**INTERSHIP REPORT**

**LOW LATENCY VIDEO TRANSMISSION OVER VARIABLE CHANNEL CAPACITY**

**SUBMITTED BY:**  **Krishna Prabhakar**

**DURATION:** **17 MAY 2021 - 02 AUG 2021**

**1.** **INTRODUCTION**

* 1. **Objective:**

The goal of this project is to develop a system using which we can drive a vehicle from control room. To achieve this goal, we need hardware devices like camera, controller, processor, etc and we also need software parts like encoder, decoder, streamer, traffic shaper, etc.

We need real-time video frames captured by camera on the vehicle in control room. For encoding, decoding and streaming we are using **FFmpeg** utilities. We are using Wi-Fi network to stream the video frames. But we also need to account for varying throughput availability, because vehicle will move so we cannot ensure constant throughput availability in Wi-Fi network. So, we need to devise an algorithm to take care of varying throughput.

I mainly contributed in understanding rate-control methods used by ffmpeg utility of FFmpeg framework.

* 1. **Setup:**

1. There will be a camera on the car to capture the video of front side.
2. Raw frames will be encoded using H.264 encoder. After encoding, the video packets will be sent to control-room using Wi-Fi network.
3. There will be a control-room from where we will control the vehicle.
4. After video data reaches at control-room, it gets decoded by H.264 decoder.
5. For encoding and decoding we will be using X264 library which is an implementation of H.264 codec.
6. For Transmission we will be using UDP as Transport layer protocol because we want real-time streaming of video-frames.

Figure 1 shows the block diagram.

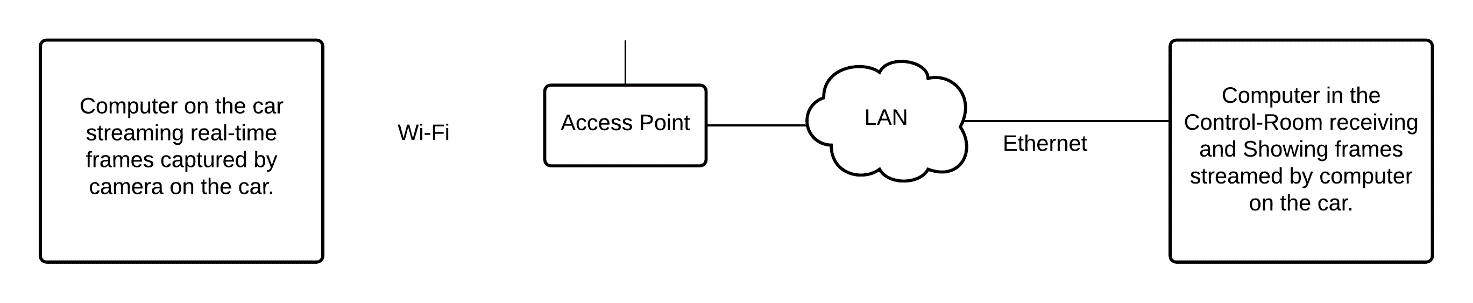


Figure : Block-diagram for the setup

* 1. **Definitions:**

In this project we are using ffmpeg for encoding and streaming the live captured video by camera and ffplay to decode and play the streamed video in the control-room. For encoding-decoding we are using H.264 codec.

**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**” [1].** There are various utilities in this

framework like ffmpeg for capturing live video/audio and converting it into another format, ffplay for decoding and playing video/audio, ffprobe for analysing video/audio streams.

**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 frames I-frames which are larger in

size and P-frames which are relatively smaller in size because of intra-frame

prediction and inter-frame prediction.**” [2]**

**X264:** It is an implementation of H.264 standards. Though it is not full but it is an

optimized implementation of H.264 and can be used in real life applications.

In our project we are using X264 library to compress the video captured by the

camera, before streaming it and decompress the compressed video data received at

control-room.

**2. MY CONTRIBUTIONS**

**2.1 Tool for comparing** **Throughputs in real-time:**

The purpose of this tool is to verify if the encoder-side and the decoder-side throughput are equal.

This tool has two parts, one at the video-sender machine and another at the video-receiver machine. It computes throughput for each 50 milli-second intervals. For computing throughput –

1. It reads content of “/sys/class/net/wlo1/statistics/rx\_bytes” file. This file contains

number of bytes received via that network interface. It reads that file at t = 0ms

and t = 50ms.

1. Then it subtracts the earlier reading from later reading.
2. After that we multiply the subtraction result by 20 to get throughput in Bytes per second because, each interval is of 50ms and in 1 second there is 20 such interval.

We use Network Time Protocol (NTP) to ensure that clock of both machines remain synchronized with each-other.

When the video-receiver side part starts, it sends a starting time (current\_time + 50ms) to video-sender side machine for synchronization. After getting that time video-sender side part knows when to start.

The video-receiver side part sends its throughput to video-sender side part using a UDP socket and then video-sender side part compares it with its own throughput and prints it in stdout.

We did not manage to do the experiment using this tool because of change in plan. Figure 2 shows the high-level flow diagram for this tool.

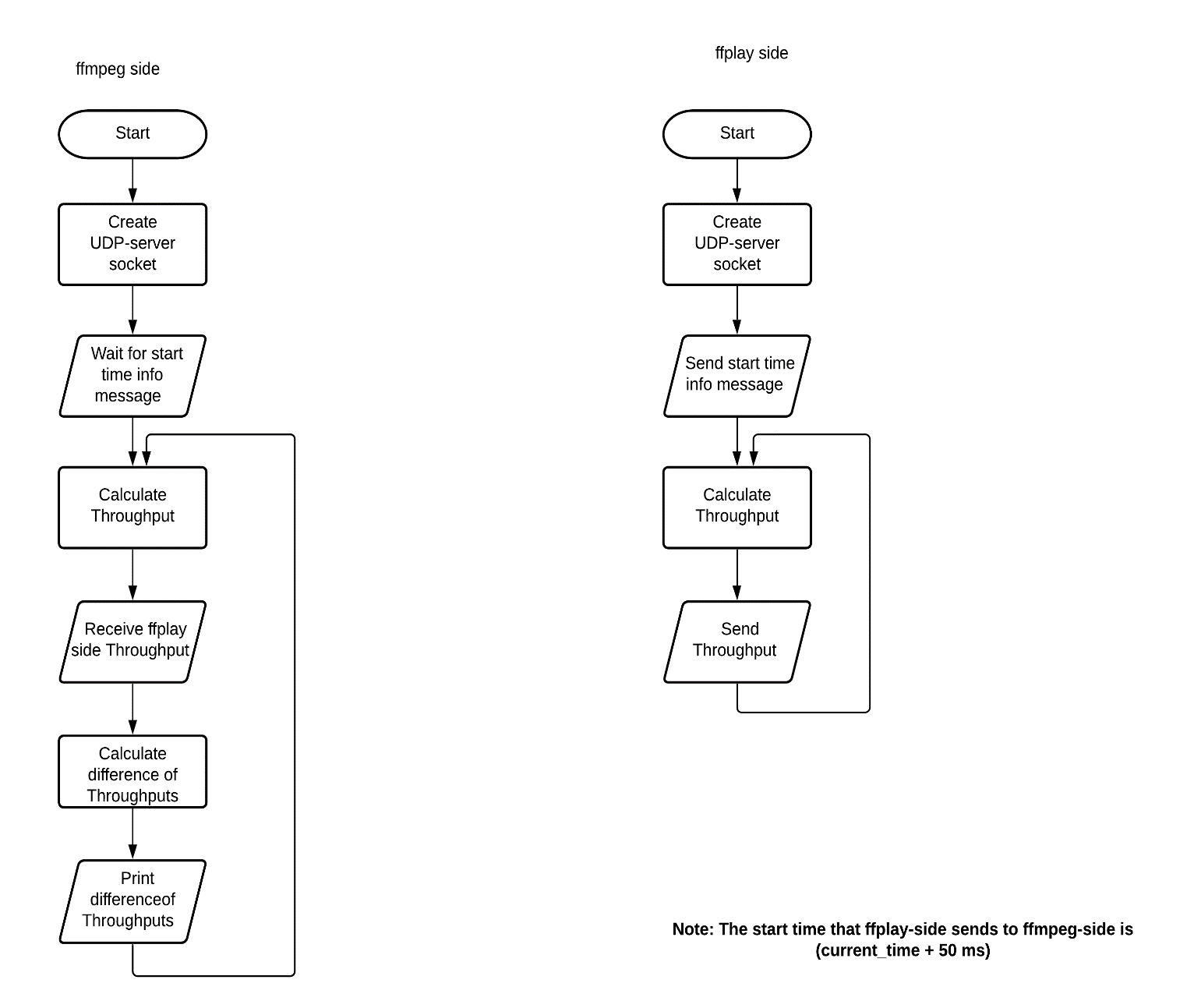


Figure 2: Flow-diagram for Throughput programs.

**2.2 Experiment with Group of Pictures (GOP) size in ffmpeg:**

**Motive:**

The Motive of this experiment was to observe the effect of GOP size on the

output bitrate of ffmpeg.

**Description:**

GOP size is the maximum number of frames that can be dependent on

each-other for decoding. First frame of a GOP is an I-frame and all the subsequent frames are P-frames. Size of a I-frame is relatively larger than the size of a P-frame, because I-frames contains all the data needed to decode it whereas a P-frame depends

on the other frames so that it can be decoded.

**Procedure:**

1. Run the ffmpeg and with GOP size set as 10.
2. Save the log file to extracting and plotting graph.
3. Repeat step 1 and step 2 for GOP=50 and GOP=100 respectively.
4. Extract and then plot the graph.

**Outcome:**

In our experiment we found that increase in GOP size corresponds to smoother

variation of throughput. Also, it does not affect quantization factor significantly.

Figure 3 shows bitrate plot for GOP = 10, Figure 4 shows bitrate plot for GOP =

50 and Figure 5 shows bitrate plot for GOP = 100.

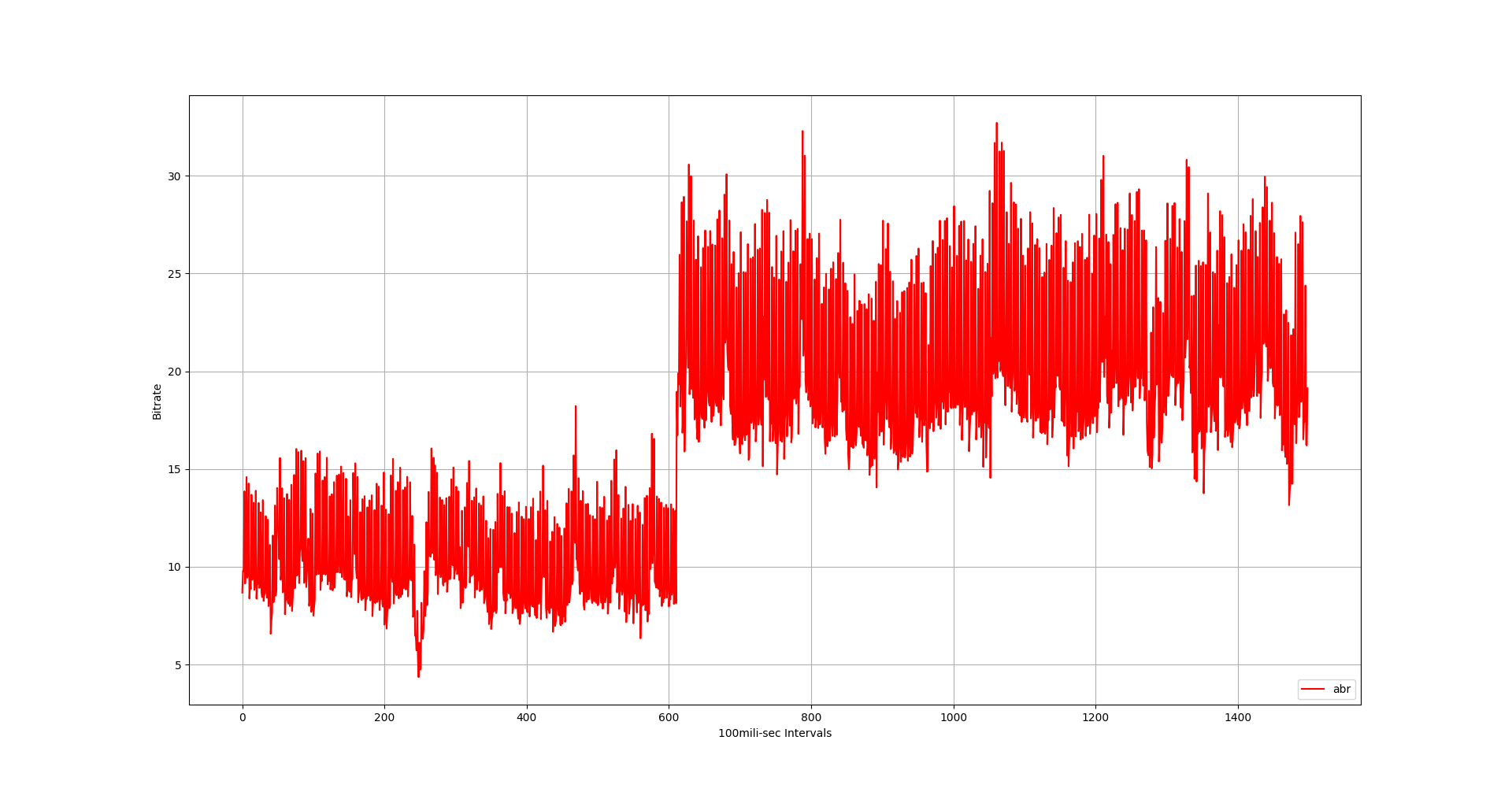


Figure 3: Bitrate plot for GOP = 10.

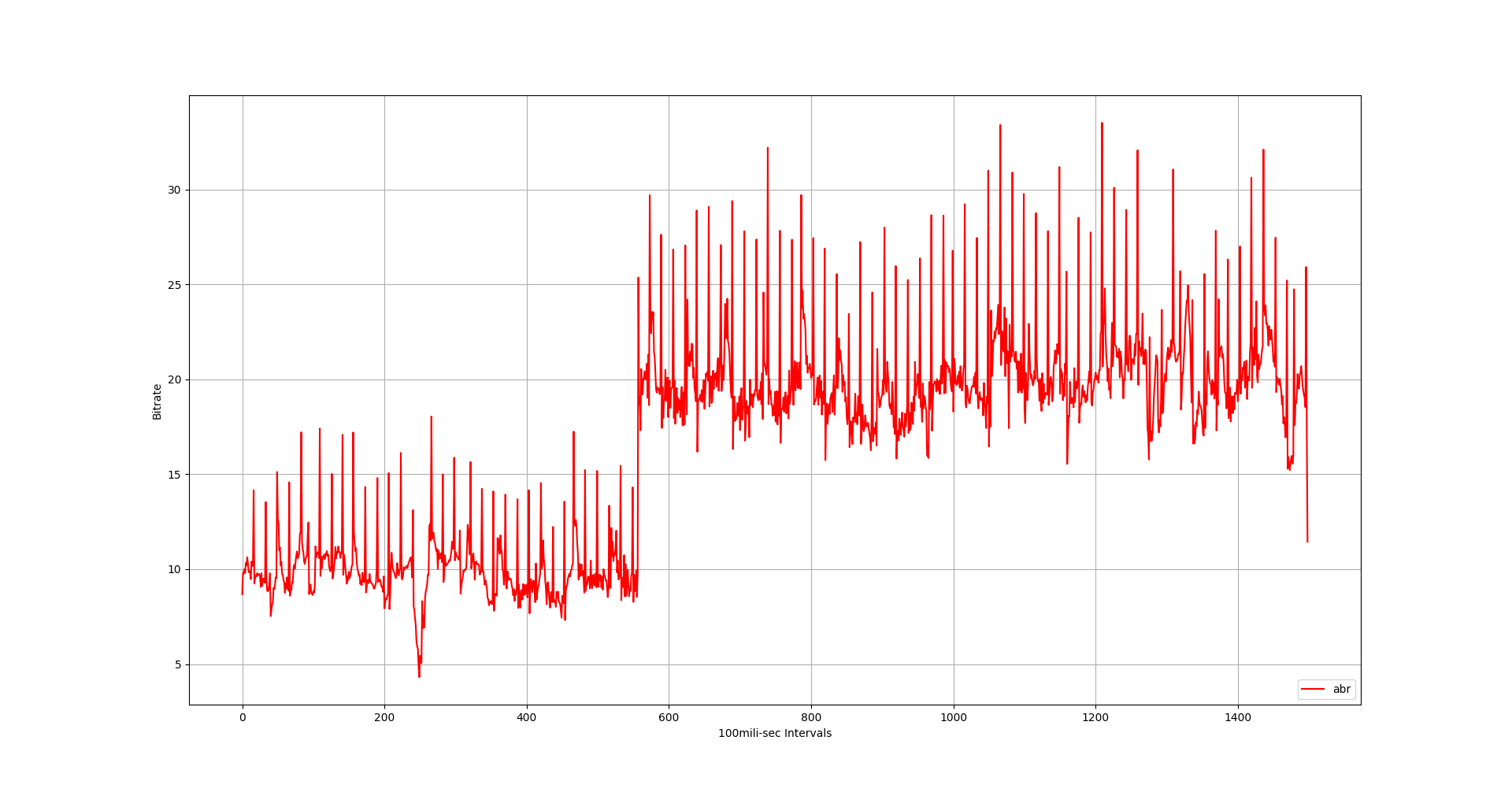


Figure 4: Bitrate plot for GOP = 50.

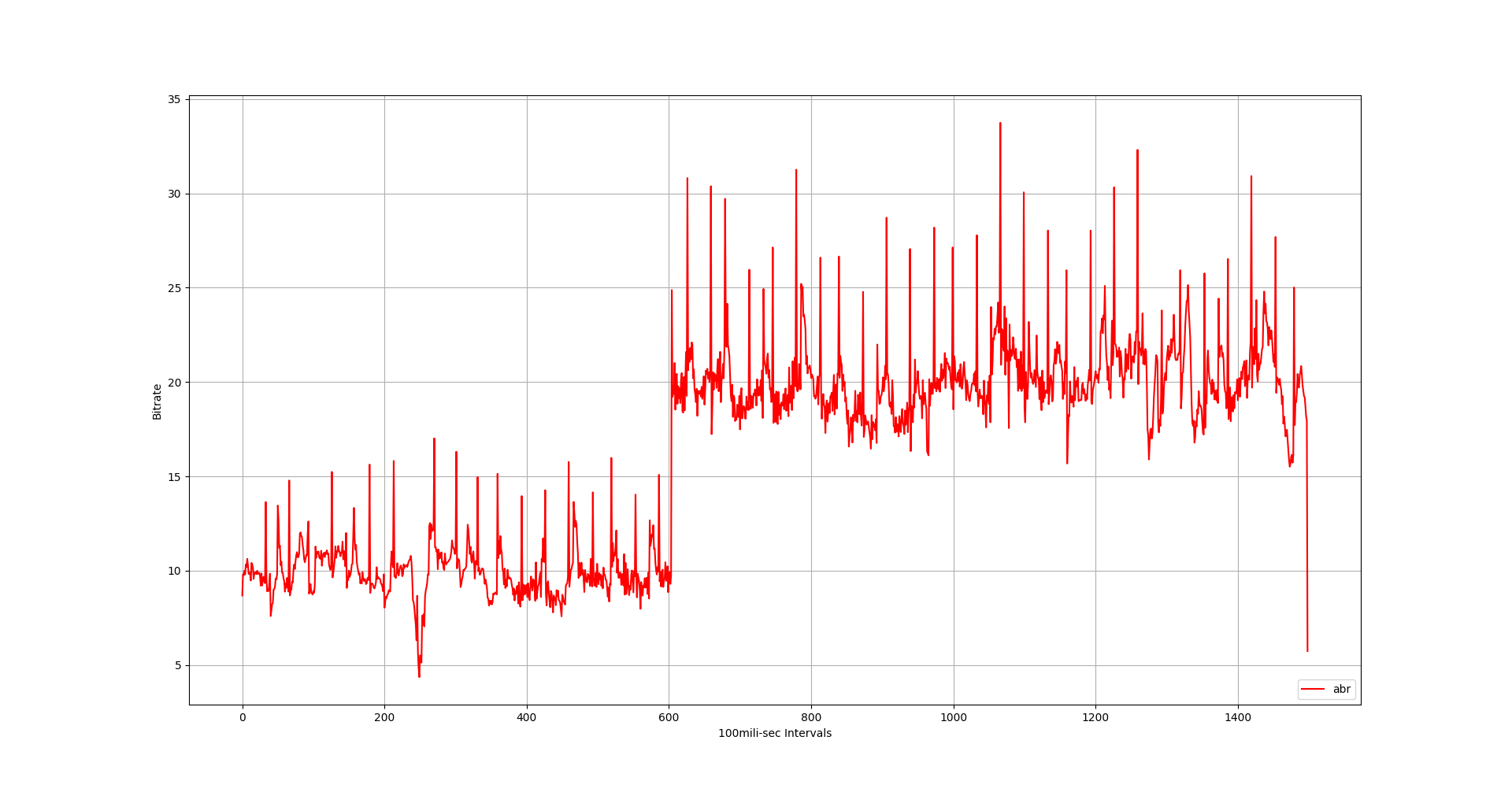


Figure 5: Bitrate plot for GOP = 100.

**2.3 Experiments with Constant Bitrate (CBR) setting in ffmpeg:**

**Motive:**

The motive of this experiment was to verify if there is any more restrictive

rate–control mode than ABR rate-control in ffmpeg.

**Description:**

Constant Bitrate (CBR) is a rate-control mode which gives almost constant

output bitrate. There is no CBR rate-control mode in ffmpeg, but we can simulate CBR using“nal-hrd=cbr:force-cfr=1” option in ffmpeg. In conjunction to this we also need to use “minrate” and “maxrate” options.

**Procedure:**

1. Run ffmpeg with “nal-hrd=cbr:force-cfr=1 option and minrate = maxrate =10Mbps.
2. Extract the logfile using some script.
3. Plot the graph using extracted file.

**Outcome:**

We get a bitrate output in which there is insignificant variation. *Figure-6* shows the output plot. But what it does is just putting filler data in simple scene frames and applying more compression on complex scene frames possibly increasing quantization parameter to give the constant bitrate.So, this is not efficient and does not match our requirement.

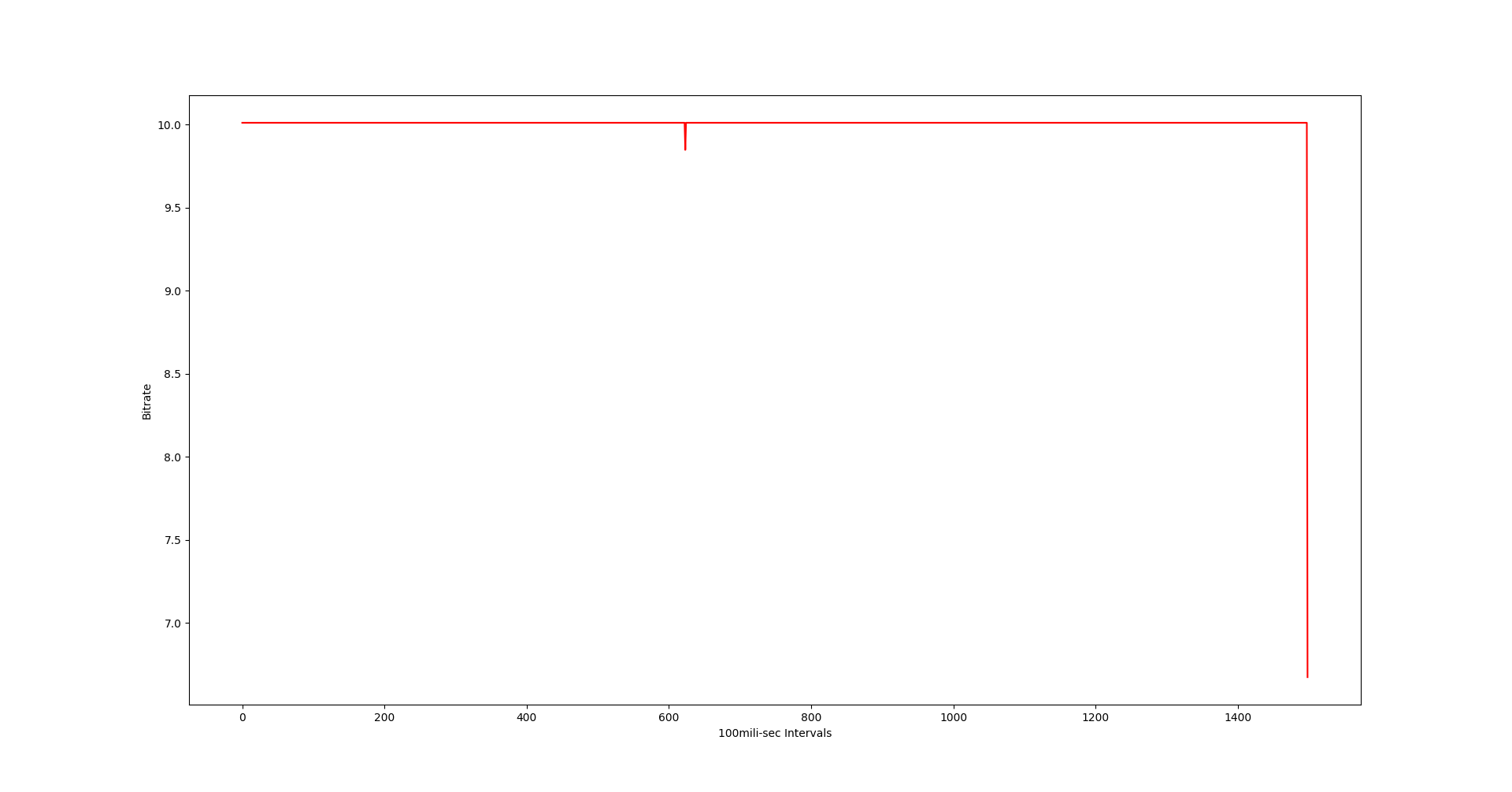


Figure 6: Plot for CBR rate-control with bitrate = 10Mbps

**2.4 Experiments with buffer size:**

**Motive:**

The motive of these experiments was to find out if we can use the vbv-buffer model

to improve output bitrate.

**Description:**

**“**The **Video Buffering Verifier** (VBV) is a theoretical MPEG video buffer model,

used to ensure that an encoded video stream can be correctly buffered, and played

back at the decoder device**.” [3]**

The “bufsize” option in ffmpeg simulates vbv-buffer model. This option should be used in conjunction with the “maxrate” option. Vbv-buffer model uses a leaky-bucket to do traffic shaping.

**Procedure:**

1. Run ffmpeg with buffer size set to zero.
2. Extract and save the log file.
3. Plot the graph using extracted file.
4. Repeat step 1 to step 3 for buffer size 667Kb and 1Mb.

**Outcome:**

During this experiment we found that smaller buffer size gives less variation in

Throughput but buffer size cannot be smaller than size of one frame.

Figure 7, 8 and 9 shows the graph for without buffer, 667Kb buffer size and 1Mb

buffer size respectively.

We also noticed that for smaller buffer-size we were not able to get 20Mbps output bitrate after switching the bitrate from 10Mbps to 20Mbps. Figure 10 shows the running average bitrate for different buffer sizes.

We also experimented with different buffer size to get the optimum value, but it was still not

clear.

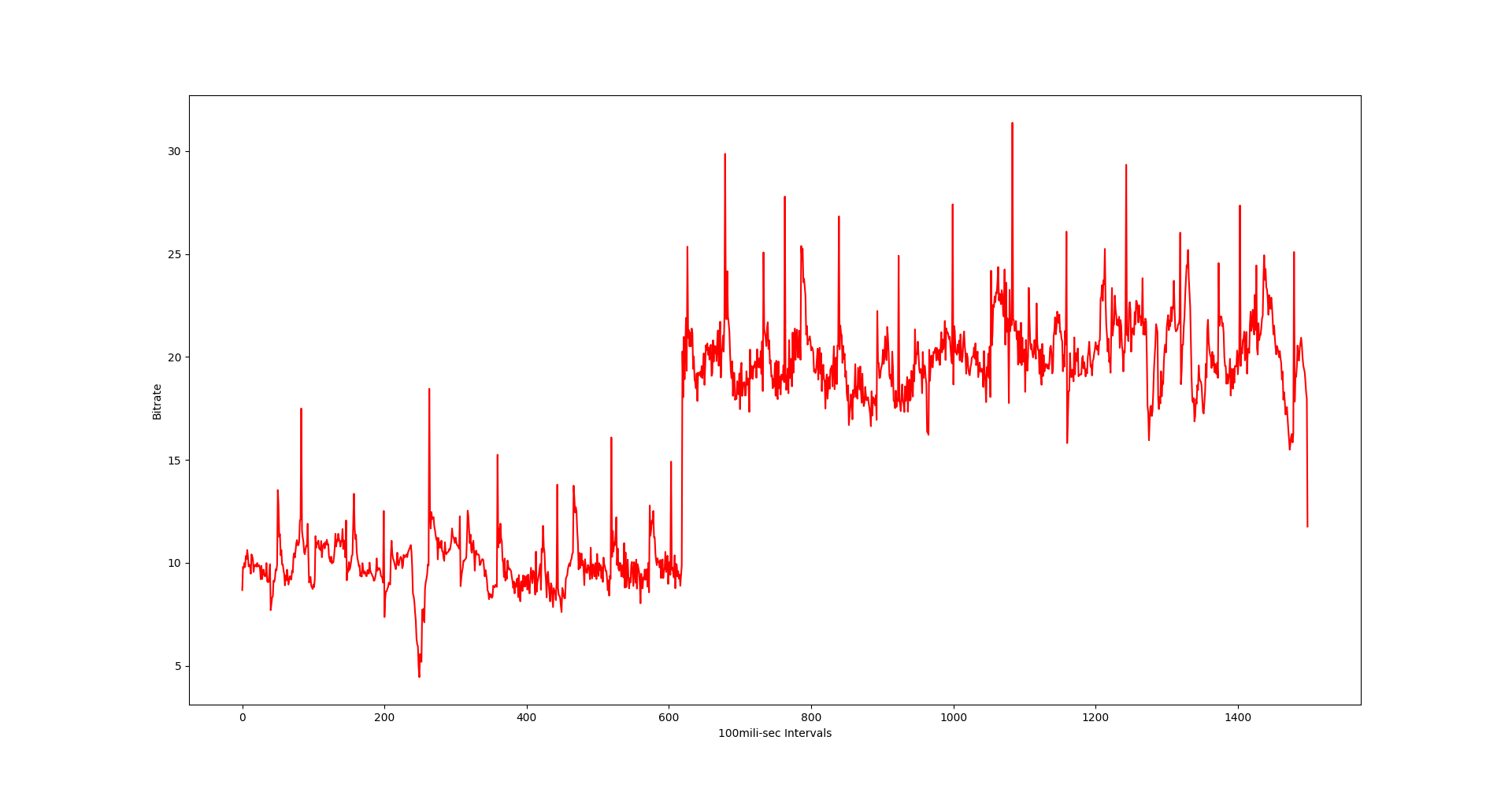


Figure 7: Output bitrate graph for 0 vbv-buffer size. (30fps video)

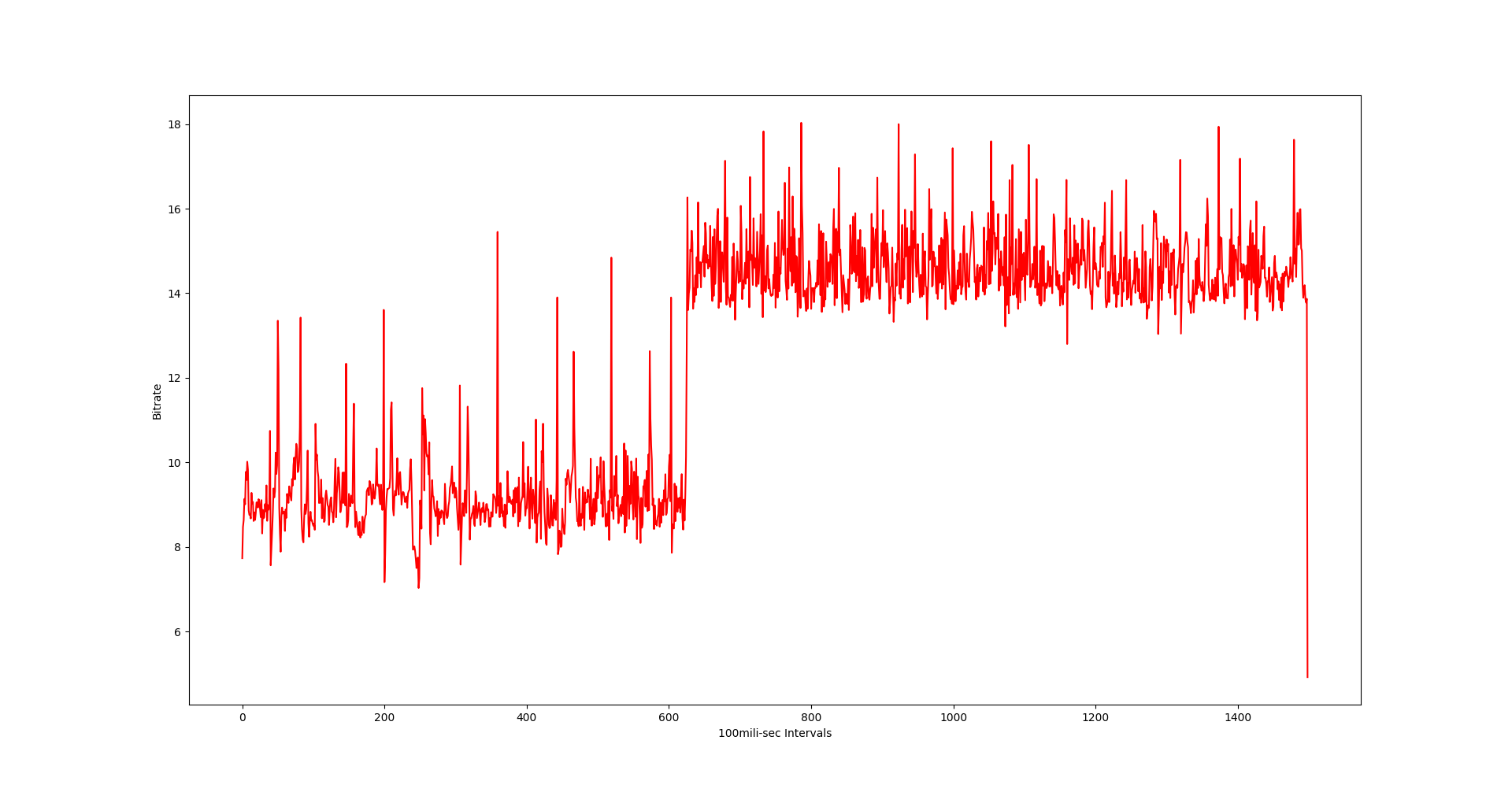


Figure 8: Output bitrate graph for 667 Kb vbv-buffer size. (30fps video)

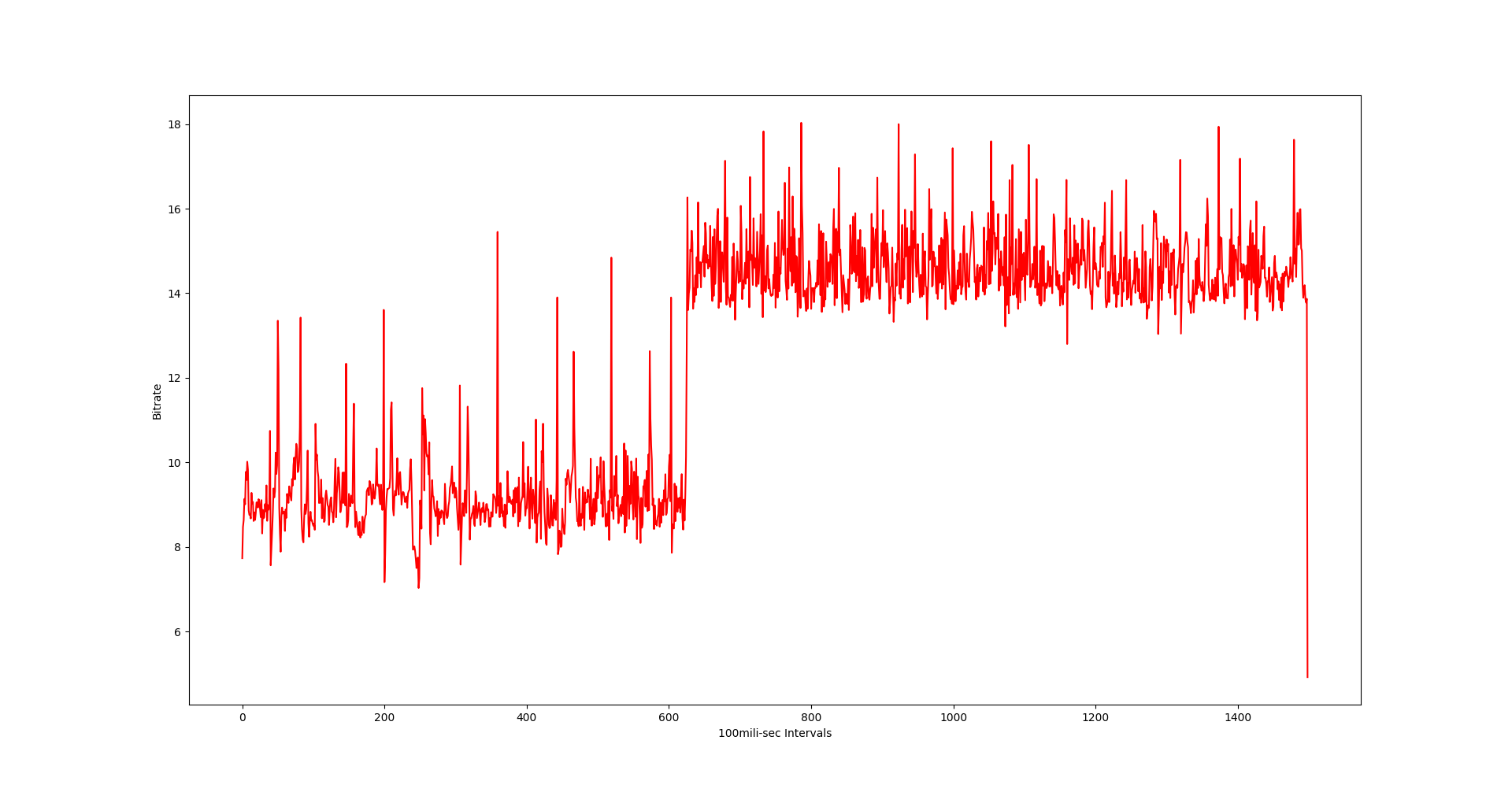


Figure 9: Output bitrate graph for vbv-buffer size 1Mb. (30fps video)

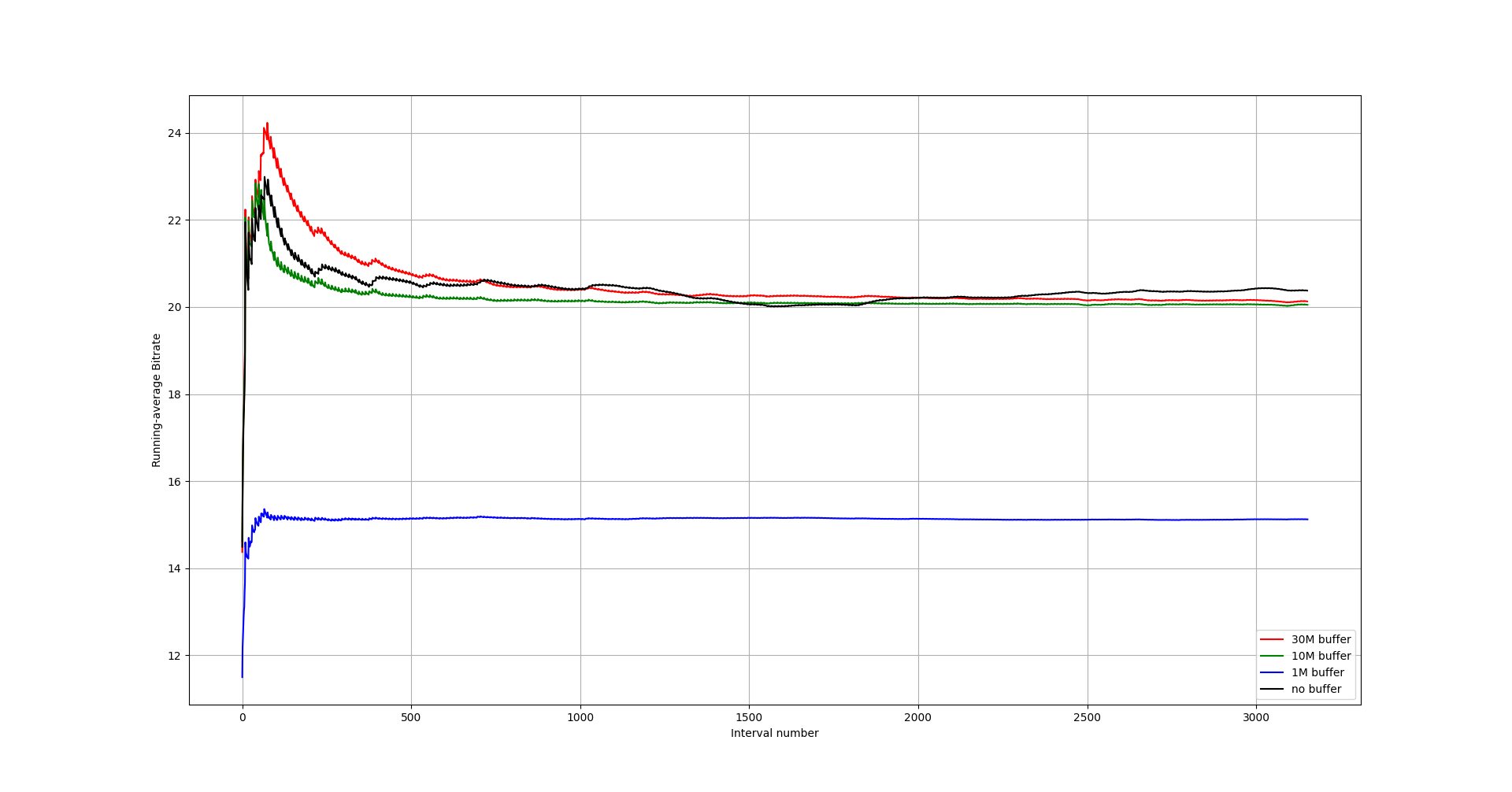


Figure 10: Running average of output bitrate with different vbv-buffer size (30fps)

**2.5 Running-average throughput:**

**Motive:**

The motive of this experiment was to observe the effect of bitrate switching on the video

traffic.

**Description:**

In this experiment we switched the target bitrate from 10 Mbps to 20 Mbps and

compared it with a normal 20Mbps bitrate output. We also switched at different time

instants and plotted them to see if different instant have different effect during bitrate

switching.

**Procedures:**

1. Run the ffmpeg with target bitrate 20Mbps.
2. Extract and save the log file.
3. Run the ffmpeg with target bitrate 10Mbps.
4. Switch the bitrate from 10Mbps to 20Mbps.
5. Extract and save the log file.
6. Truncate both the extracted log file till 20Mbps bitrate starts in second file.
7. Plot both the truncated bitrate files.
8. Repeat step-iii to step-vii for different time instants.

**Outcome:**

We found that the different time-instant does not differently affect bitrate during

switching the bitrate. And both the normal and the switched case bitrate approaches

to a constant difference.

Figure 11, 12 and 13 shows the graph for switching bitrate at 7th second, 15th second and 32nd second respectively. In the graphs green line is the switched case and red is the normal case.

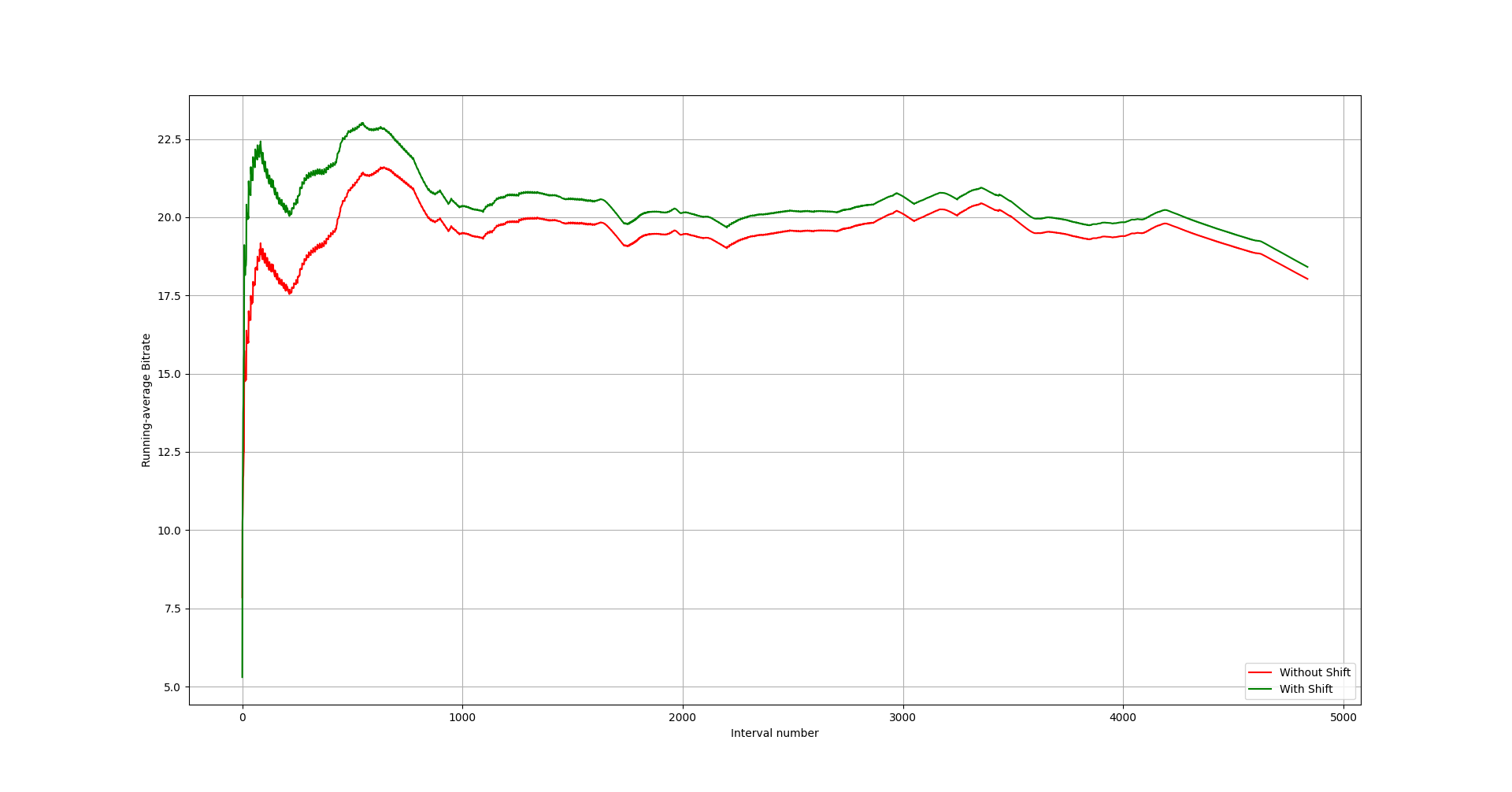


Figure 11: Switching bitrate at 7th second (60fps video)

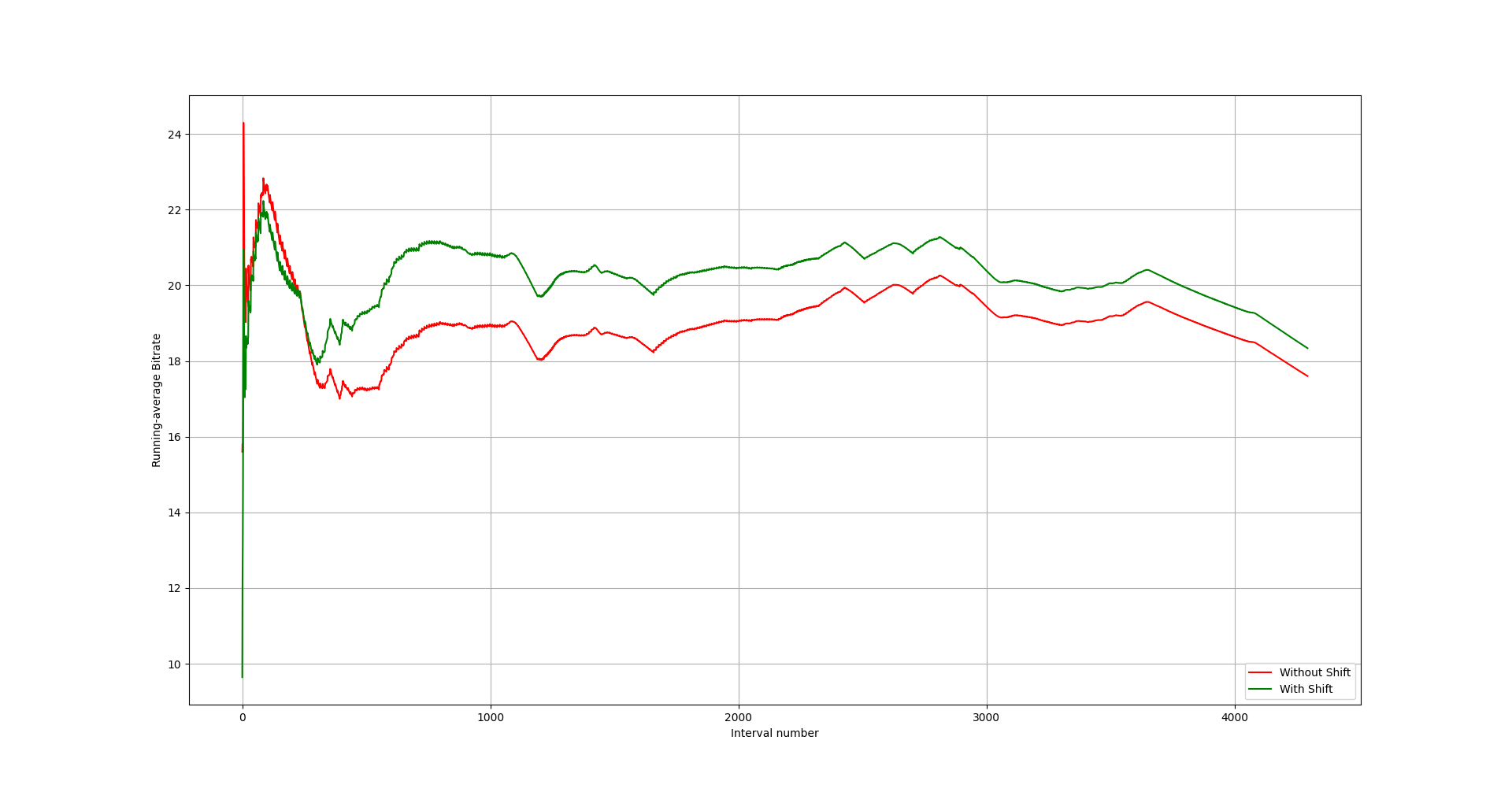


Figure 12: Switched bitrate at 15th second (60fps video)

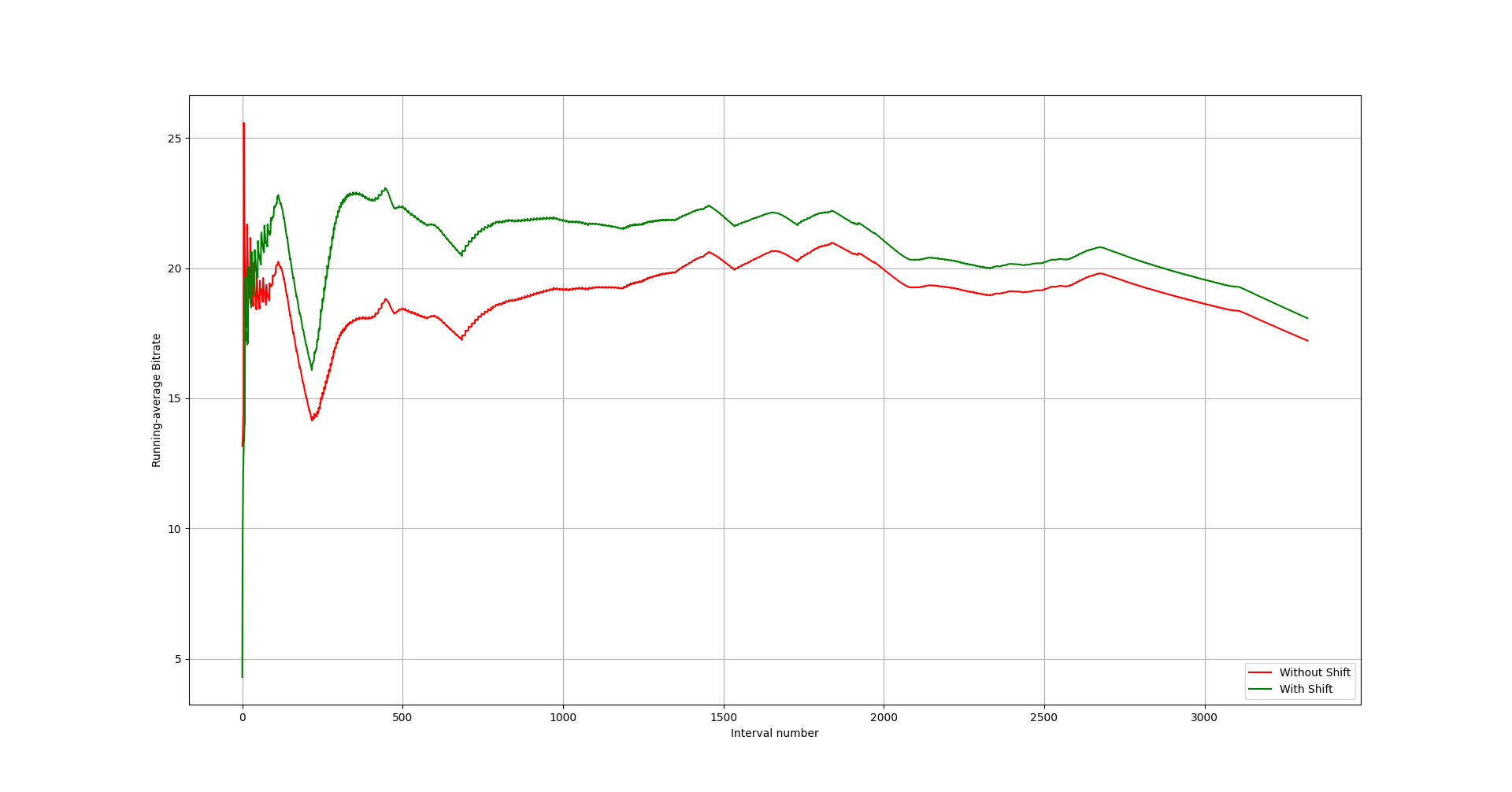


Figure 13: Switched bitrate at 32nd second (60 fps)

**2.6 End to end delay experiment:**

**Motive:**

The motive behind this experiment was to observe the effect of buffer size on the

end-to-end delay between video-sender and video-receiver machines.

**Description:**

In this experiment NTP was used to synchronize both the machines. Wi-Fi was

used to connect the two machines possibly affecting synchronization.

**Procedure:**

1. Run ffplay on receiver-side machine.
2. Run ffmpeg on the sender-side machine with buffer size set to 0.
3. Extract the timestamps and get delay information.
4. Repeat step 2 and step 3 for 5M, 10M and 100M buffer sizes.
5. Plot the graph.

**Outcome:**

The experiment shows that increasing buffer size increases delay, zero buffer (Blue

curve) also gives more delay than small buffer.

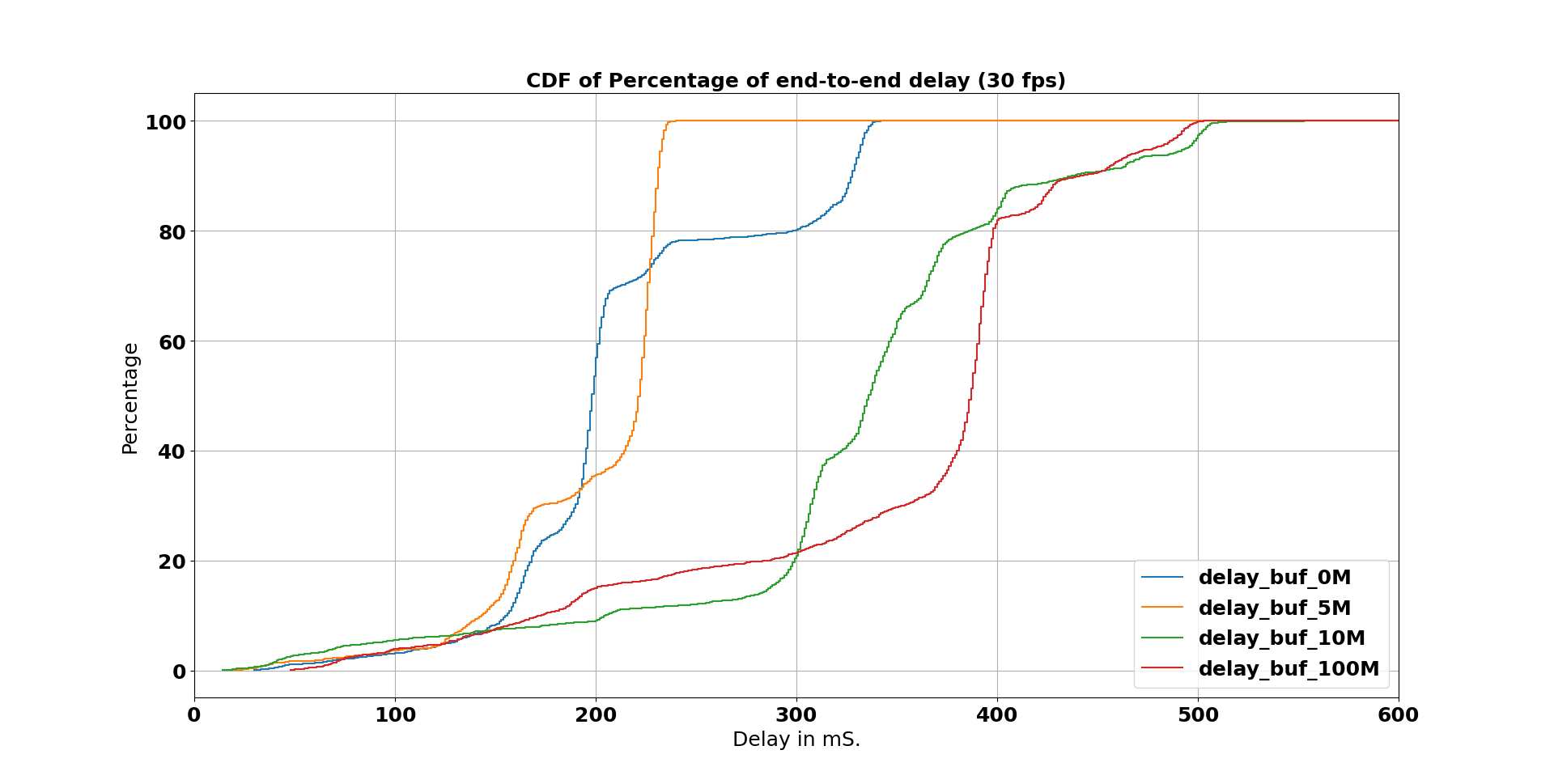


Figure 1: End to end delay graph

**3. CONCLUSION**

Based on the experiment following things can be concluded-

1. GOP size can be increased to make output bitrate variation smooth.
2. CBR rate-control method is not suitable for our application.
3. Smaller buffer size gives less variation in output-bitrate, but vbv-buffer model requires more research to be done in order to understand it properly.
4. Switching bitrate does not alters the output bitrate distribution. Also, the time instant at which bitrate switches do not affect output bitrate.
5. Increase in buffer size also increases end to end delay.

**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 technical problem. My mentors were helpful and guided me throughout the internship. I learned how to work in a team and it gave me confidence that I can work in a team on any real-life project which requires my skillset.

Apart from soft-skill development, I also gained technical 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 used 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 stylefiles (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 keep using Linux.

**4.2 H.264:**

H.264 is a video codec standard to compress and encode raw video frames because, it is not practical to store and transfer raw video frames. It will take huge resources like memory and throughput. H.264 is also known 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 socket over Wi-Fi network. 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 provided books to learn basics of 802.11 protocol which is used to implement Wireless LAN. From those books, 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 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. I also learned about RTS/CTS handshake for collision avoidance and NAV which is used to reserve the medium by a station.

**References**

[1] Wikipedia: https://en.wikipedia.org/wiki/ffmpeg

[2] Wikipedia: https://en.wikipedia.org/wiki/advanced\_video\_coding

[3] Wikipedia: https://en.wikipedia.org/wiki/video\_buffering \_verifier