HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY

---□&□---

ASSIGNMENT REPORT

TOPIC: libopenh264 and opus codec

Instructor	Assoc. Prof. Phạm Văn Tiến	
	Dr. Lưu Quang Trung	
Member	Vũ Minh Hiển	20224311
	Lê Hoàng Anh	20224297
	Đào Hữu Mão	20233865
	Phạm Thị Thanh Trúc	20224293
	Vũ Xuân Anh	20233832
Subject	Multimedia data compression and coding	
Class	157320	
Group	10	

Table of Contents

I. Introduction

- Students who have been contributing for the project:

Vũ Minh Hiển	20224311	Video transmission, Measure quality and analyze results
Lê Hoàng Anh	20224297	Video transmission, Develope a program discard function, Measure quality and analyze results
Đào Hữu Mão	20233865	Video transmission, Add dynamic graphics overlays, Develope a program discard function
Phạm Thị Thanh Trúc	20224293	Video transmission, Measure quality and analyze results
Vũ Xuân Anh	20233832	Video transmission, Add dynamic graphics overlays, Write report

- Brief view of the project:
  This project evaluates the performance of *libopenh264* (video) and *libopus* (audio) codecs in a simulated real-time transmission scenario. A short video with clear facial visuals, named *landscape_original.mp4*, was recorded and encoded using FFmpeg. The encoded video was transmitted over Wi-Fi between two computers at five different bitrates (500k–2500k) via UDP. At the receiver, VLC was used to stream and save the transmitted video.
  A custom Python script simulated packet loss, added dynamic overlays, and measured video quality using PSNR and SSIM. Over five experiments were conducted to ensure consistency. Challenges such as dynamic IP changes and short video length were addressed by setting static IPs and re-recording a longer clip. The project highlights the impact of bitrate and network conditions on codec performance in multimedia streaming.

II. Details

2.1. Overview about two codecs: libopenh264 and libopus.

- Video codec: libopenh264 (H.264) is an open-source library implementing the H.264 (MPEG-4 AVC) video codec, developed by Cisco. It provides efficient compression for high-quality video streaming and storage, widely used in applications like video conferencing and media playback. It's optimized for real-time encoding and decoding with low latency.
- Audio codec: libopus (Opus) is an open-source library implementing the Opus audio codec, designed for high-quality, low-latency audio streaming and storage. It excels in compressing speech and music, supporting bitrates from 6 kbps to 510 kbps with excellent quality. Highly versatile, it's used in VoIP, video conferencing, and media playback, and is standardized by the IETF.

2.2. Record the audio

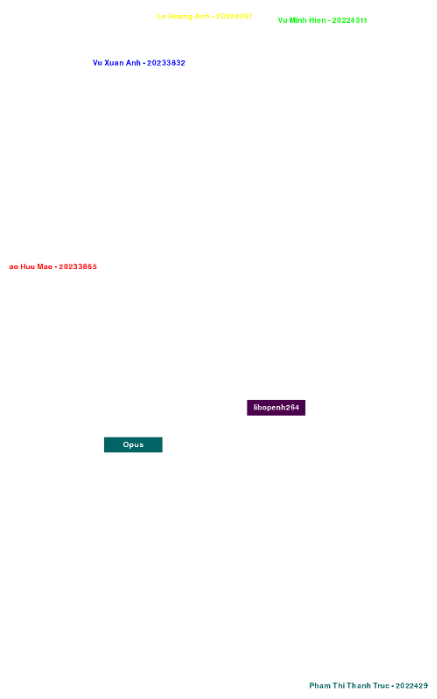
- All of us recorded two video with landcape and portrait orientation at 19/05/2025
- Then we decide to use **Portrait video** at **.mp4** to peform next steps and Landscape one for contingency

2.3. Create overlay before transferring

2.2.1. Use *pygame* library to create overlay with transparent background

- First, we use pygame to create dynamic content include: **Name of 5 students in group + StudentID** and **Name of 2 codec.**
- The result give us a folder *frame* which containt transparent background overlay

Folder frame



Overlay sample

2.2.2. Use *moviepy* library to put the overlay in frame file on video

- After that, we use moviepy to put an overlay frame on the original video. This is similar to 2D motion.
- This line positions the overlay (image sequence) in the center of the original video, both horizontally and vertically. You can change these values, for example:
  ("left", "top") – positions the overlay in the top-left corner.
  (x, y) – positions the overlay at specific pixel coordinates.

overlay = overlay.with_position(("center", "center"))

- This creates a composite clip by overlaying the overlay on top of the video. The order in the list is important: the first layer (video) is the background, and the subsequent layer (overlay) is drawn on top of it.

final = CompositeVideoClip([video, overlay])

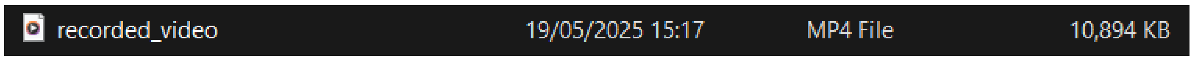
- This exports the final video to a file named output_with_overlay.mp4.

final.write_videofile("output_with_overlay.mp4", codec=" libopenh264 (H.264) ", audio_codec=" libopus (Opus) ")

- The result is a video that has overlay in frame folder on that. The frame has larger size than original video so final video has frame that some overlay

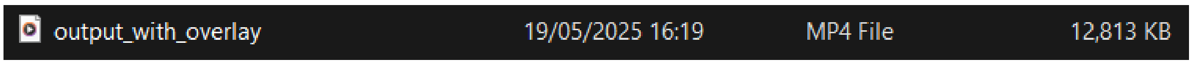
Sample frame

– We can see that the video with overlay has bigger size than original one:



```
recorded_video            19/05/2025 15:17      MP4 File            10,894 KB
```

Original video



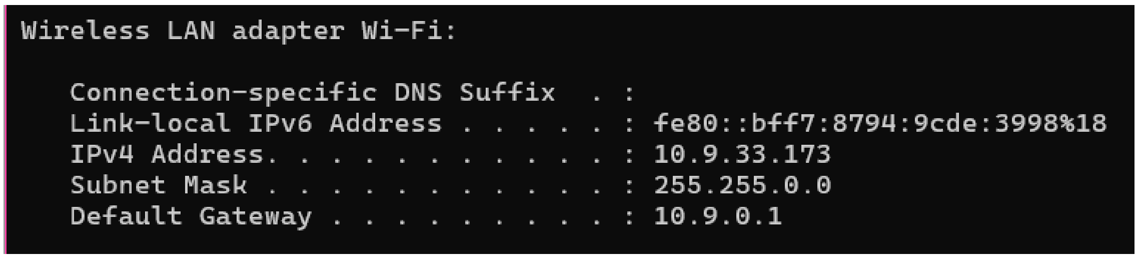
```
output_with_overlay       19/05/2025 16:19      MP4 File            12,813 KB
```

Video with overlay

2.4. Transmission

– To obtain the reconstructed videos and associated statistics, we have chosen the video codec "libopenh264" and audio codec "Opus".

– After get the overlaid video, we need to find the IP of the received computer by running "ipconfig" in Windows PowerShell and searching for "IPv4 Address" in Wireless LAN adapter Wi-fi. The result can be show as below picture:



```
Wireless LAN adapter Wi-Fi:

   Connection-specific DNS Suffix  . :
   Link-local IPv6 Address . . . . . : fe80::bff7:8794:9cde:3998%18
   IPv4 Address. . . . . . . . . . . : 10.9.33.173
   Subnet Mask . . . . . . . . . . . : 255.255.0.0
   Default Gateway . . . . . . . . . : 10.9.0.1
```

"ipconfig" command result

– Then, at the sending computer, we go to the folder that contains the video recorded and open it in the terminal. In the terminal, run suitalbe code as form:

ffmpeg -re -i <transmitting_mp4_file> -c:v libopenh264 -b:v <bit_rate> -profile:v main -g 30 -c:a libopus -b:a 192k -f mpegts -mpegts_flags system_b -flush_packets 0 udp://<Target_IP>:<Target_Port>?pkt_size=1280

Example:

ffmpeg -re -i output_with_overlay.mp4 -c:v libopenh264 -b:v 500k -profile:v main -g 30 -c:a libopus -b:a 192k -f mpegts -mpegts_flags system_b -flush_packets 0 udp://10.9.33.173:1234?pkt_size=1280

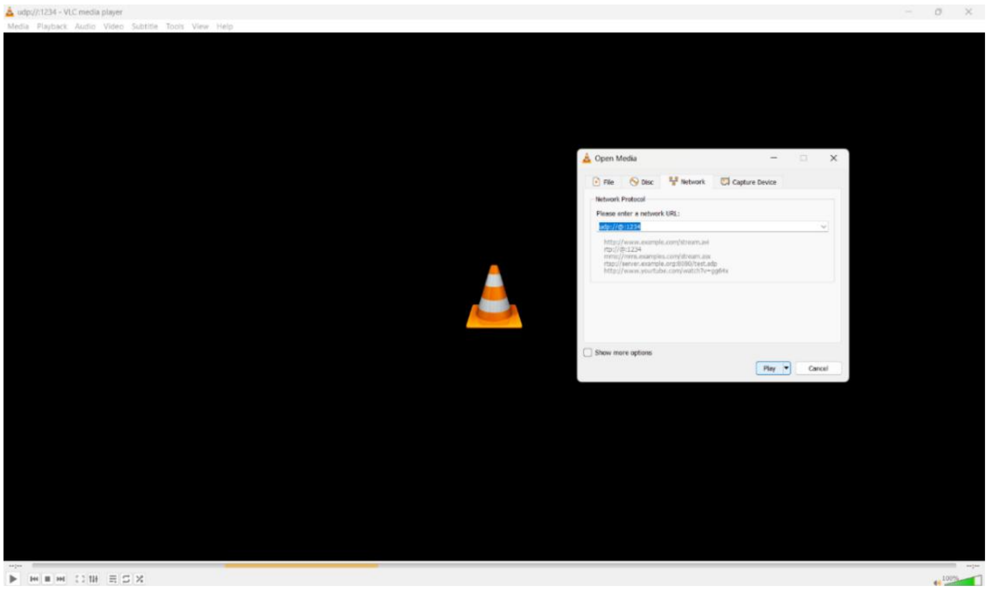
Explanation of Each Component:

1. **ffmpeg**:
   This is the command-line tool used for handling multimedia files, including video and audio encoding, decoding, muxing, demuxing, streaming, and more.
2. **-re**:
   Stands for "real-time." This flag tells FFmpeg to read the input file at its native frame rate, simulating real-time playback. It's commonly used for streaming to avoid buffering issues by ensuring the input is processed at a natural pace.
3. **-i output_with_overlay.mp4**:
   Specifies the input file, output_with_overlay.mp4. This is the video file FFmpeg will process, likely a file with some overlay (e.g., text, logo, or graphics) already applied.
4. **-c:v libopenh264**:
   Specifies the video codec as libopenh264, an open-source H.264 video encoder. H.264 is a widely used video compression standard that provides good quality at relatively low bitrates.
5. **-b:v 500k**:
   Sets the video bitrate to 500 kbps (kilobits per second). This controls the amount of data used to represent the video, affecting quality and file size. A lower bitrate like 500k is suitable for low-bandwidth streaming but may reduce quality.
6. **-profile:v main**:
   Sets the H.264 profile to main. The Main profile is a common H.264 profile that balances compression efficiency and compatibility. It's widely supported by devices and suitable for most streaming scenarios.
7. **-g 30**:
   Sets the GOP (Group of Pictures) size to 30 frames. This means a keyframe (I-frame) is inserted every 30 frames. A smaller GOP size improves error resilience and seeking but increases bitrate for the same quality.
8. **-c:a libopus**:
   Specifies the audio codec as libopus, which uses the Opus audio codec. Opus is a highly efficient, lossy audio codec optimized for low-latency streaming and voice applications, offering high quality at low bitrates.
9. **-b:a 192k**:
   Sets the audio bitrate to 192 kbps. This determines the quality of the audio stream. 192 kbps is a relatively high bitrate for Opus, providing excellent audio quality suitable for music or complex audio.
10. **-f mpegts**:

Specifies the output format as MPEG-TS (MPEG Transport Stream). MPEG-TS is a container format commonly used for streaming video and audio over networks, especially for broadcast and UDP streaming.

11. **-mpegts_flags system_b**:
    Sets specific flags for the MPEG-TS output. The system_b flag likely refers to compatibility with certain systems or standards (e.g., DVB or ATSC), ensuring the stream adheres to specific requirements for broadcast systems. Exact behavior depends on FFmpeg's implementation.
12. **-flush_packets 0**:
    Disables automatic flushing of packets. By default, FFmpeg may flush packets to the output as soon as they are ready. Setting this to 0 ensures packets are only sent when complete, which can help with synchronization in streaming scenarios.
13. **udp://10.9.33.173:1234?pkt_size=1280**:
    - Specifies the output destination as a UDP stream to the IP address 10.9.33.173 on port 1234.
    - UDP (User Datagram Protocol) is a lightweight, connectionless protocol often used for real-time streaming because it prioritizes speed over reliability (no retransmission of lost packets).
    - ?pkt_size=1280 sets the packet size to 1280 bytes, which is typical for MPEG-TS over UDP to match network MTU (Maximum Transmission Unit) constraints and optimize delivery.

– At the receiving computer, we use VLC to stream and save the video after the communication. We go to VLC, select "Open Network Stream" in "Media", write the UDP address in "Network" and finally press "Play".



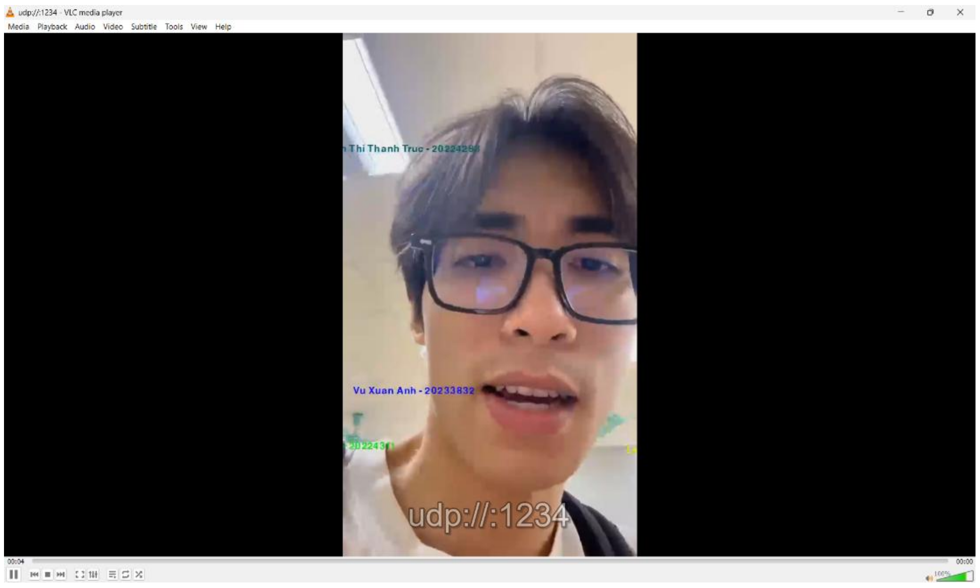
Network Protocol

Please enter a network URL:

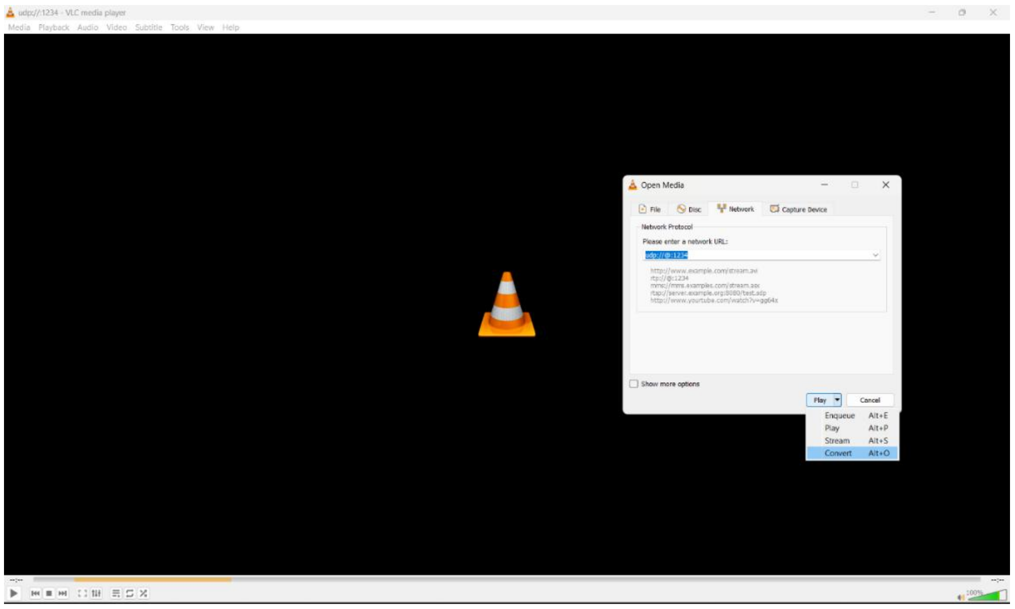
udp://@:1234

http://www.example.com/stream.avi
rtp://@:1234
mms://mms.examples.com/stream.asx
rtsp://server.example.org:8080/test.sdp
http://www.youtube.com/watch?v=gg64x

– Then the screen during transmitting should be like that:


udp://:1234 - VLC media player
Media Playback Audio Video Subtitle Tools View Help
Thi Thanh Truc - 2022425
Vu Xuan Anh - 20233832
udp://:1234
00:04
00:00

– To save the video after transmitting. At the receiving computer, we use VLC to stream and save the video after the communication. We go to VLC, select “Open Network Stream” in “Media”, write the UDP address in “Network” and press “Convert”.



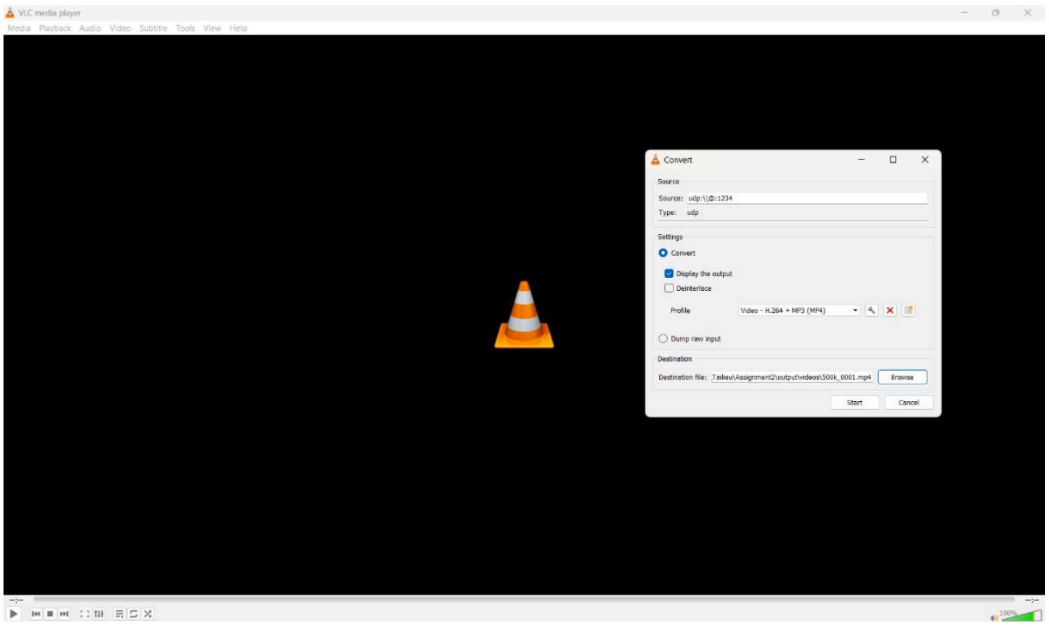
Network Protocol

Please enter a network URL:

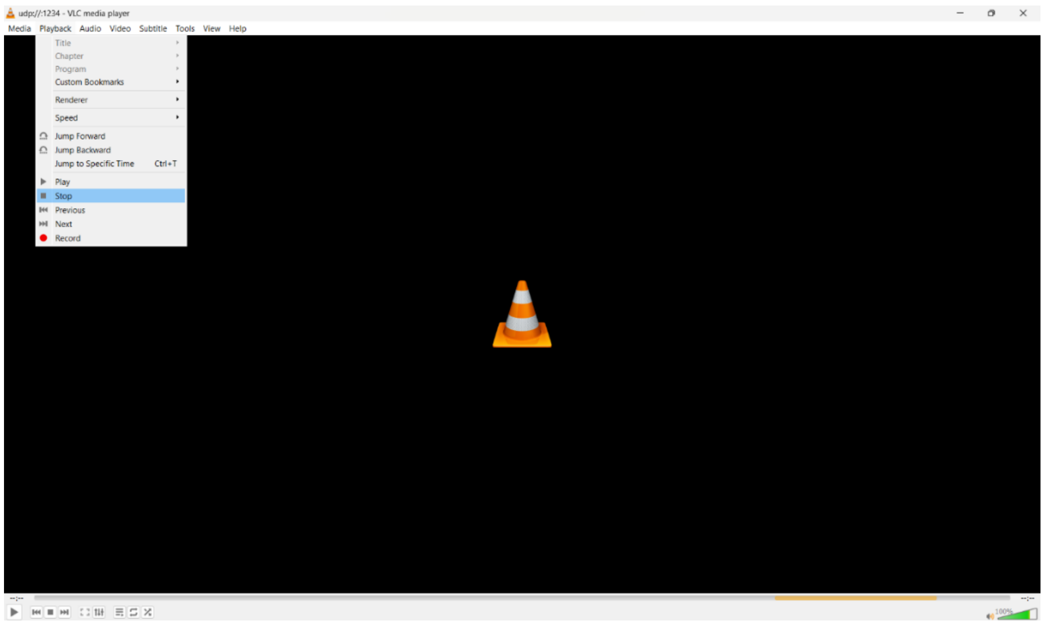
udp://@:1234

http://www.example.com/stream.avi
rtp://@:1234
mms://mms.examples.com/stream.asx
rtsp://server.example.org:8080/test.sdp
http://www.yourtube.com/watch?v=gg64x

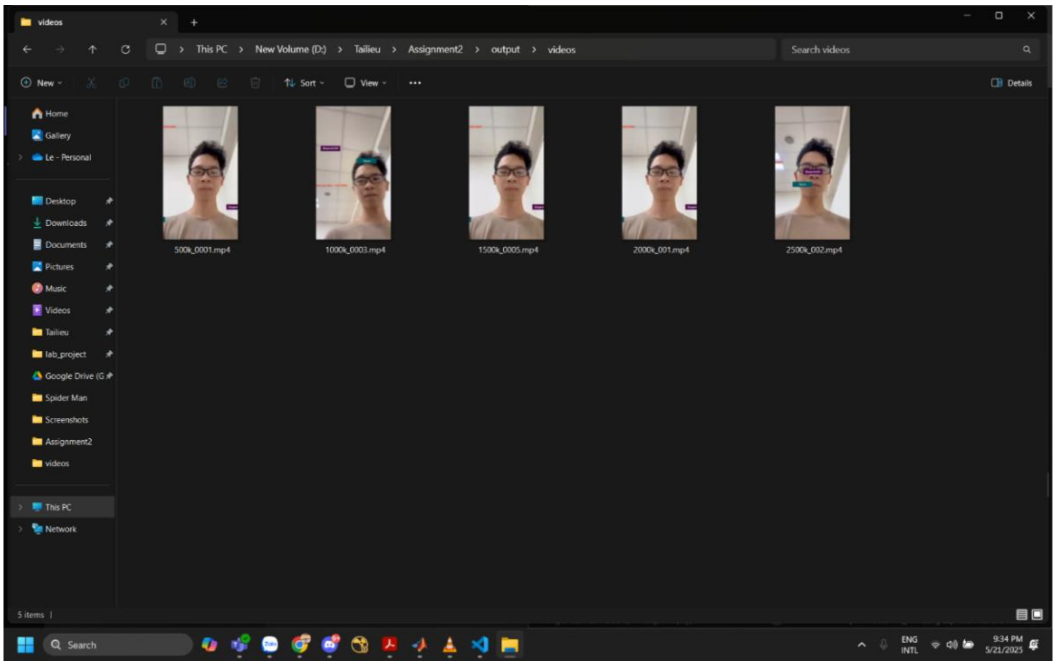
– After that select "Display the output" in order to preview the received videos, browse the destination where we want to save the transmitted videos(after overlaid and dropped packets) and click "Start". We can also choose the decoder codec at the "Profile" line.


Source: udp:\\@:1234
Type: udp
[✓ Convert]
[✓ Display the output]
Profile: Video - H.264 + MP3 (MP4)
Destination file: ,Tailieu\Assignment2\output\videos\500k_0001.mp4

– When sucessfully transmitted videos, go to "Playback" - top left screen bar and select "Stop" button in order to save the videos.


udp://:1234 - VLC media player
Media Playback Audio Video Subtitle Tools View Help
Title
Chapter
Program
Custom Bookmarks
Renderer
Speed
Jump Forward
Jump Backward
Jump to Specific Time Ctrl+T
Play
Stop
Previous
Next
Record

– We do the transmission 5 times with different video bit rate, discarding rate and receive 5 corresponding video:



- We chose discard rates ranging from 0.1% to 2% to reflect realistic levels of packet loss commonly observed in wireless and real-time streaming environments. A discard rate of 0.1% simulates near-optimal network conditions, while higher rates such as 1% or 2% represent mild to moderate network congestion or interference. This range allows us to assess the codec's robustness to typical packet loss scenarios without entering unrealistic or extreme conditions rarely encountered in practical applications.
- Our project described did not utilize video bitrates in the 500k-2500k range because the primary goal was to evaluate libopenh264 for high-quality video streaming. Bitrates this low would typically result in severely compromised video quality, characterized by significant artifacts and loss of detail, especially for content featuring "clear facial visuals." Such poor quality would likely fall outside the intended performance spectrum for libopenh264 in typical streaming applications and would not align with the project's aim to assess quality variations where H.264 is commonly used. Furthermore, pairing extremely low-bitrate video with the chosen high-quality audio setting of 192 kbps for libopus would create an imbalanced and less practical multimedia scenario for their investigation into real-time transmission performance.

2.5. Quality measurement statistics and comments

2.5.1. Measurement

1. PSNR (Peak Signal-to-Noise Ratio) measures the ratio between the maximum possible power of a signal and the power of corrupting noise that affects the quality of its representation.
   - Y channel (Luminance): a measure of the brightness
   - U channel (Chrominance): a measure of the color information
   - V channel (Chrominance): another measure of color information
   - Average: combined the measurement of quality across three channels (Y,U,V)

– Min PSNR: lowest quality found
– Max PSNR: highest quality found

We achieved the table of values:

BITRATE-DISCARDING RATE	Y	U	V	Average (dB)	Min (dB)	Max (dB)
500k – 0.1%	15.31573	30.05889	28.55362	16.96826	10.6100	42.9400
1000k- 0.3%	17.05054	31.96071	30.03422	18.69303	10.4700	45.7900
1500k-0.5%	15.11782	30.32909	28.32651	16.77777	10.4700	43.1900
2000k-1%	19.32464	33.50654	32.33266	20.95455	12.7000	47.0000
2500k-2%	13.24073	29.52764	26.33554	14.91054	10.3600	44.5500

Comments:

a. Average Quality (Average PSNR)
   - The 2000 kbps bitrate achieves the highest average PSNR (20.95 dB), indicating the best overall video quality among all tested settings.
   - Other bitrates result in lower average PSNRs, especially 2500 kbps, which scores only 14.91 dB, reflecting poor quality.
   - Interestingly, 1000 kbps (18.69 dB) performs better than both 1500 kbps (16.77 dB) and 500 kbps (16.96 dB), showing that a higher bitrate does not always guarantee better quality.

b. Color Components (Y, U, V)
   - The Y component (luminance) consistently has the lowest PSNR values (ranging from 13.24 dB to 19.32 dB), indicating that brightness and detail are most affected by compression and noise.
   - The U and V components (chroma) retain significantly higher PSNRs (between 26.33 dB and 33.50 dB), suggesting that color information is preserved better than brightness.

c. Stability (Min - Max PSNR)
   - Minimum PSNR ranges from 10.36 dB to 12.70 dB, revealing moments of very poor video quality (heavy noise, detail loss).
   - Maximum PSNR reaches up to 47.00 dB (at 2000 kbps), indicating that some video segments are nearly perfect, unaffected by noise or compression artifacts.

d. Observations by Bitrate
   - 500 kbps:

+ Low quality (16.96 dB).
+ Suitable for use cases with minimal quality requirements.

- 1000 kbps:
  + Significant improvement (18.69 dB).
  + Offers a good balance between bandwidth and quality.
- 1500 kbps:
  + Unexpected drop in quality (16.77 dB).
  + Possibly caused by compression artifacts or packet loss.
- 2000 kbps:
  + Best quality overall (20.95 dB).
  + Suitable for high-quality video streaming.
- 2500 kbps:
  + Worst quality despite highest bitrate (14.91 dB).
  + May result from inefficient compression or bitrate overflow.

e. Conclusion and Recommendations

- The average PSNR tends to increase as the bit rate rises from 500k to 2000k, indicating that video quality improves when more data is transmitted.
- Exception at 2500k: Despite being the highest bitrate, 2500k results in a lower average PSNR (14.91 dB), suggesting that quality degradation may occur due to network noise, packet loss, or issues in video reception.
- Variation range (Min/Max): The minimum and maximum PSNR values at each bitrate reflect the fluctuations in quality during transmission, possibly caused by device positioning or environmental network interference.

2. SSIM (Structural Similarity Index): a metric used to measure the similarity between two images or video frames, considering changes in luminance, contrast, and structure. The values are typically between 0 and 1, where 1 indicates perfect similarity. The y,u and v are channels for luminance and chrominance as above.

BITRATE-DISCARDING RATE	Y	U	V	Avarage
500k-0.1%	0.71303	0.94074	0.92670	0.78660
1000k-0.3%	0.73870	0.94824	0.93328	0.80605
1500k-0.5%	0.70475	0.94281	0.92982	0.78194
2000k-1%	0.78154	0.95492	0.94615	0.83787
2500k-2%	0.65489	0.94222	0.92800	0.74830

Comments:

a. General
  - The average SSIM increases steadily from 500kbps (0.786) to 2000kbps (0.838), consistent with the earlier PSNR trend → Video quality improves as bitrate increases (up to 2000kbps).
  - Exception at 2500kbps: SSIM drops significantly to 0.748 despite being the highest bitrate, possibly due to:
  - High packet loss rate (discard rate of 2% — the highest).
  - An "over-bitrate" effect where the compression algorithm becomes less effective.

b. Evaluation by Bitrate
  - 500kbps – 0.1% loss:
    + Lowest average SSIM (0.786).
    + Y component (0.713) most affected → low brightness, poor details.
    + Color channels (U/V ~0.94) more stable but still lower than higher bitrates.
    + Suitable for: low-bandwidth applications (e.g., low-quality video calls).
  - 1000kbps – 0.3% loss:
    + SSIM improves to 0.806, especially Y component (0.738).
    + Despite slightly higher packet loss, quality surpasses 500kbps.
    + Suitable for: standard 720p streaming.
  - 1500kbps – 0.5% loss:
    + Unexpected SSIM drop to 0.781, lower than 1000kbps.
    + Possibly due to higher packet loss or suboptimal compression.
    + Recommendation: review compression settings or reduce packet loss.
  - 2000kbps – 1% loss:
    + Highest SSIM (0.838), with Y component at 0.781 and color U/V ~0.95 → sharp, clear video.
    + Ideal for: high-quality applications (HD streaming, video conferencing).
  - 2500kbps – 2% loss:
    - Sharp SSIM drop to 0.748, with Y as low as 0.654.
    - High packet loss severely disrupts video structure.
    - Recommendation: lower the bitrate or apply error correction (e.g., FEC).

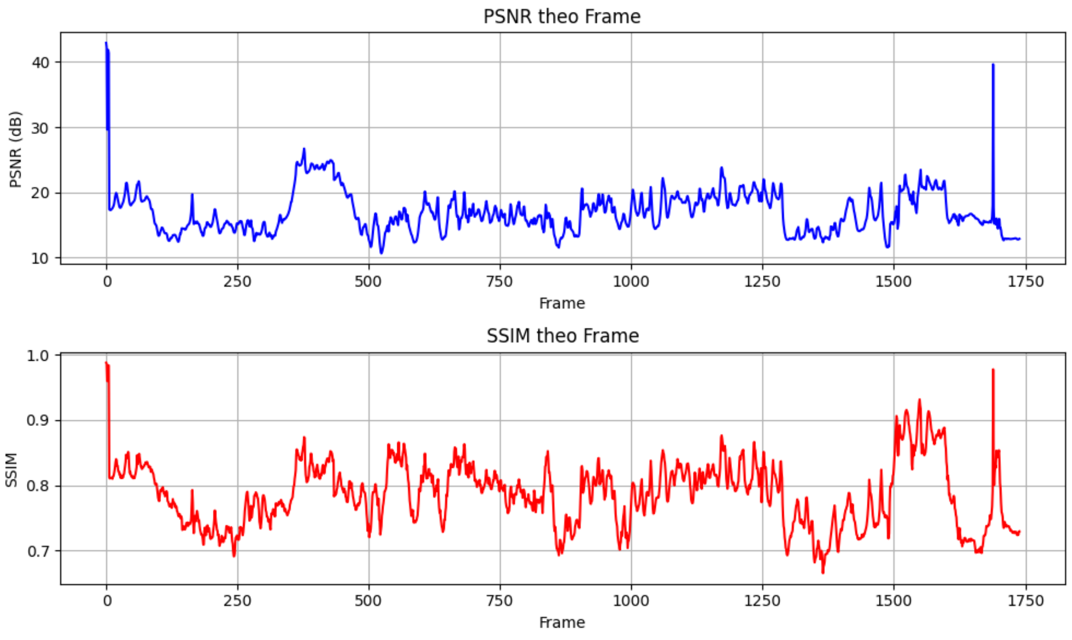
c. Comparison between SSIM and PSNR
  - Similarities:
    - Both metrics confirm 2000kbps as the optimal bitrate.
    - 2500kbps shows the worst quality, despite the highest bitrate.
  - Differences:
    - SSIM is more sensitive to structural distortions (e.g., blurring, blocking), while PSNR focuses on pixel-wise errors.
    - SSIM for the Y component is consistently lower than U/V, indicating brightness is most vulnerable — consistent with PSNR findings.

d. Summary

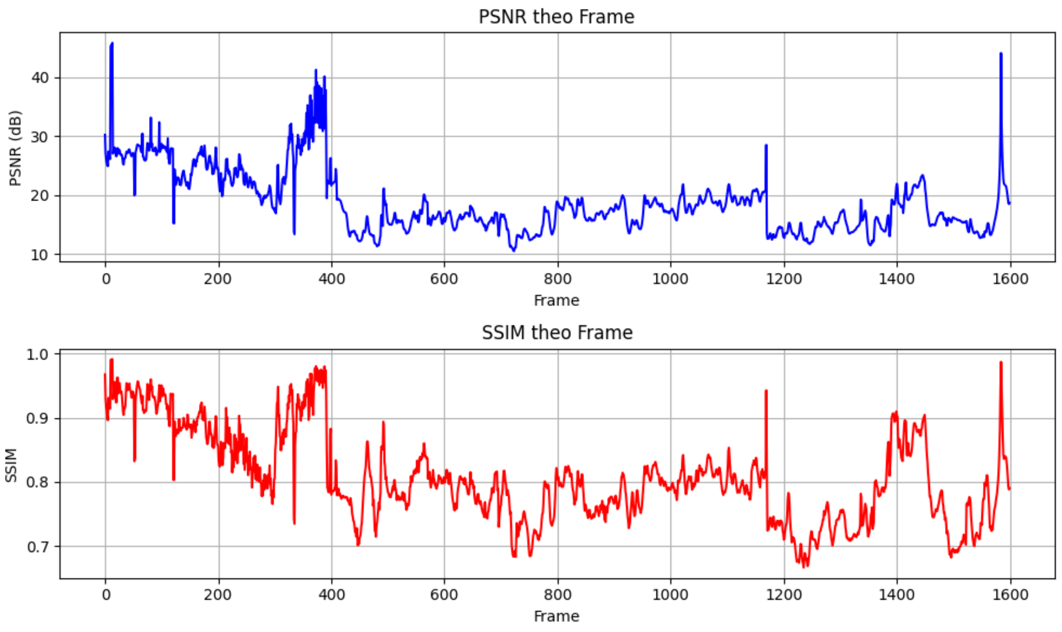
   The average SSIM values range from 0.7483 to 0.8379 across the tested bitrates (500k to 2500k), with the highest value observed at 2000k with a 1% discarding rate (0.8379) and the lowest at 2500k with a 2% discarding rate (0.7483). This indicates that while higher bitrates generally improve structural similarity, excessive packet loss (e.g., 2% at 2500k) significantly degrades video quality, likely due to the loss of critical data such as keyframes. The SSIM values for the Y channel (luminance) are consistently lower than those for the U and V channels (chrominance), suggesting that luminance distortions are more pronounced, which aligns with the codec's performance under varying network conditions.

2.5.2. Visualization

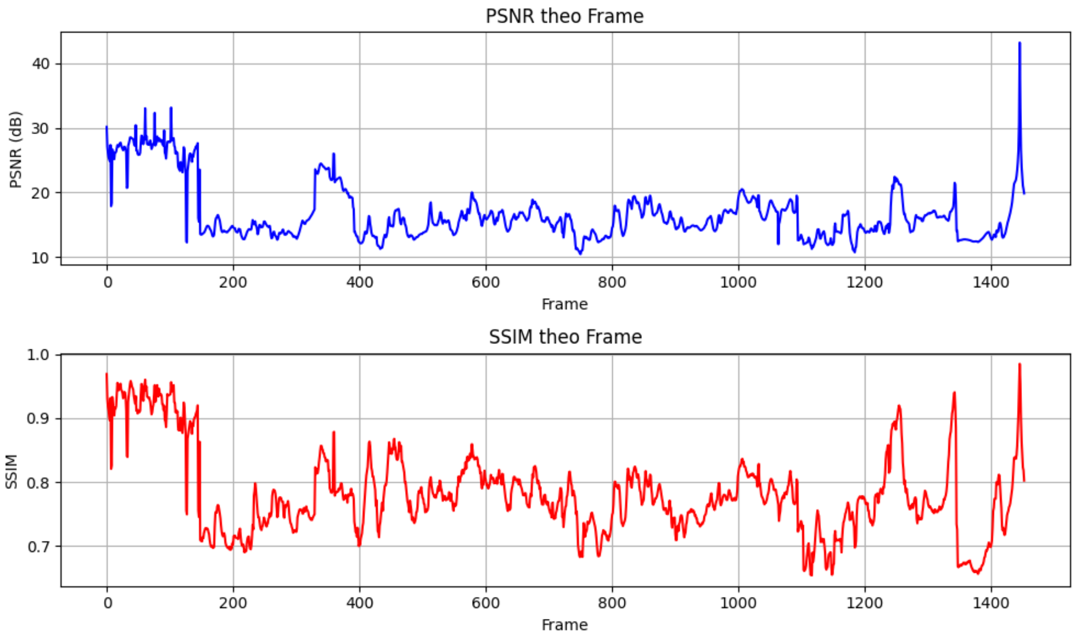
1. 500k - 0.1%



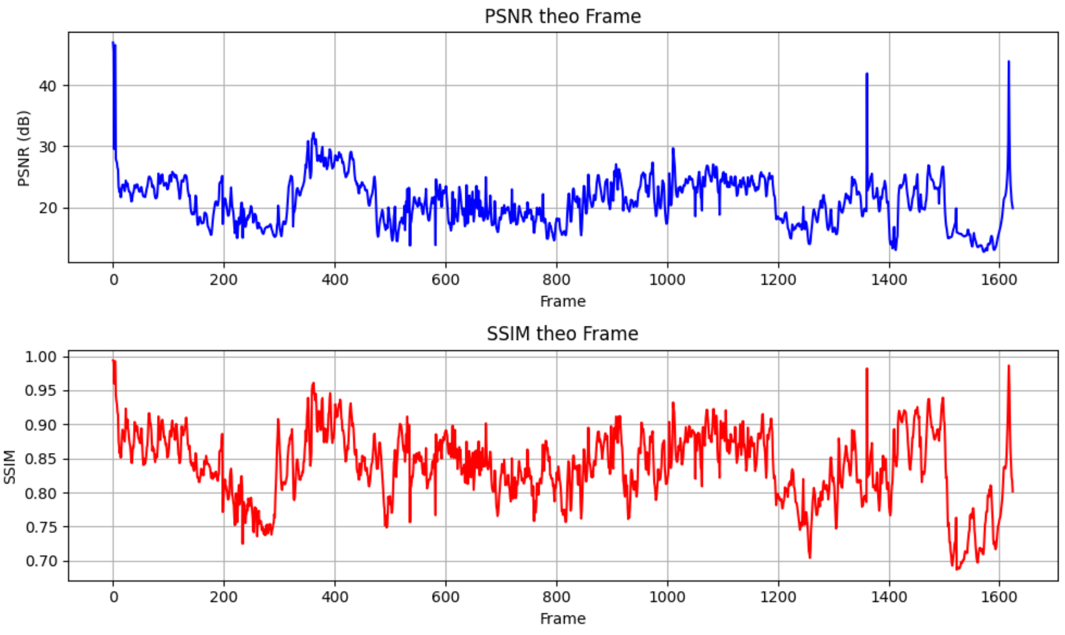
2. 1000k - 0.3%



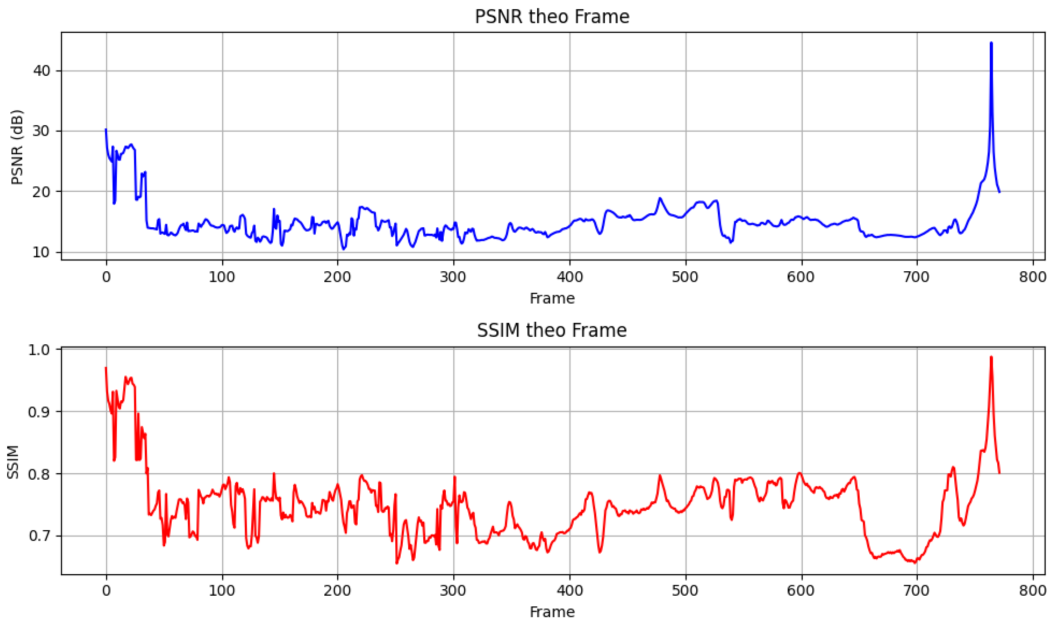
3. 1500K - 0.5%



4. 2000k - 1%



4. 2500k - 2%



III. Conclusion

The evaluation of streaming performance using libopenh264 and libopus codecs demonstrates that bitrate and packet loss critically influence multimedia quality. The configuration at 2000 kbps with a 1% discarding rate achieves optimal video quality (PSNR: 20.95 dB, SSIM: 0.8379), while 2500 kbps with 2% packet loss yields the lowest quality (PSNR: 14.91 dB, SSIM: 0.7483). The libopus codec at 192 kbps ensures consistent audio excellence. Discarding rates of 0.1%–2% reflect realistic network conditions, but rates above 1% degrade quality. For robust real-time streaming, bitrates near 2000 kbps, discarding rates below 1%, and codec optimizations are recommended. However there are still some exceptions considering higher bitrates like 3000k or 9000k, while they achieve higher video structure, the noise appears more due to connection and distance.