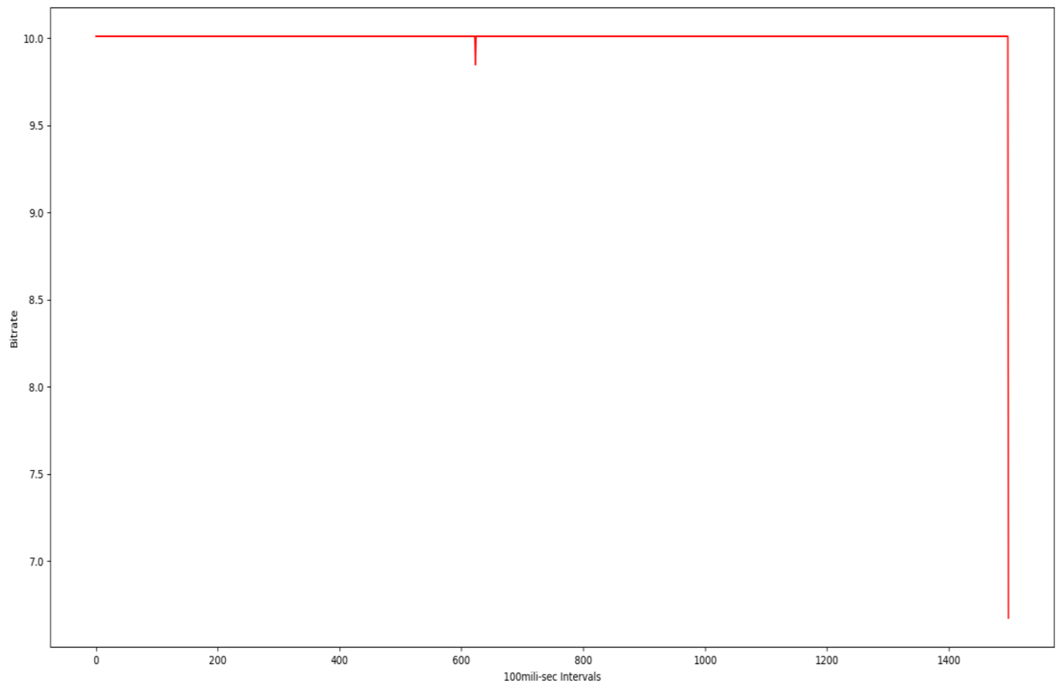
***GOP = 50***



***GOP=100***

## 2.3 Experiments with CBR setting in FFmpeg:

The motive of this experiment was to verify if there is any more restrictive rate–control mode than ABR rate-control in FFmpeg .
There is no CBR rate-control mode in FFmpeg, but we can simulate CBR using "nal-hrd=cbr:force-cfr=1" option while running the ffmpeg. Doing this can give us a graph in which there is insignificant variation in throughput. But what it does is just putting filler data in simple scene frames and applying more compression on complex scene frames to give the constant bitrate.
So, this is not efficient and does not match our requirement.

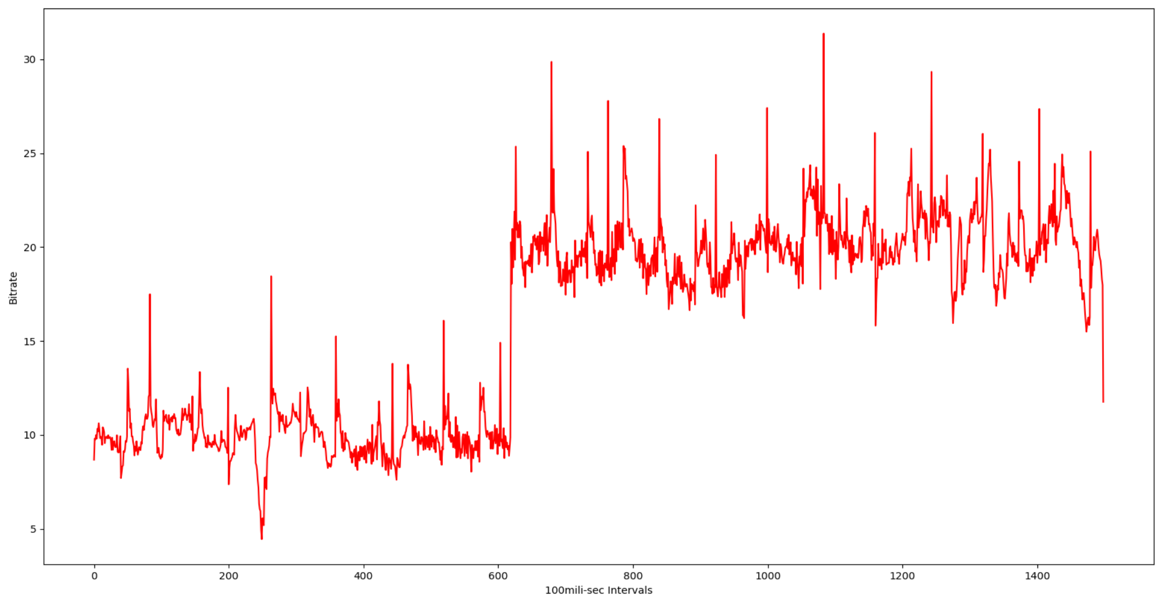


***Plot with "nal-hrd=cbr:force-cfr=1"***

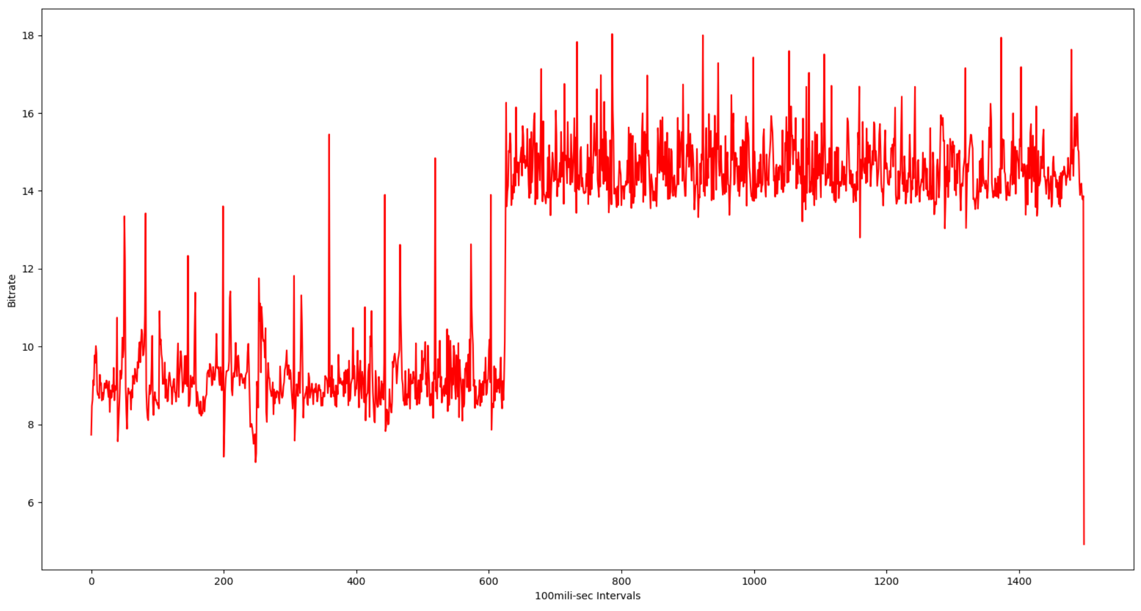
## 2.4 Experiments with buffer size:

The motive of these experiments was to find out if we can use the buffer option provided by the ffmpeg to improve our traffic shaping.
The buffer option should be used in conjunction to the maxrate option. The bufsize option uses a leaky-bucket model to do traffic shaping.

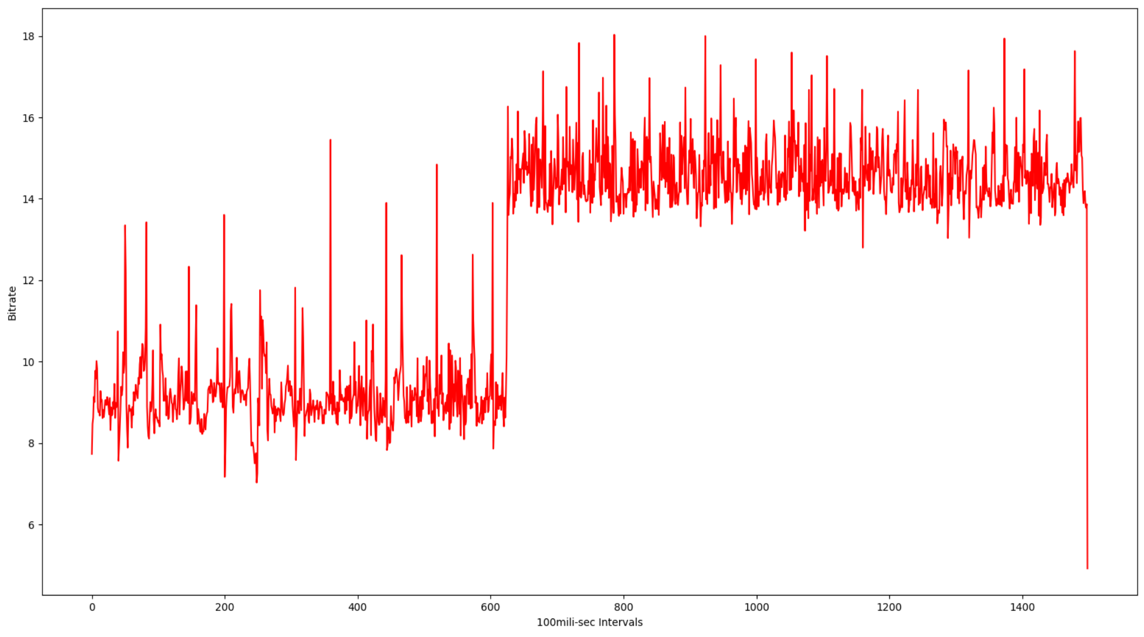
During this experiment we found that we can use the bufsize option to improve output traffic. Smaller buffer size gives less variation in throughput. We also experimented with different buffer size to get the optimum value, but it was still not clear how to get the optimum size of buffer.



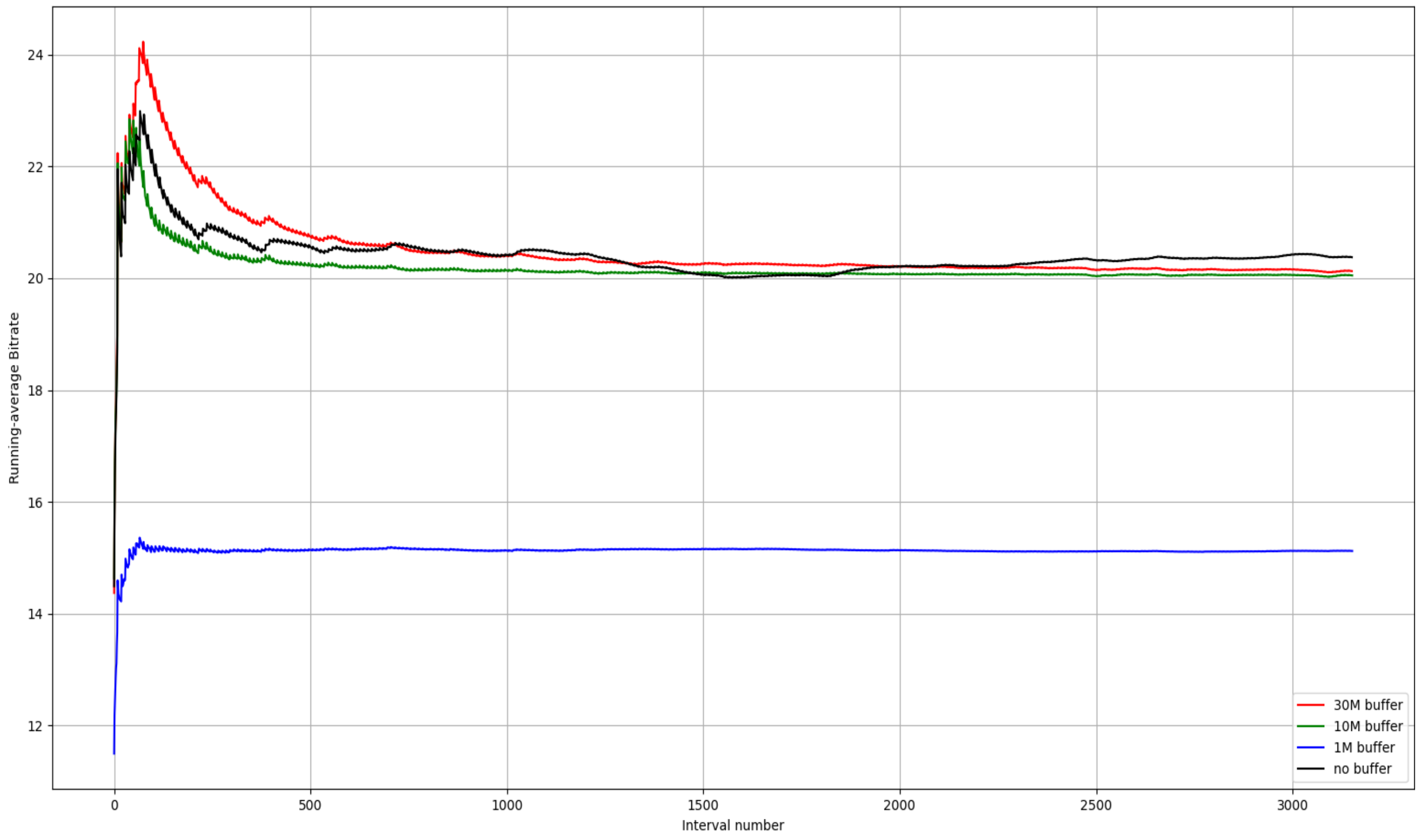
***No buffer (30fps video)***



***667Kb buffer (30fps video)***



***1M buffer(30fps video)***

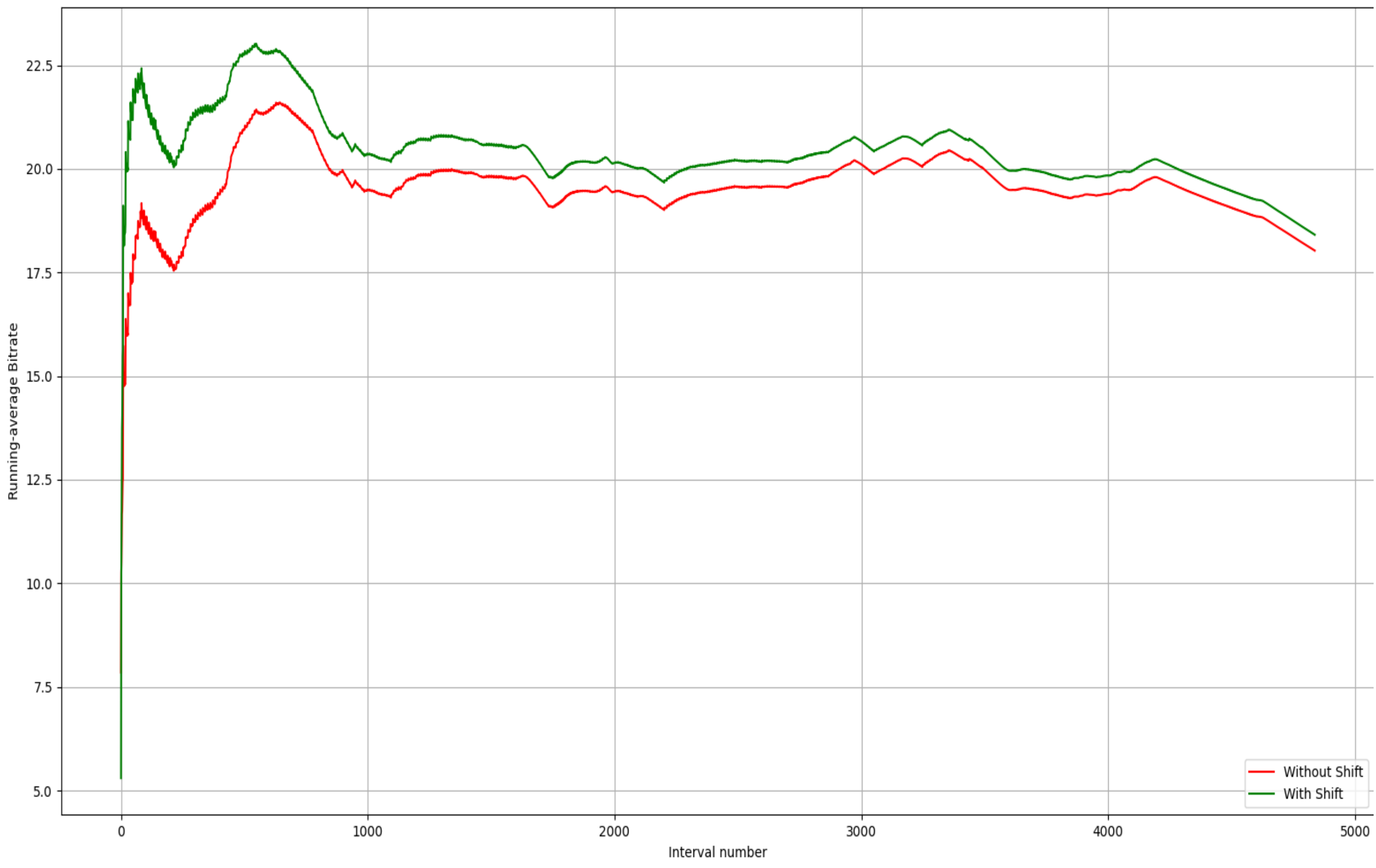


***Running average of bitrate with different buffer size (30fps)***

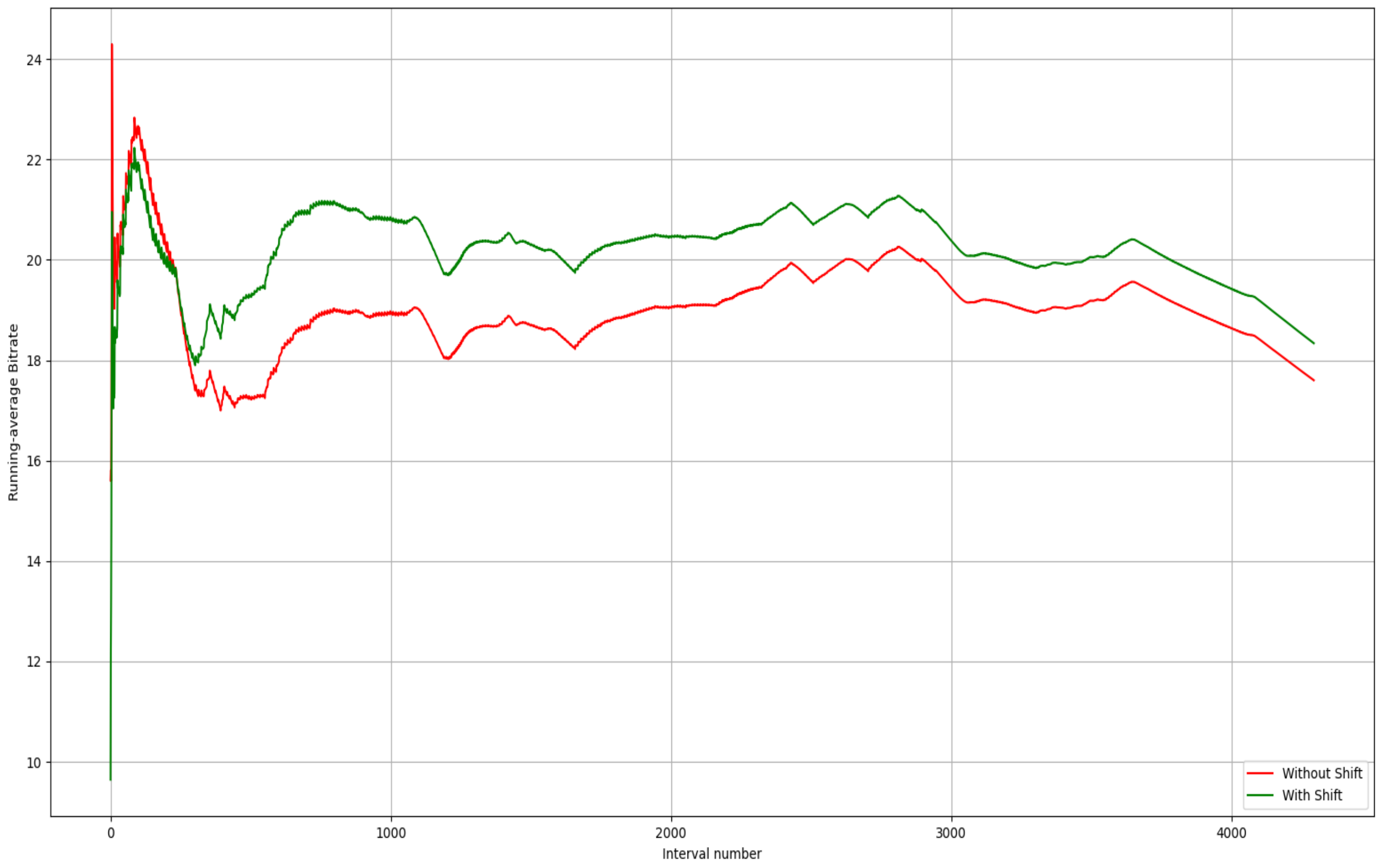
## 2.5 Running time average-throughput curve:

The motive of this experiment was to observe the effect of switching on the video traffic. In our experiment we tried to switch the target bitrate from 10Mbps to 20Mbps and compare it with a normal 20Mbps. We also experimented with switching after different duration and plotted them to see if different frames have different effect during switching. We found that the frame does not affect the curve. And both the normal and the switched case curve approaches to a constant difference.

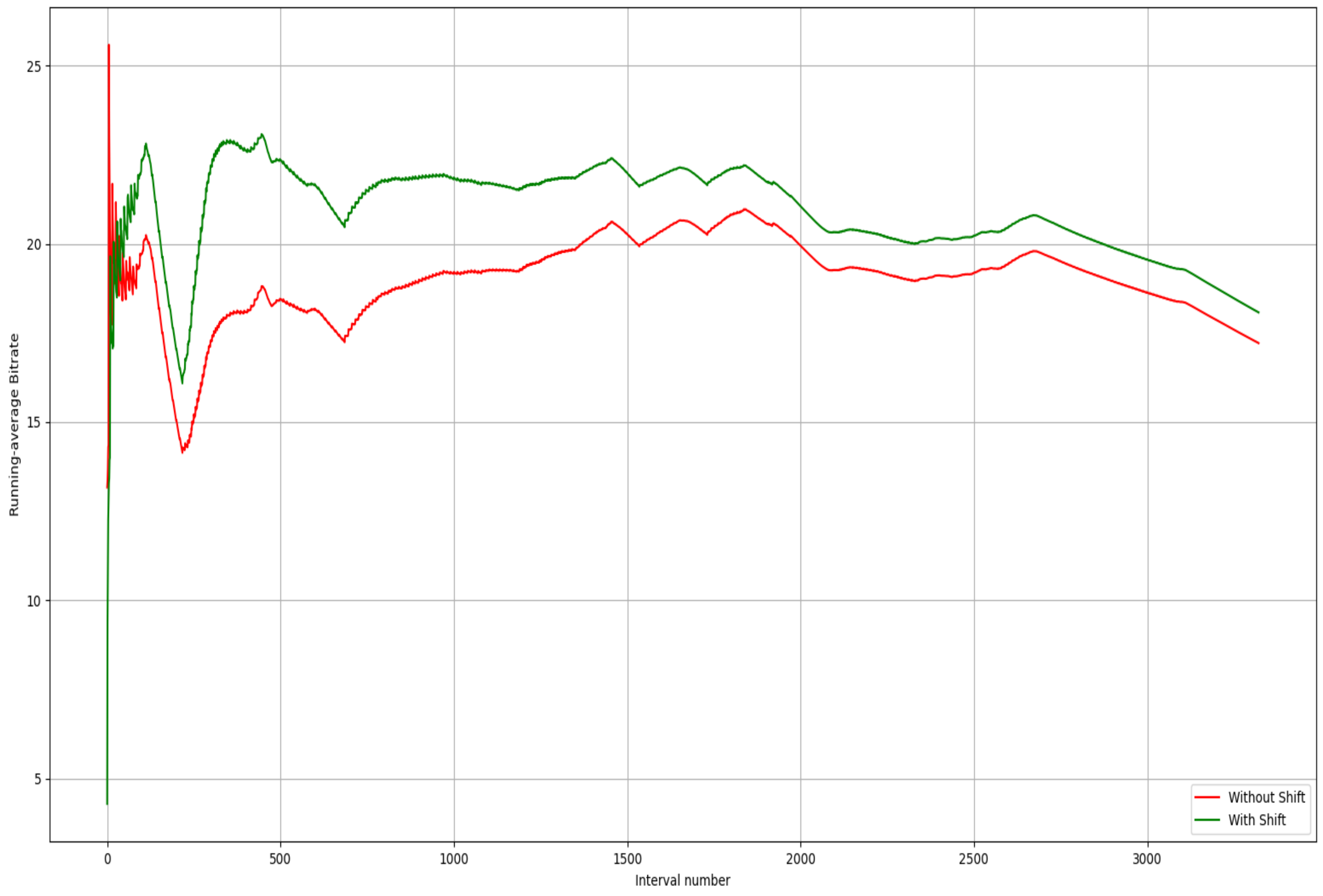
In the graph red curve is the switch case and green curve is the normal curve



***Switching at 7th second (60fps video)***



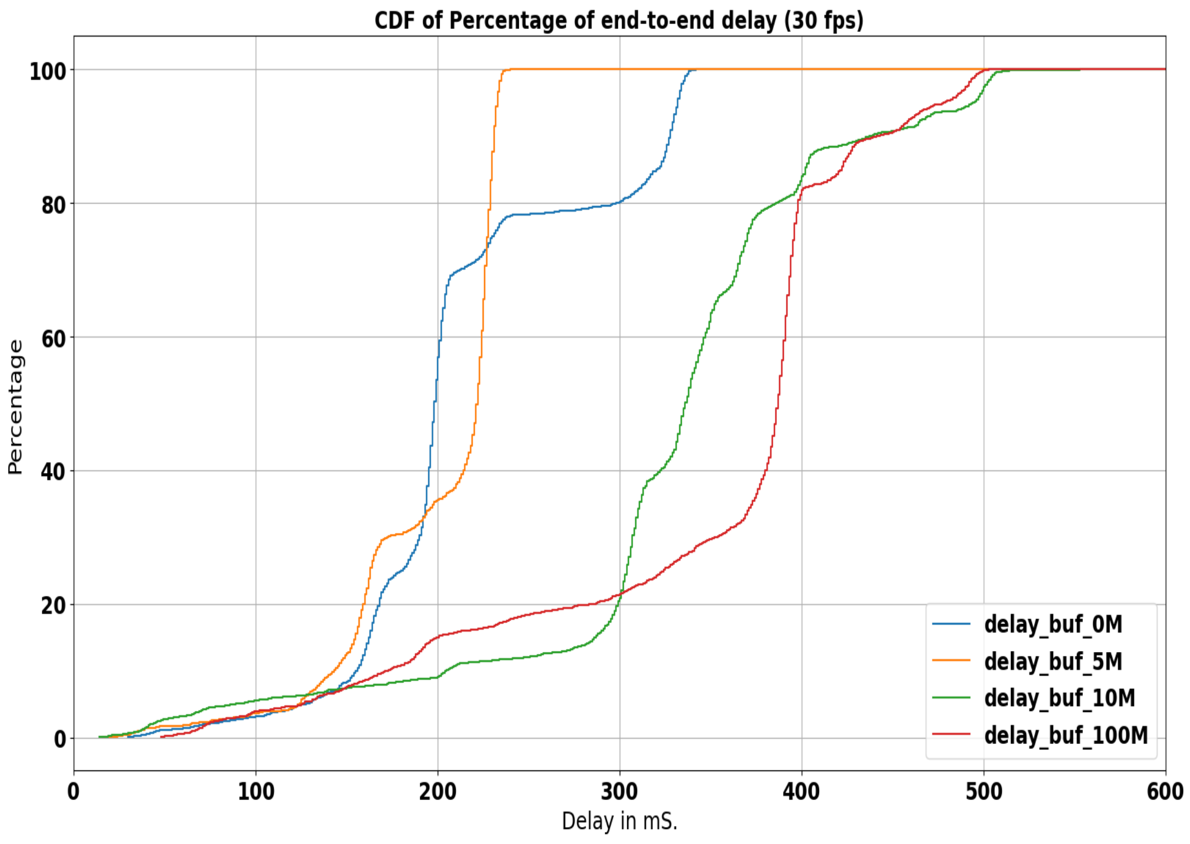
***Switched at 15th second (60fps video)***



***Switched at 32nd second (60 fps)***

## 2.6 End to end delay experiment:

The motive behind this experiment was to observe the effect of buffer size on the end-to-end delay between ffmpeg and ffplay machine.
In this experiment I used NTP to synchronize both the machine. But I was using Mobile hotspot to connect the two machines so the synchronization was not good.
The experiment shows that increasing buffer size increases delay but not using bufsize option (Blue curve) also gives more delay than small buffer.


CDF of Percentage of end-to-end delay (30 fps)
Percentage
0
20
40
60
80
100
0
100
200
300
400
500
600
Delay in mS.
delay_buf_0M
delay_buf_5M
delay_buf_10M
delay_buf_100M

## 3. CONCLUSIONS

**After all these experiments following things were concluded-**

1. **We can use the bufsize option i.e., HRD model to improve the streaming experience**
2. **If we do not use the bufsize option then also our original rate control algorithm will do fine job.**
3. **After implementing the rate control algorithm, we may consider bufsize option to improve our algorithm.**

## 4. THINGS I LEARNED IN THIS INTERNSHIP

During this internship I learned a lot about how to apply my skillset to solve a real-life problem. The team in which I was working there were experienced mentors who guided me throughout the internship. I learned how to work in a team with a cool behaviour. It gave me confidence that I can work in a team on any real-life project which requires my skillset.

Apart from the personality development, I also learned some technical information and skills. Some of them are-

**4.1 Linux:** Linux as we all know is an operating system kernel. We were using Ubuntu operating system which is a Linux based OS. I learned using some command like **grep** for finding matching patterns in file(s), **find** to find a file with matching a regular expression, **sort** to sort the content in a file, **uniq** to remove redundant lines from a file, **mac2unix** to convert the mac style files (log files of ffmpeg) to Unix style file, etc. I also used some **awk** scripts to process the file content. And some advanced things like setting up **NTP** server and client to synchronize with each other. It was a good start with Linux for me and I will continue to learn more things in Linux.

**4.2 H.264:** H.264 is a video codec standard to compress the raw video frame, because it is not practical to store and transfer raw video frames. It will take huge amount of resources like memory and throughput. H.264 is also called as Advanced Video Coding. I learned how H.264 compress raw video frames and some advanced concept like HRD model. Though I did not get the whole picture but I understood some basic things like quantization, DCT transformation, inter-frame and intra-frame prediction, etc.

**4.3 FFMPEG:** FFmpeg is a multimedia framework there are many libraries and tools in this framework. I mainly worked with ffmpeg and ffplay tools of this framework. ffmpeg was used to take a raw video and compress it using H.264 encoder then send it to other side using UDP and Wi-Fi protocol. ffplay was used to receive the compressed video packets and then decompress it to show it on screen.

**4.4 IEEE 802.11:** During the starting of the internship, I was given books and materials to learn some basic facts about 802.11 protocol which is used to implement Wireless LAN. From that book I learned about the different components used in the WLAN like Stations which are end devices, Access Point which communicates directly with end devices, etc. I also learned about the different topology of WLAN like IBSS in which there is no central device like Access Point to connect them, BSS in which there is Access Point and stations, ESS in which there are multiple BSS connect with each other via switches. stations, ESS in which there are multiple BSS connect with each other via switches. I also learned about RTS/CTS handshake for collision avoidance and NAV which is used to reserve the medium by a station.

*******