HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY

**SCHOOL OF ELECTRICAL AND ELECTRONIC ENGINEERING**



**PROJECT REPORT**

**Assignment 2**

Codec: ProRes/WavPack

Name of students: *Class code : 157320 –Group 3*

Mai Việt Anh – 20233827

Nguyễn Quang Dũng – 20233843

Hà Trung Dũng – 20233841

Trần Phú Nghĩa – 20233871

Phạm Nguyễn Tùng An - 20233825

Name of instructor: Dr. Pham Van Tien.

Hà Nội, 4/2025

# 

| **Member** | **Task** | **Completeness** |
| --- | --- | --- |
| Mai Việt Anh | Write a program to randomly discard video packets | 100% |
| Hà Trung Dũng | Communication between computers and Encoding video | 100% |
| Nguyễn Quang Dũng | Bitrate and Loss Rate Experiment | 100% |
| Trần Phú Nghĩa | Quality Assessment with PSNR and SSIM | 100% |
| Phạm Nguyễn Tùng An | Dynamic Graphic Overlay | 100% |

# 

# 

**Problem**

Work in group of no more than 5 members. Record a video clip of at least 3 minutes long while you are moving, during which your face show up clearly speak out your full name and student ID.

Use FFMPEG and VLC or Linphone to communicate the video from a computer to another one via Wi-Fi connection. Apply the bit rate control function at the sender to limit the data out.

Additionally, you must carry out the following jobs:

Write a program to randomly discard video packets from time to time at a given rate;

Create dynamic graphic contents (blocks, objects, animation, etc) to overlay on the transmitted video. Names of A/V codecs, members should also appear.

Repeat the experiment at least 5 times with different values of encoding bit rate and discarding rate so that the received videos clearly vary.

Assess the quality of the reconstructed videos measuring PSNR and SSIM.

1. **Introduction to ProRes & WavPack codec**
2. **ProRes codec:**

Apple ProRes, developed by Apple, is a high-quality, lossy video compression format used mainly in professional video editing. It supports resolutions up to 8K and is designed for excellent image quality and performance, making it ideal for post-production. Its ability to maintain visual fidelity while allowing bitrate control makes it suitable for transmission projects.

1. **WavPack codec:**

WavPack is an open-source audio codec offering both lossless and lossy compression, with a unique hybrid mode that creates a lossy file and a correction file for lossless restoration. It supports various audio formats and is known for efficiency, making it a flexible choice for controlling audio bitrate in experiments.

**II. Dynamic Graphic Overlay**

**Assigned Member:** Phạm Nguyễn Tùng An - 20233825

**Description:**

This task involves creating dynamic graphical content to overlay on the transmitted video.

**Methodology:**

* Generate animation frames using *pygame*, including dynamic objects (shapes, text) and scrolling member names.
* Convert the sequence of PNG frames into a transparent overlay video using the *ffmpeg* encoder with ProRes 4444.
* Overlaythe transparent animation video onto the original video using *ffmpeg*, aligning both resolution and timing precisely.

**Scrift:**

* generate\_frame.py

**import os**

**import pygame**

**import math**

**WIDTH, HEIGHT = 1280, 720**

**FPS = 30**

**DURATION = 15**

**TOTAL\_FRAMES = FPS \* DURATION**

**FONT\_SIZE = 48**

**members = [**

**"Mai Viet Anh-20233827",**

**"Pham Nguyen Tung An-20233825",**

**"Nguyen Quang Dung-20233843",**

**"Ha Trung Dung-20233841",**

**"Tran Phu Nghia-20233871"**

**]**

**code\_name = "Group 3-Code A/V:Wavpack/Prores"**

**pygame.init()**

**screen = pygame.Surface((WIDTH, HEIGHT), pygame.SRCALPHA)**

**font = pygame.font.SysFont("Arial", FONT\_SIZE)**

**os.makedirs("frames", exist\_ok=True)**

**# Khoảng cách cố định giữa các tên, đủ rộng để không dính nhau**

**padding\_per\_gap = 150**

**# Tạo text surfaces và lấy chiều rộng từng tên**

**member\_texts = [font.render(name, True, (255, 255, 0)) for name in members]**

**text\_widths = [text.get\_width() for text in member\_texts]**

**n = len(members)**

**total\_names\_width = sum(text\_widths)**

**# Tính chiều dài chu kỳ chạy đủ dài, đảm bảo có khoảng cách lớn hơn màn hình**

**cycle\_length\_members = total\_names\_width + padding\_per\_gap \* (n - 1) + WIDTH**

**speed\_members = cycle\_length\_members / DURATION # pixels per second**

**# Tính vị trí bắt đầu từng tên thành viên trong chu kỳ, theo thứ tự đều nhau**

**positions = [0]**

**for i in range(1, n):**

**# Vị trí tiếp theo = vị trí trước + chiều rộng tên trước + padding cố định**

**positions.append(positions[i-1] + text\_widths[i-1] + padding\_per\_gap)**

**# Phần code name chạy ngược lại, từ trái sang phải**

**code\_text = font.render(code\_name, True, (255, 255, 255))**

**code\_text\_width = code\_text.get\_width()**

**cycle\_length\_code = WIDTH + code\_text\_width**

**speed\_code = cycle\_length\_code / DURATION**

**for i in range(TOTAL\_FRAMES):**

**t = i / FPS**

**screen.fill((0, 0, 0, 0))**

**# Vẽ từng tên thành viên chạy từ ngoài màn hình bên phải vào**

**y\_members = 20**

**for idx, text in enumerate(member\_texts):**

**x = WIDTH + positions[idx] - (t \* speed\_members) % cycle\_length\_members**

**screen.blit(text, (x, y\_members))**

**# Vẽ code name chạy ngược chiều từ trái sang phải, bắt đầu ngoài màn hình trái**

**x\_code = -code\_text\_width + (t \* speed\_code) % cycle\_length\_code**

**y\_code = HEIGHT - 65**

**screen.blit(code\_text, (x\_code, y\_code))**

**# --- Hình tròn scale lớn nhỏ ---**

**circle\_surf = pygame.Surface((80, 80), pygame.SRCALPHA)**

**pygame.draw.circle(circle\_surf, (0, 255, 0, 180), (40, 40), 30)**

**scale\_factor = 1 + 0.5 \* math.sin(t \* 2 \* math.pi)**

**new\_size = int(80 \* scale\_factor)**

**circle\_surf = pygame.transform.smoothscale(circle\_surf, (new\_size, new\_size))**

**x1 = int((t \* WIDTH / DURATION) % WIDTH)**

**y1 = HEIGHT // 2 - 60**

**rect = circle\_surf.get\_rect(center=(x1, y1))**

**screen.blit(circle\_surf, rect)**

**# --- Hình vuông xoay 3 vòng ---**

**square\_surf = pygame.Surface((60, 60), pygame.SRCALPHA)**

**pygame.draw.rect(square\_surf, (255, 0, 0, 180), (10, 10, 40, 40))**

**angle2 = -(t \* 360 \* 3 / DURATION) % 360**

**square\_surf = pygame.transform.rotate(square\_surf, angle2)**

**x2 = WIDTH - int((t \* WIDTH / DURATION) % WIDTH)**

**y2 = HEIGHT // 2**

**rect2 = square\_surf.get\_rect(center=(x2, y2))**

**screen.blit(square\_surf, rect2)**

**# --- Hình tam giác xoay 4 vòng ---**

**triangle\_surf = pygame.Surface((80, 80), pygame.SRCALPHA)**

**triangle\_points = [(40, 10), (10, 70), (70, 70)]**

**pygame.draw.polygon(triangle\_surf, (0, 0, 255, 180), triangle\_points)**

**angle3 = (t \* 360 \* 4 / DURATION) % 360**

**triangle\_surf = pygame.transform.rotate(triangle\_surf, angle3)**

**x3 = WIDTH // 2**

**y3 = int((t \* HEIGHT / DURATION) % HEIGHT)**

**rect3 = triangle\_surf.get\_rect(center=(x3, y3))**

**screen.blit(triangle\_surf, rect3)**

**# Lưu frame**

**pygame.image.save(screen, f"frames/frame\_{i:04d}.png")**

### **\*Explanation:**

### *1. Initialization and Configuration*

Purpose: Set up basic video parameters.

* Import libraries: pygame, os, math.
* Define screen resolution, FPS, duration, font size.  
  Declare team member list and group code name.

### *2. Pygame Setup and Text Preprocessing*

Purpose: Initialize Pygame and prepare texts for display.

* Initialize font and screen surface, create folder for saving frames.
* Create text surfaces for each member with fixed spacing.
* Calculate initial positions and movement speed of member names.
* Prepare the group code name to move in the opposite direction.

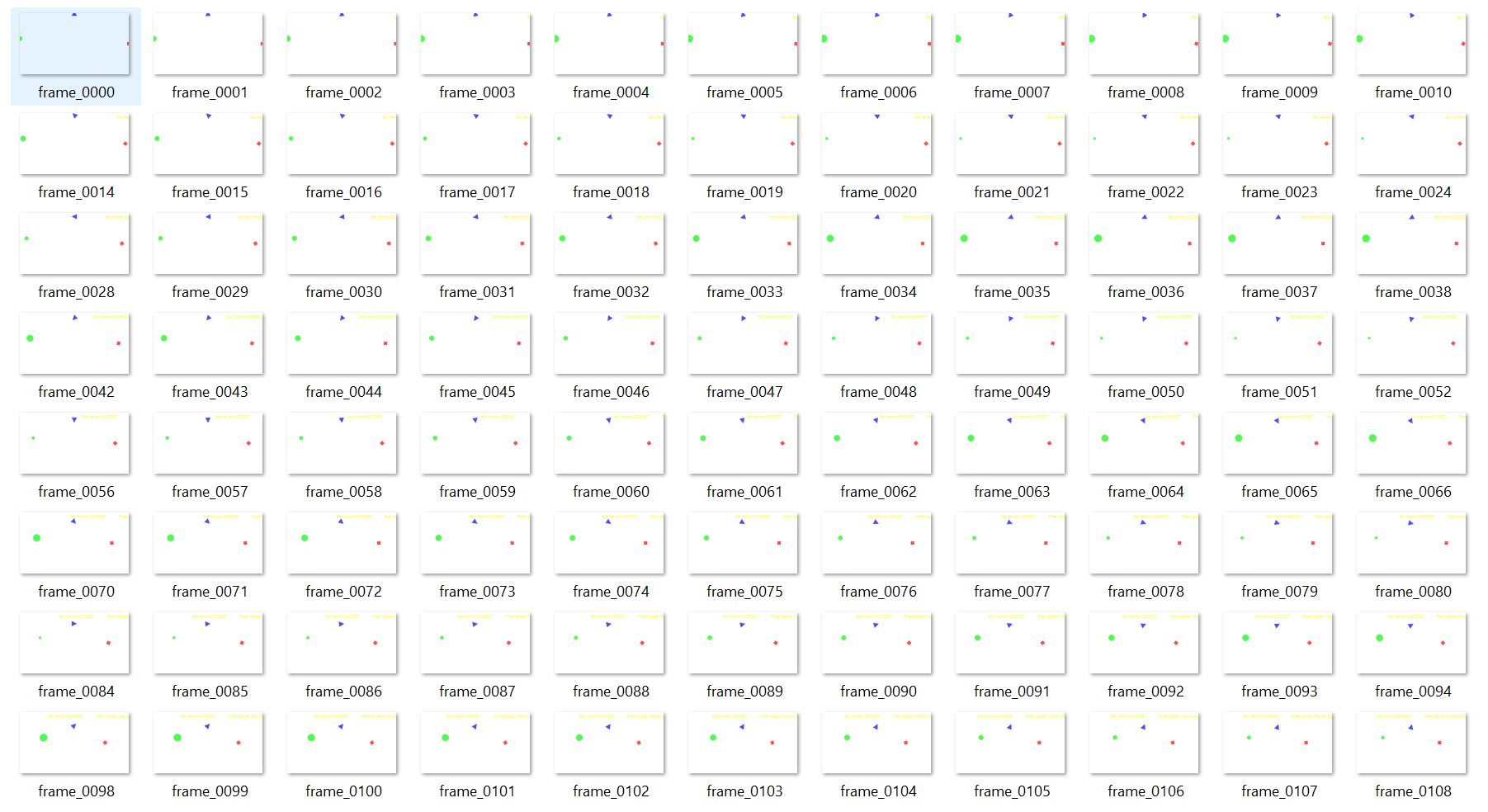
### *3. Frame Rendering*

Purpose: Draw visual effects for each frame and save as image.

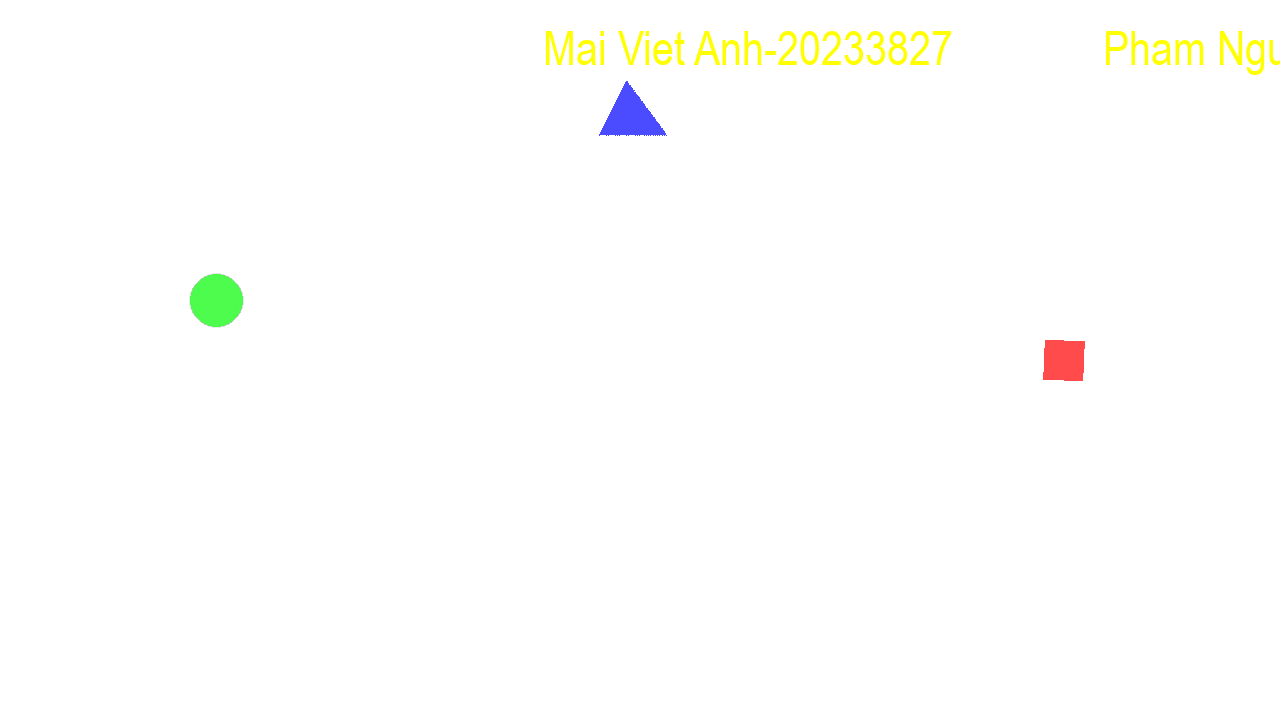
For each frame:

* Clear the screen.
* Draw member names scrolling right to left at the top.
* Draw the group code name scrolling left to right at the bottom.
* Draw a scaling circle, a rotating square, and a rotating triangle moving vertically.
* Save the frame as a PNG image.

**\*Result:**

****

*Folder frame*

**

*Overlay sample*

**Script:** overlay.py

**import subprocess**

**# Thông số**

**frame\_pattern = "frames/frame\_%04d.png"**

**fps = 30**

**overlay\_video = "overlay\_video.mov"**

**input\_video = "input.mp4"**

**output\_video = "output.mp4"**

**def pngs\_to\_prores\_with\_alpha():**

**cmd = [**

**"ffmpeg",**

**"-y",**

**"-framerate", str(fps),**

**"-i", frame\_pattern,**

**"-c:v", "prores\_ks",**

**"-profile:v", "4444",**

**"-pix\_fmt", "yuva444p10le",**

**overlay\_video**

**]**

**subprocess.run(cmd, check=True)**

**print("Đã xuất video overlay ProRes với alpha:", overlay\_video)**

**def overlay\_on\_video():**

**cmd = [**

**"ffmpeg",**

**"-y",**

**"-stream\_loop", "-1", # lặp overlay vô hạn**

**"-i", overlay\_video,**

**"-i", input\_video,**

**"-filter\_complex", "[1:v][0:v]overlay=0:0:format=auto",**

**"-shortest", # cắt theo độ dài video gốc**

**"-c:a", "copy",**

**output\_video**

**]**

**subprocess.run(cmd, check=True)**

**print("Đã tạo video đầu ra với overlay:", output\_video)**

**if \_\_name\_\_ == "\_\_main\_\_":**

**pngs\_to\_prores\_with\_alpha()**

**overlay\_on\_video()**

**\*Explanation:**

*1.Convert PNG frames to a ProRes video with alpha channel*

Uses ffmpeg to:

* Read a sequence of PNG frames (e.g., frame\_0000.png, frame\_0001.png, …).
* Set the frame rate using -framerate.
* Encode the video using ProRes 4444 (-c:v prores\_ks -profile:v 4444), which supports alpha transparency.
* Use pixel format yuva444p10le to retain 10-bit color depth and an alpha channel.
* Output a .mov file that preserves the transparent background — ideal for overlaying on another video later.

### *2. Overlay the generated video onto a base video*

Uses ffmpeg to:

* Loop the overlay infinitely (-stream\_loop -1).
* Overlay it on the input video using overlay=0:0:format=auto.
* Trim to the base video’s length (-shortest).
* Copy audio from the base video.

**\*Result:**

* The script creates an overlay video in ProRes 4444 format because this codec supports alpha channel (transparency), which is essential for preserving the transparent parts of the overlay. (ProRes 4444 also provides high-quality video with minimal compression and is widely supported in professional workflows.)
* The final output video (output.mp4) overlays this transparent video on the original input video, preserving transparency and matching the original video length and audio.

*A screenshot of the video with overlay*

**III. Bitrate and Loss Rate Experiment**

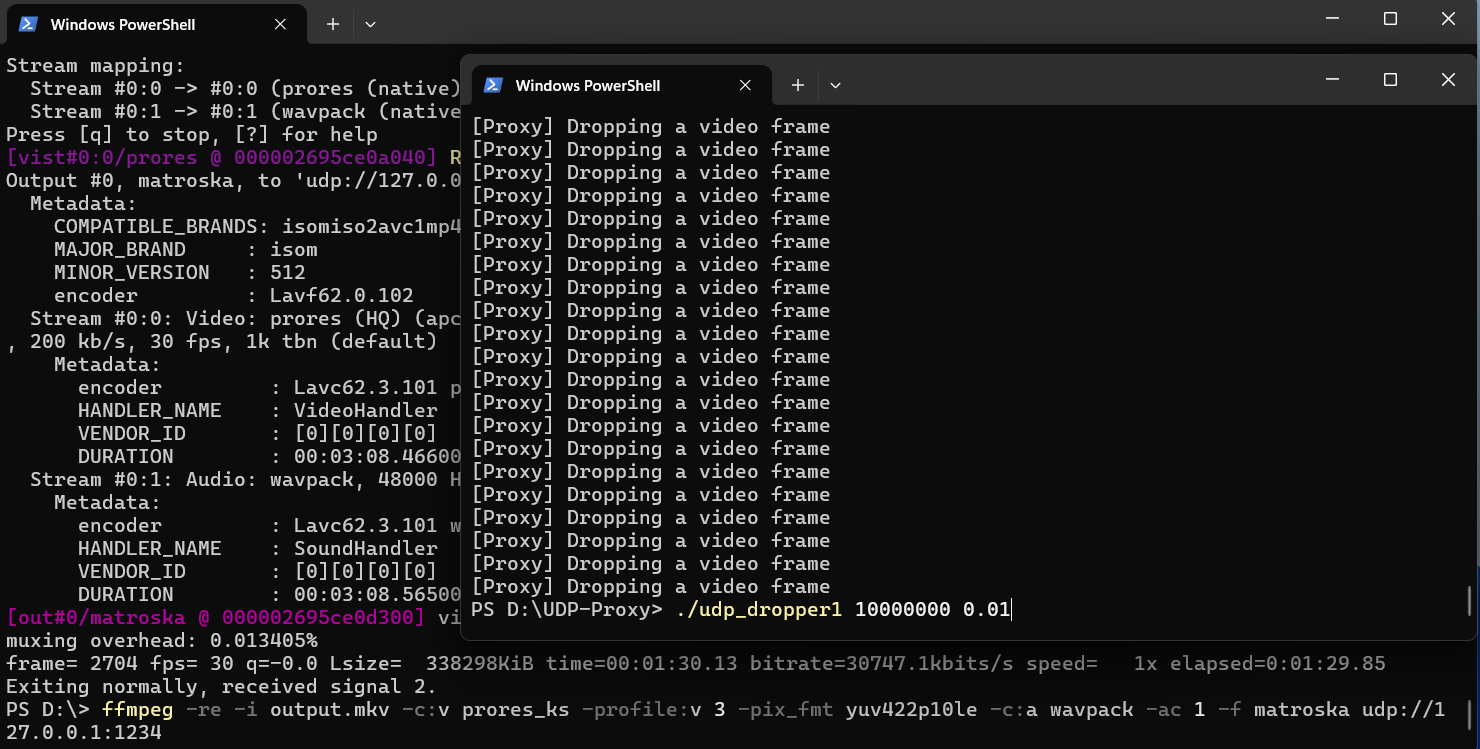
**Assigned Member:** Nguyễn Quang Dũng- 20233843

This section focuses on examining the impact of varying bitrate constraints and packet loss rates on the quality of video transmission over a simulated network environment. The aim is to assess how these two parameters—representing typical network limitations—affect the integrity and visual quality of the received video stream.

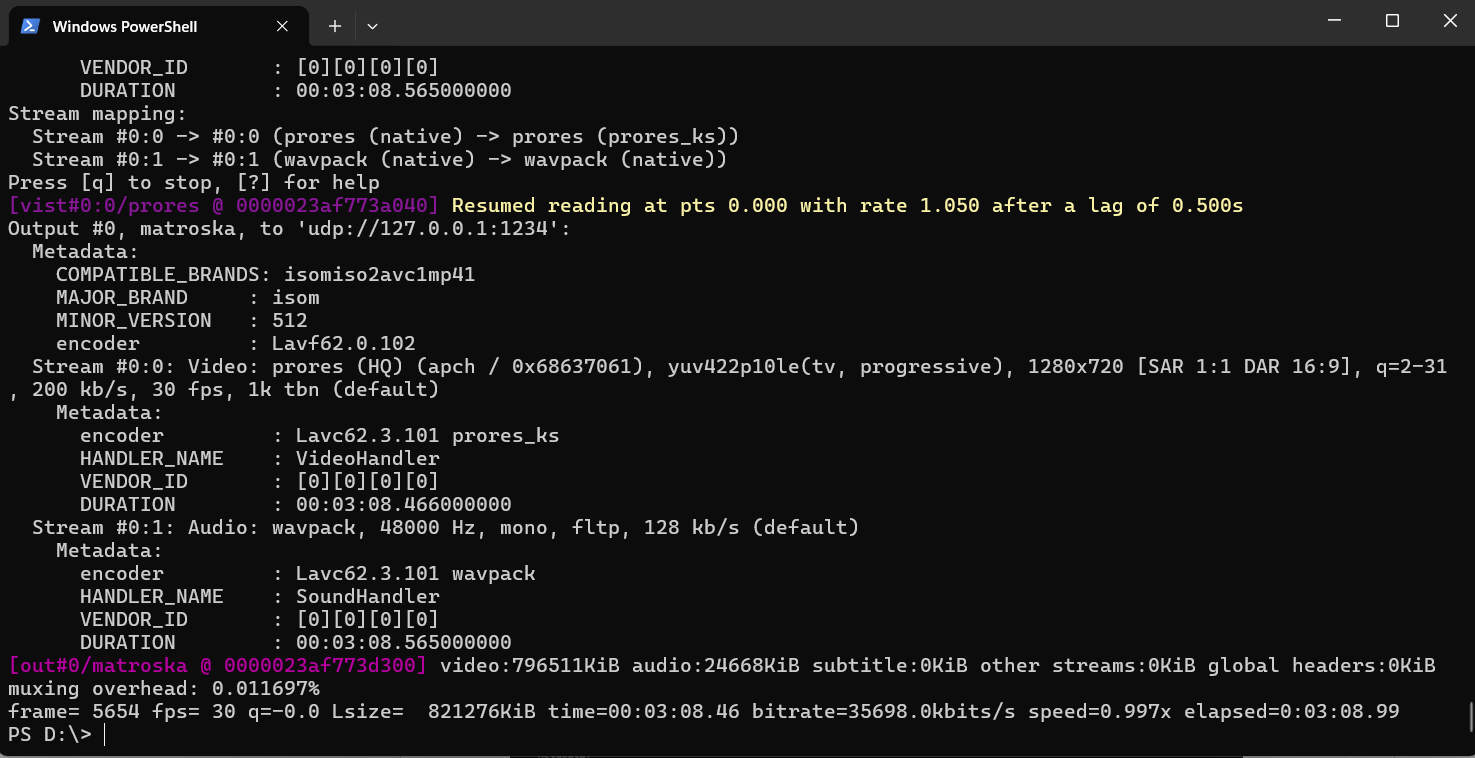
To carry out this experiment, a custom UDP proxy program named udp\_dropper was developed. This tool receives UDP video packets on a specified port, applies both a bitrate cap and probabilistic packet drops, and then forwards the modified stream to a different local port. Although the network simulation occurred on a single machine, the setup effectively emulates network behavior under bandwidth and reliability stress.

Five configurations were tested, each with a different target bitrate and packet drop rate. The commands used and their corresponding parameters are listed in the table below:

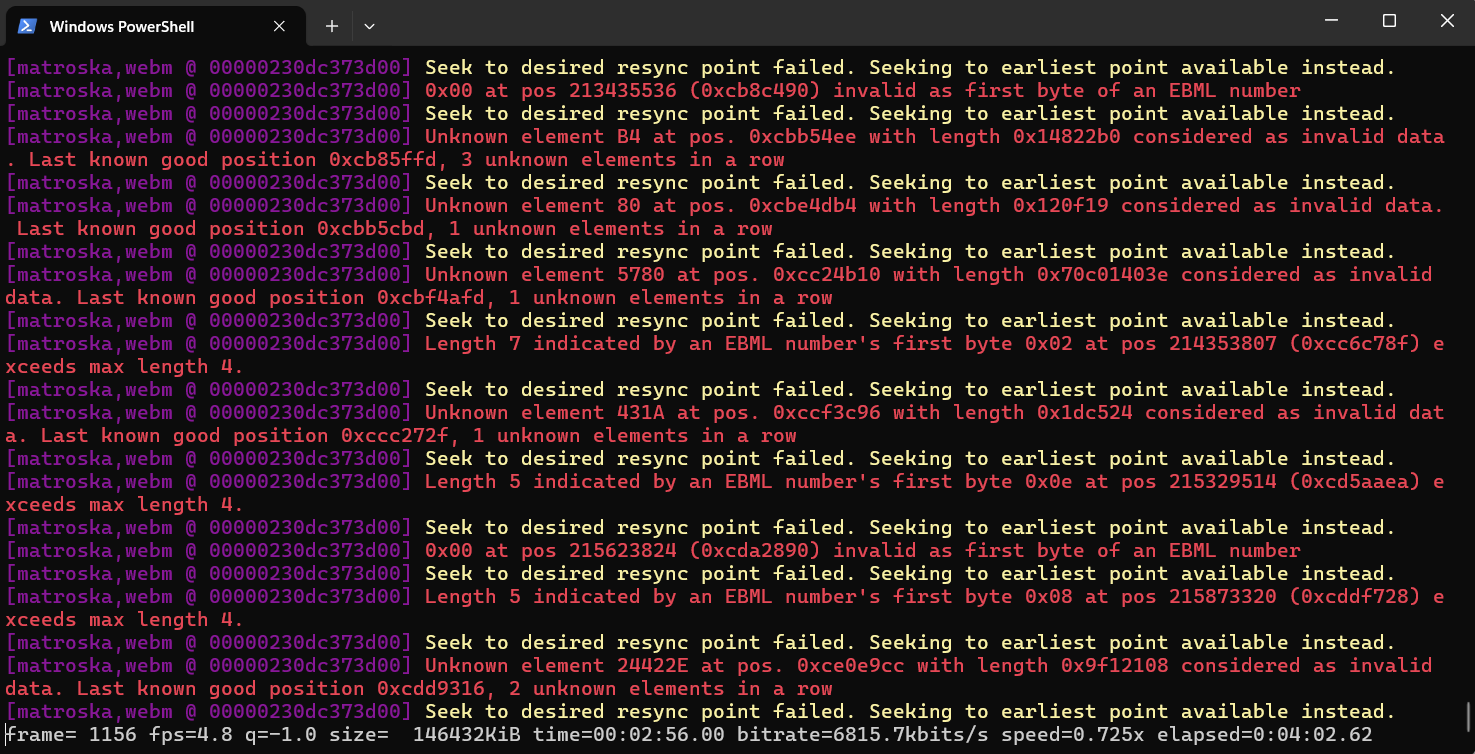
| Configuration | Bitrate (bps) | Drop rate | Command |
| --- | --- | --- | --- |
| 1 | 10,000,000 | 1% | ./udp\_dropper 10000000 0.01 |
| 2 | 20,000,000 | 4% | ./udp\_dropper 20000000 0.04 |
| 3 | 50,000,000 | 2% | ./udp\_dropper 50000000 0.02 |
| 4 | 100,000,000 | 1% | ./udp\_dropper 100000000 0.01 |
| 5 | 150,000,000 | 0.5% | ./udp\_dropper 150000000 0.005 |



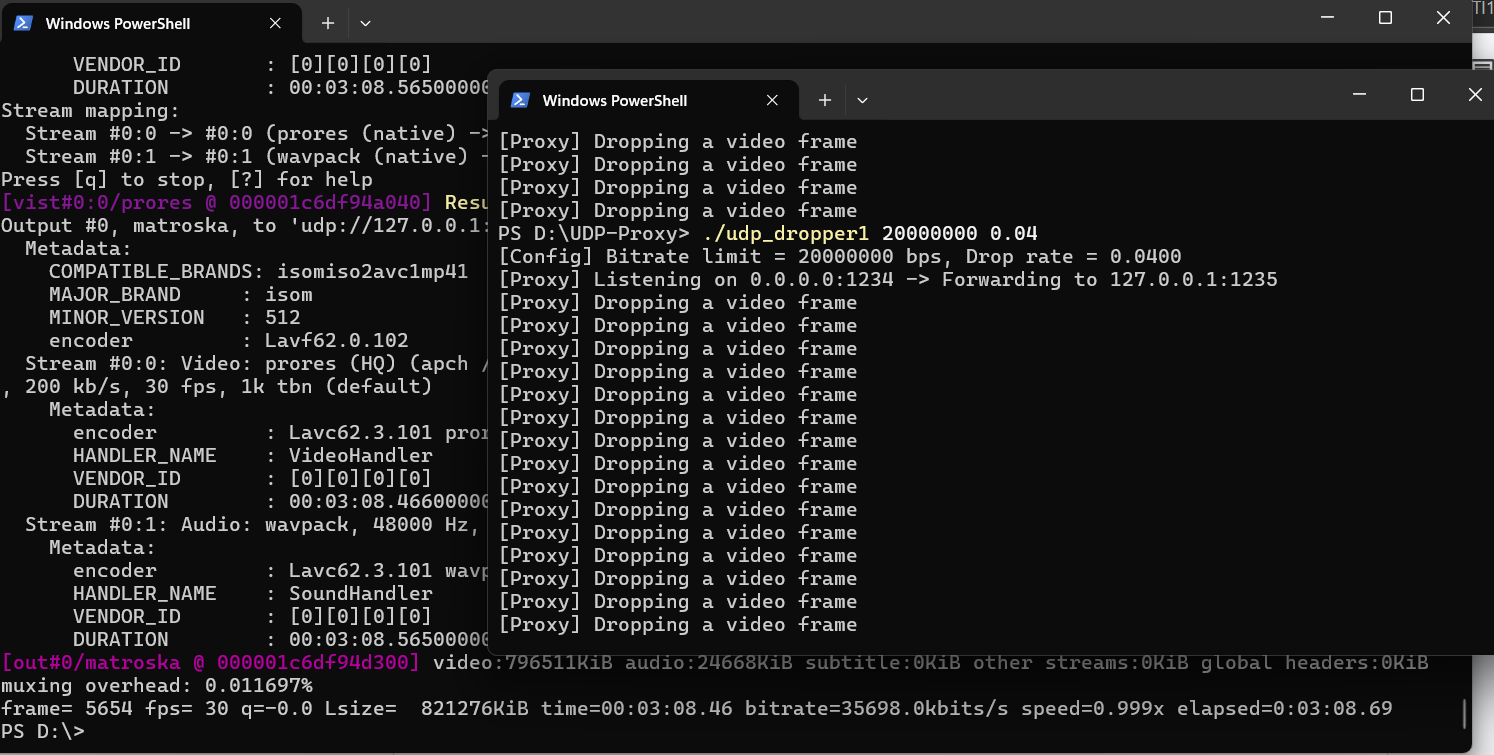
Initially, the video transmission process was configured to operate at a bitrate of 10,000,000 bits per second, accompanied by an intentional packet loss rate of 1%, simulating network instability conditions.



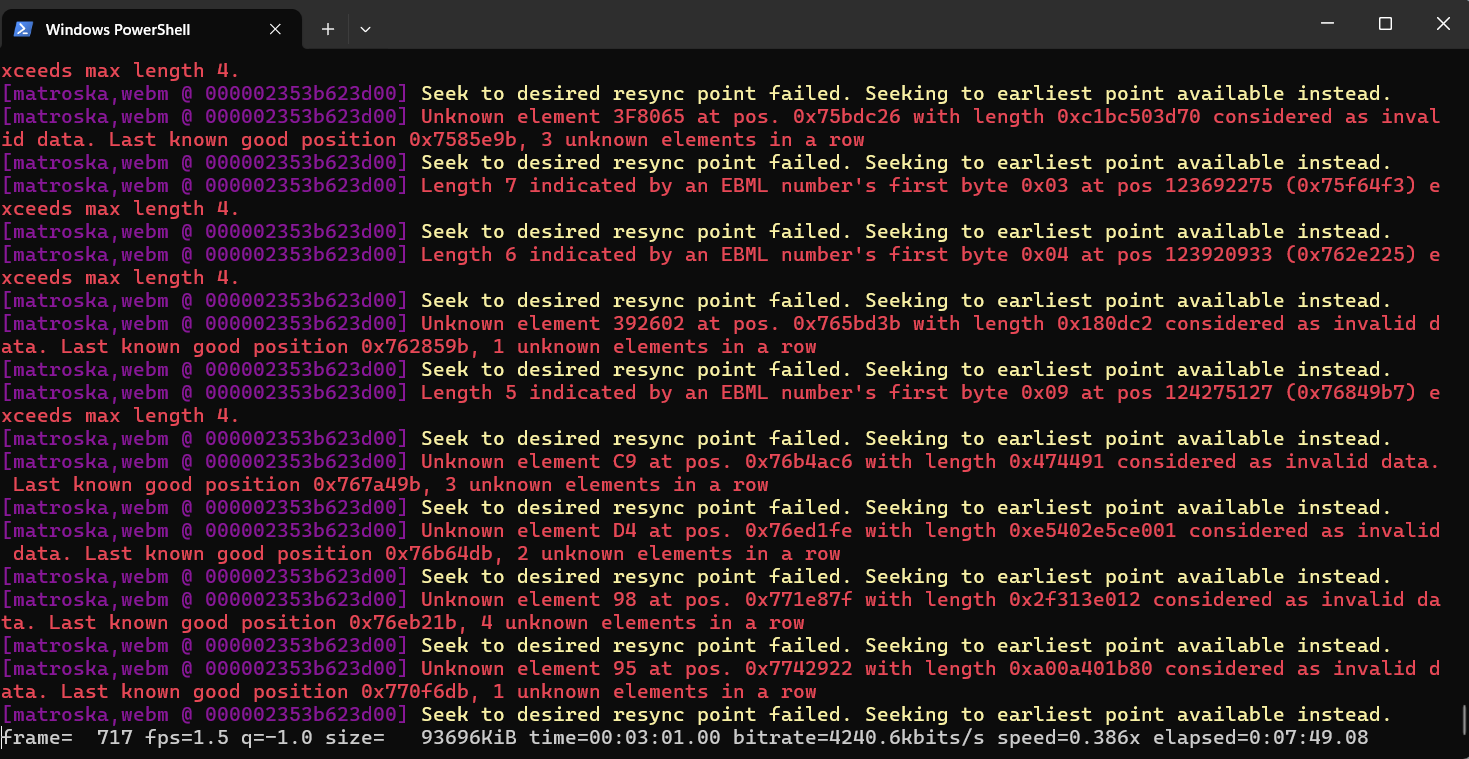
As observed in the output, the system successfully encoded and transmitted a total of 5654 video frames over the specified duration, suggesting that the source stream was functioning correctly without encoding interruptions.



However, due to the relatively high bitrate and simulated packet loss, the receiving end was only able to decode and reconstruct a maximum of 1156 frames, resulting in a significant degradation in playback quality and temporal continuity.



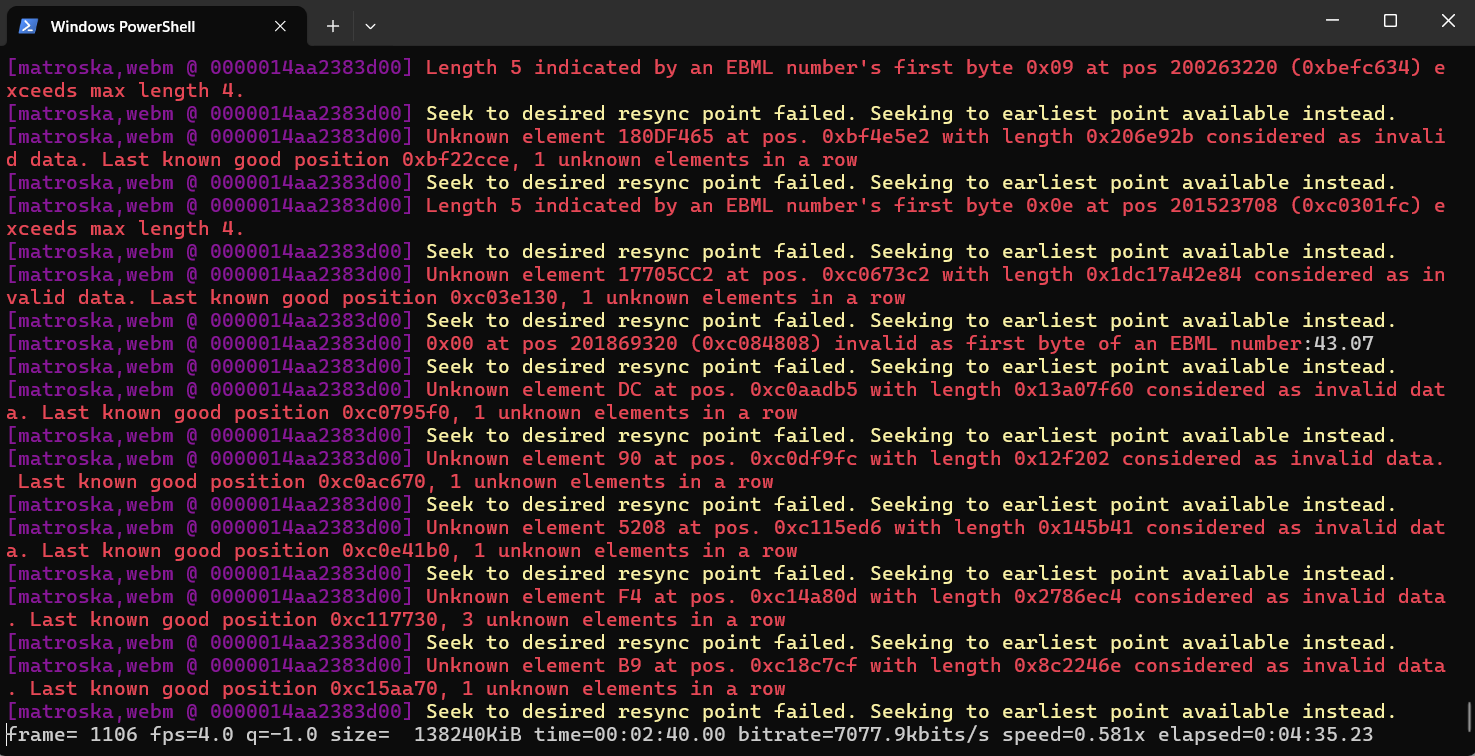
In the second experiment, the transmission rate was elevated to 20,000,000 bits per second while the simulated network conditions included a 4% packet loss rate, representing a notably unstable connection environment.



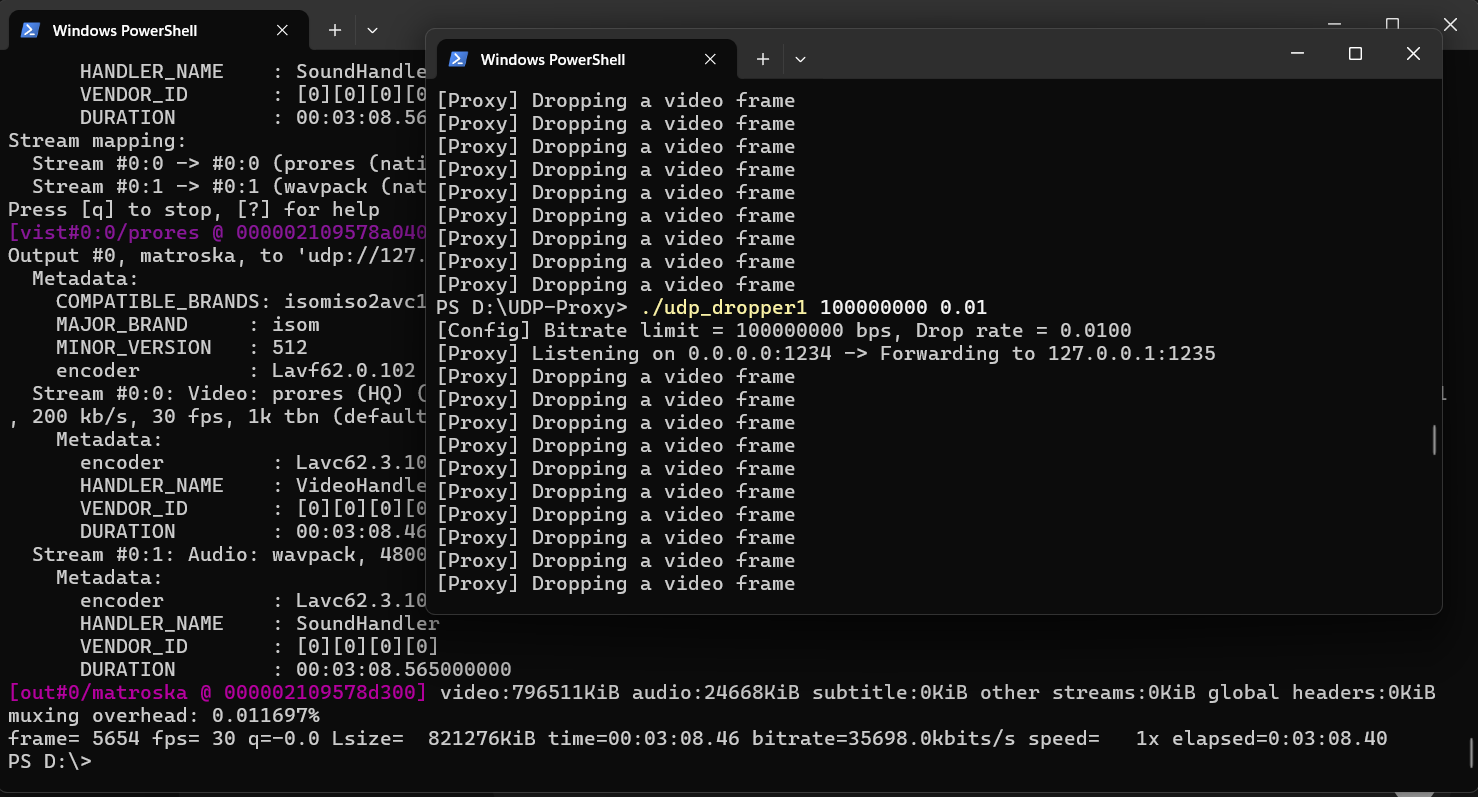
Despite the increased data throughput, the receiving system was only capable of recovering 717 frames, highlighting the detrimental effect of higher loss rates on video integrity, especially under dense transmission conditions.



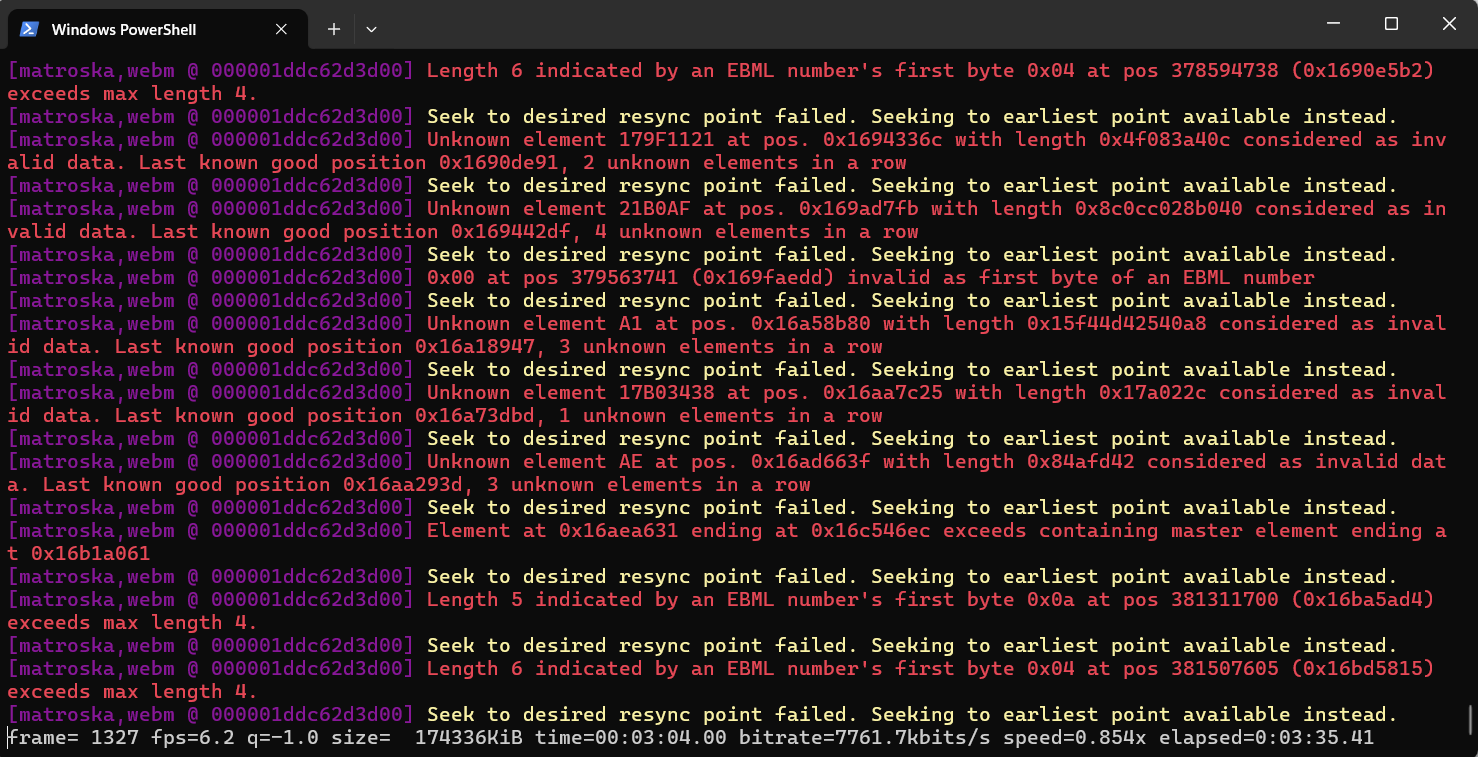
For the third test, a further increase in bandwidth was implemented, setting the bitrate at 50,000,000 bits per second with a reduced packet loss of 2%, aiming to observe improvements in received video quality under more moderate loss levels.



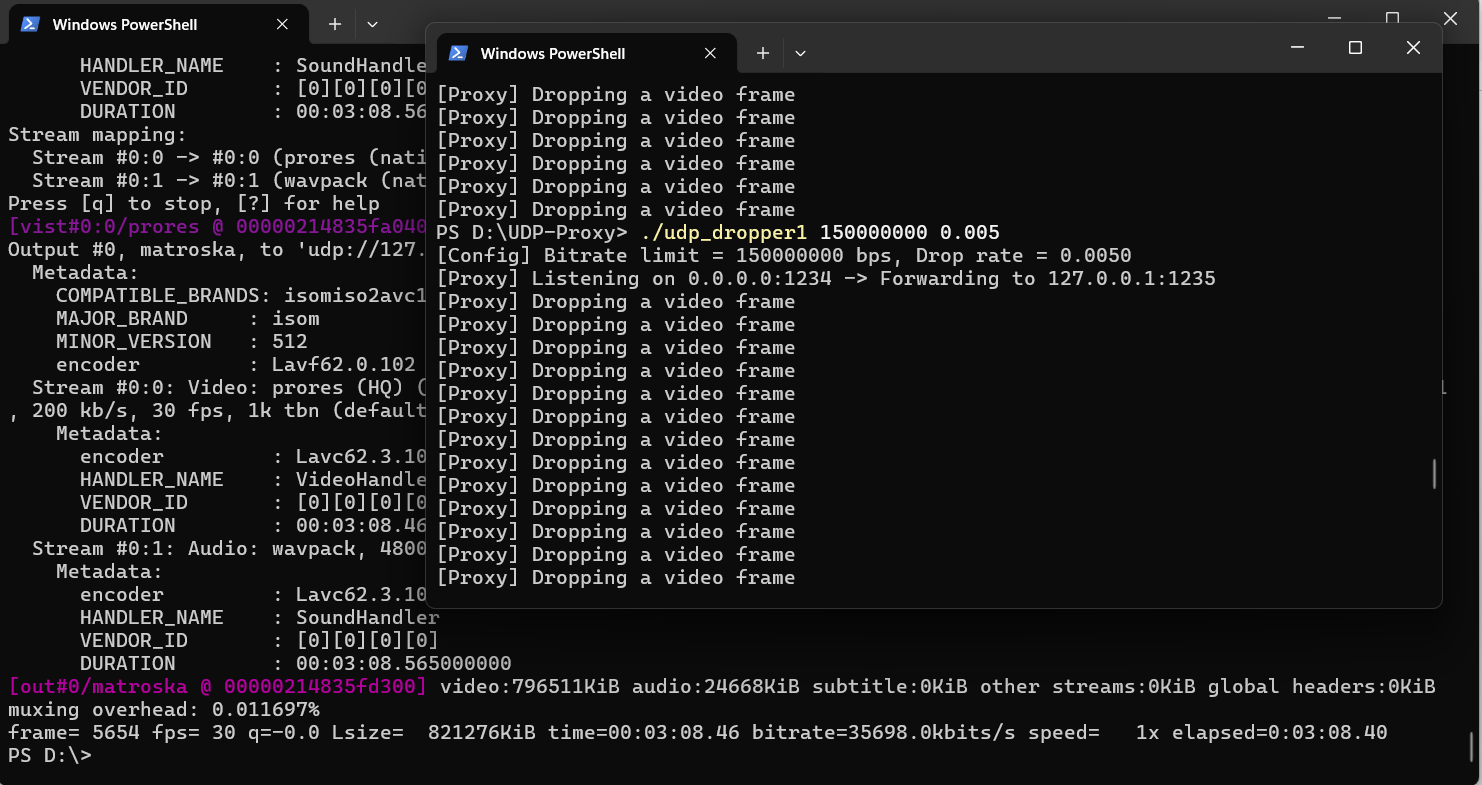
Under these conditions, a total of 1186 frames were successfully received, suggesting a partial recovery in data fidelity, although transmission imperfections still caused noticeable frame drops.



The fourth scenario involved pushing the bitrate even higher to 100,000,000 bits per second, coupled with a controlled packet loss of only 1%, simulating a relatively stable streaming environment with minimal interference.



This setup allowed the receiver to reconstruct 1327 frames, indicating a significant improvement in reliability and frame continuity as both the bitrate and network stability approached optimal thresholds.



In the final configuration, the system was tested under high-performance conditions, with a transmission rate of 150,000,000 bits per second and only 0.5% packet loss, closely resembling real-world high-speed, low-latency networks used in professional-grade streaming setups.

**IV. Packet Loss Simulation**

**Assigned Member:** Mai Việt Anh - 20233827

**Description:** Create a program to simulate unstable network conditions. In this program, I implemented a UDP proxy application that listens for UDP packets on a specified local address and port, then forwards those packets to a designated destination address and port.

**Script: udp\_dropper.c**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <time.h>

// Include Windows socket libraries if compiling on Windows

#ifdef \_WIN32

#include <winsock2.h>

#include <ws2tcpip.h>

#include <windows.h>

#pragma comment(lib, "ws2\_32.lib")

// Implement gettimeofday for Windows (to mimic Linux behavior)

int gettimeofday(struct timeval\* tp, struct timezone\* tzp) {

FILETIME ft;

unsigned \_\_int64 tmpres = 0;

static int tzflag;

if (tp) {

GetSystemTimeAsFileTime(&ft);

tmpres |= ft.dwHighDateTime;

tmpres <<= 32;

tmpres |= ft.dwLowDateTime;

// Convert to microseconds since Unix epoch

tmpres /= 10; // to microseconds

tmpres -= 11644473600000000ULL; // Windows epoch to Unix epoch

tp->tv\_sec = (long)(tmpres / 1000000UL);

tp->tv\_usec = (long)(tmpres % 1000000UL);

}

return 0;

}

#else

// If Linux/Unix, include standard socket libraries

#include <unistd.h>

#include <arpa/inet.h>

#include <sys/socket.h>

#include <sys/time.h>

#endif

// Listen address and port

#define LISTEN\_IP "0.0.0.0"

#define LISTEN\_PORT 1234

// Forward address and port

#define FORWARD\_IP "127.0.0.1"

#define FORWARD\_PORT 1235

// Maximum UDP packet size

#define MAX\_UDP\_PACKET\_SIZE 65536

// Default drop rate (probability to drop a packet)

double drop\_rate = 0.05;

// Bitrate limit (bps), 0 = unlimited

unsigned long bitrate\_limit\_bps = 0;

int main(int argc, char\* argv[]) {

#ifdef \_WIN32

// Initialize Winsock for Windows

WSADATA wsaData;

if (WSAStartup(MAKEWORD(2, 2), &wsaData) != 0) {

fprintf(stderr, "WSAStartup failed\n");

return 1;

}

#endif

// Check command-line arguments, print usage if missing

if (argc < 2) {

printf("Usage: %s <bitrate\_bps> [drop\_rate]\n", argv[0]);

printf("Example: %s 5000000 0.05\n", argv[0]);

return 1;

}

// Read bitrate and drop\_rate from command line

bitrate\_limit\_bps = strtoul(argv[1], NULL, 10);

if (argc >= 3) {

drop\_rate = atof(argv[2]);

if (drop\_rate < 0) drop\_rate = 0;

if (drop\_rate > 1) drop\_rate = 1;

}

// Print the configuration

printf("[Config] Bitrate limit = %lu bps, Drop rate = %.4f\n", bitrate\_limit\_bps, drop\_rate);

// Initialize random number generator for packet dropping

srand((unsigned int)time(NULL));

// Declare input and output sockets and address structures

int sock\_in, sock\_out;

struct sockaddr\_in addr\_in, addr\_out;

// Create UDP input socket

sock\_in = socket(AF\_INET, SOCK\_DGRAM, 0);

if (sock\_in < 0) {

perror("Failed to create input socket");

return 1;

}

// Increase receive buffer to 4MB to avoid packet loss under heavy load

int rcvbuf\_size = 4 \* 1024 \* 1024;

setsockopt(sock\_in, SOL\_SOCKET, SO\_RCVBUF, (char\*)&rcvbuf\_size, sizeof(rcvbuf\_size));

// Set up input address information

memset(&addr\_in, 0, sizeof(addr\_in));

addr\_in.sin\_family = AF\_INET;

addr\_in.sin\_addr.s\_addr = inet\_addr(LISTEN\_IP);

addr\_in.sin\_port = htons(LISTEN\_PORT);

// Bind input socket to address/port

if (bind(sock\_in, (struct sockaddr\*)&addr\_in, sizeof(addr\_in)) < 0) {

perror("Bind failed");

#ifdef \_WIN32

closesocket(sock\_in);

WSACleanup();

#else

close(sock\_in);

#endif

return 1;

}

// Create UDP output socket

sock\_out = socket(AF\_INET, SOCK\_DGRAM, 0);

if (sock\_out < 0) {

perror("Failed to create output socket");

#ifdef \_WIN32

closesocket(sock\_in);

WSACleanup();

#else

close(sock\_in);

#endif

return 1;

}

// Increase send buffer to 4MB to avoid congestion when sending a lot

int sndbuf\_size = 4 \* 1024 \* 1024;

setsockopt(sock\_out, SOL\_SOCKET, SO\_SNDBUF, (char\*)&sndbuf\_size, sizeof(sndbuf\_size));

// Set up output (forward) address information

memset(&addr\_out, 0, sizeof(addr\_out));

addr\_out.sin\_family = AF\_INET;

addr\_out.sin\_addr.s\_addr = inet\_addr(FORWARD\_IP);

addr\_out.sin\_port = htons(FORWARD\_PORT);

// Print proxy information

printf("[Proxy] Listening on %s:%d -> Forwarding to %s:%d\n", LISTEN\_IP, LISTEN\_PORT, FORWARD\_IP, FORWARD\_PORT);

// Buffer for receiving/sending UDP data

unsigned char buffer[MAX\_UDP\_PACKET\_SIZE];

// Statistics variables for received, sent, dropped packets and bytes sent in a period

unsigned long long packets\_received = 0;

unsigned long long packets\_sent = 0;

unsigned long long packets\_dropped = 0;

unsigned long long bytes\_sent\_this\_period = 0;

// Time variables for bitrate calculation

struct timeval start\_time, current\_time;

gettimeofday(&start\_time, NULL);

// Main loop to receive and process packets

while (1) {

// Receive UDP packet from input socket

int len = recvfrom(sock\_in, (char\*)buffer, sizeof(buffer), 0, NULL, NULL);

if (len < 0) {

perror("recvfrom failed");

break;

}

packets\_received++;

// Randomly drop packet according to drop\_rate

double rnd = (double)rand() / RAND\_MAX;

if (rnd < drop\_rate) {

packets\_dropped++;

continue;

}

// Bitrate limiting: check total bits sent in 1 second

gettimeofday(&current\_time, NULL);

long elapsed\_usec = (current\_time.tv\_sec - start\_time.tv\_sec) \* 1000000L + (current\_time.tv\_usec - start\_time.tv\_usec);

if (elapsed\_usec >= 1000000L) {

// Reset every second

bytes\_sent\_this\_period = 0;

start\_time = current\_time;

}

if (bitrate\_limit\_bps > 0) {

unsigned long bits\_to\_send = (unsigned long)len \* 8;

if ((bytes\_sent\_this\_period \* 8 + bits\_to\_send) > bitrate\_limit\_bps) {

// If bitrate limit exceeded, drop packet

packets\_dropped++;

continue;

}

}

// Send packet to forward address

int sent\_len = sendto(sock\_out, (char\*)buffer, len, 0, (struct sockaddr\*)&addr\_out, sizeof(addr\_out));

if (sent\_len < 0) {

perror("sendto failed");

break;

}

packets\_sent++;

bytes\_sent\_this\_period += sent\_len;

// Print log every 5000 packets received

if (packets\_received % 5000 == 0) {

printf("[Stats] Received: %llu, Sent: %llu, Dropped: %llu, Current bitrate: %.2f kbps\n",

packets\_received, packets\_sent, packets\_dropped, (bytes\_sent\_this\_period \* 8 / 1000.0));

}

}

// Close sockets and clean up resources when done

#ifdef \_WIN32

closesocket(sock\_in);

closesocket(sock\_out);

WSACleanup();

#else

close(sock\_in);

close(sock\_out);

#endif

return 0;

}

1. Initialization and Configuration

* **Import libraries**: Include standard C libraries and network libraries for both Windows and Linux.
* **Define constants**: Listening/sending address and port, maximum UDP packet size, default drop rate, and default bitrate limit.

2. Network Environment Setup

* **Windows**: Call WSAStartup to initialize Winsock.
* **Linux**: No additional setup required.

3. Read Command-Line Parameters

* If insufficient parameters are provided, print usage instructions and exit.
* Read bitrate and drop rate values from the command line and validate them.

4. Initialize Receiving Socket

* Create a UDP socket for receiving data.
* Increase the receive buffer size to **4MB**.
* Configure the listening address and port.
* Call bind to attach the socket to the specified address/port.

5. Initialize Sending Socket

* Create a UDP socket for forwarding data.
* Increase the send buffer size to **4MB**.
* Configure the forwarding address and port.

6. Print Proxy Configuration

* Display the listening address/port and forwarding address/port.

7. Main Packet Processing Loop

* **Receive packet**: Use recvfrom to read incoming data.
* **Increment counter**: Track the total number of received packets.
* **Random drop**:
  + Generate a random number; if it’s below drop\_rate, skip the packet (increment dropped counter).
* **Bitrate throttling**:
  + Calculate the total bytes sent in the last second.
  + If the limit is exceeded, skip the packet (increment dropped).
  + Otherwise, proceed to forward.
* **Send packet**: Use send to to forward data to the target address.
* **Update stats**: Increment sent packets counter and add the bytes sent.
* **Logging**: Print statistics every **5000 received packets**.

8. Cleanup and Termination

* On error or loop exit, close all sockets and release resources (WSACleanup on Windows).

**V. Video Communication Between Computers And Encoding Video**

1:First we will encode the input file output.mp4 to a mkv file with 2 A/V codec: Wavpack/Prores

Using : ffmpeg -i output.mp4 -c:v prores\_ks -profile:v 0 -pix\_fmt yuv422p10le -c:a wavpack output.mkv





**‘ffmpeg’:** Runs the FFmpeg tool to convert or process multimedia files.

**‘-i output.mp4’:** Specifies the input file to be processed – in this case, output.mp4

**‘-c:v prores\_ks’:** Sets the video codec to prores\_ks, which is an encoder for Apple ProRes

**‘-profile:v 0’:** Chooses the ProRes proxy profile (lowest bitrate and quality among ProRes profiles). Profile levels:

• 0 – proxy

• 1 – LT

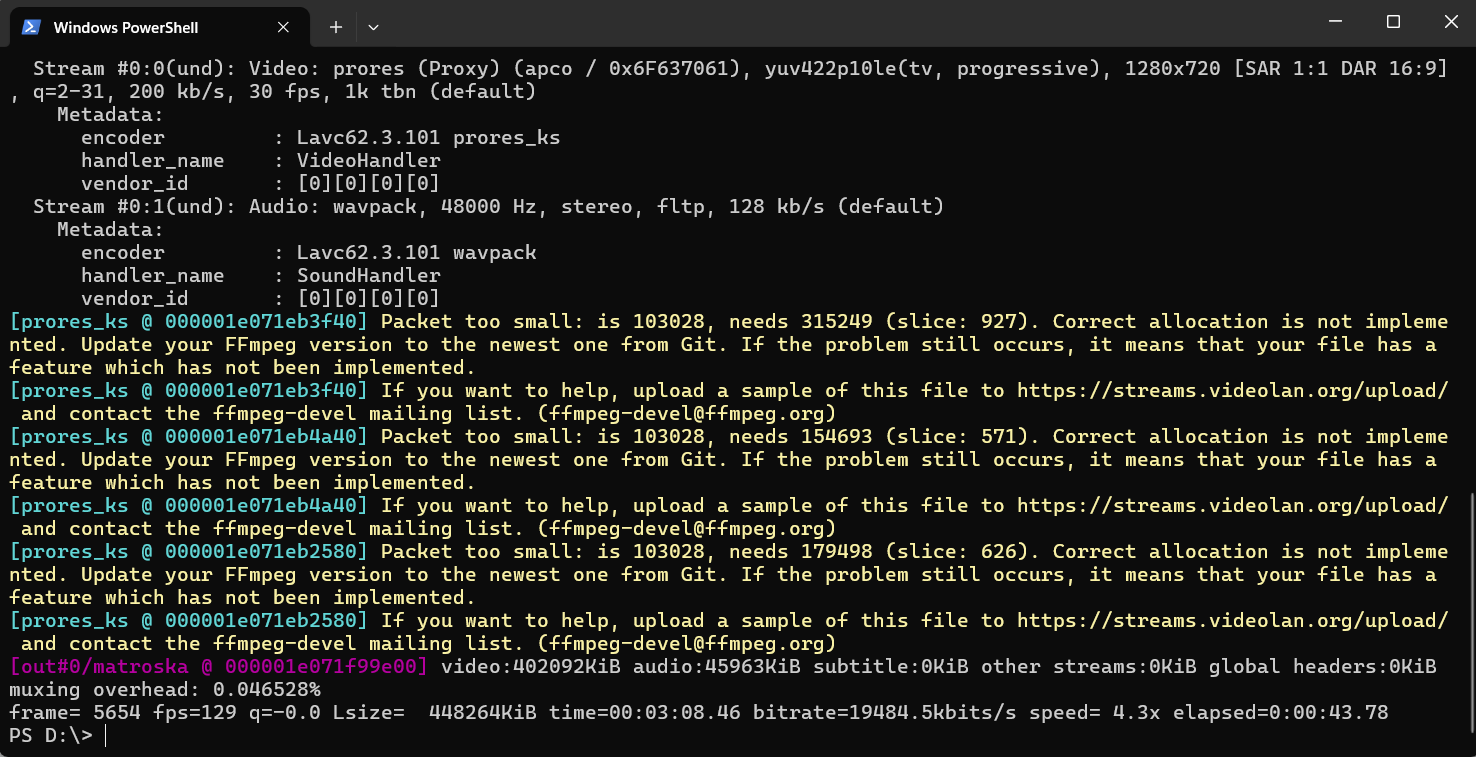
• 2 – standard

• 3 – HQ

**‘-pix\_fmt yuv422p10le’:** Sets the pixel format to 10-bit 4:2:2 planar YUV with little-endian data layout. Required for valid ProRes encoding.

**‘-c:a wavpack’:** Sets the audio codec to WavPack (.wv), a lossless (or hybrid) audio compression format.

**‘output.mkv’:** Specifies the output container format (.mkv = Matroska), which supports both ProRes video and WavPack audio.



Result will be an output.mkv

****

2: We stream out the video from computer 1 to computer 2.

At computer 1

****

**‘ffmpeg’ :** Launches the FFmpeg program to handle media processing

**‘-re’ :** Reads the input in real-time mode . Mimics live playback speed**.**

**‘-i output.mkv’:** Specifies the input file: output.mkv

**‘-c:v prores\_ks’:** Encodes the video using Apple ProRes via the high-quality

prores\_ks encoder.

**‘-profile:v 3’ :** Sets the ProRes profile to 422 HQ (profile 3) — a high-quality, lightly compressed format.

**‘-pix\_fmt yuv422p10le’:** Sets the pixel format to YUV 4:2:2, 10-bit, little-endian, which is compatible with ProRes

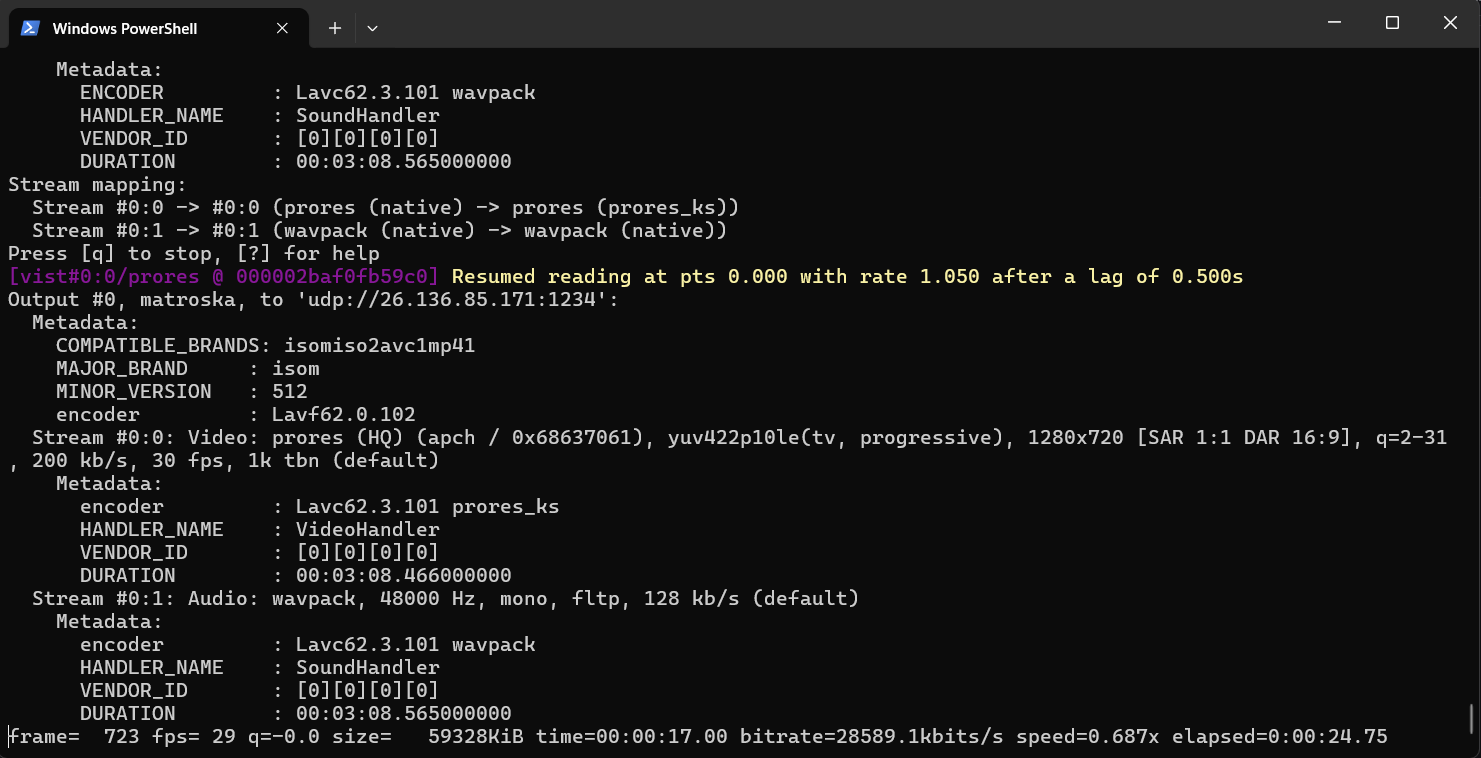
**‘-c:a wavpack’ :** Encodes the audio using the WavPack codec

**‘-ac 1’:** Forces the output audio to have 1 channel (mono) — helps reduce bandwidth.

**‘-b:a 128k’:** Sets the audio bitrate to 128 kbps. Without this, audio might be missing or have poor defaults.

**‘udp://26.136.85.171:1234’ :** Sends the stream via UDP protocol to IP address 26.136.85.171 and port 1234.

Then, start running the terminal at the computer 1:

****

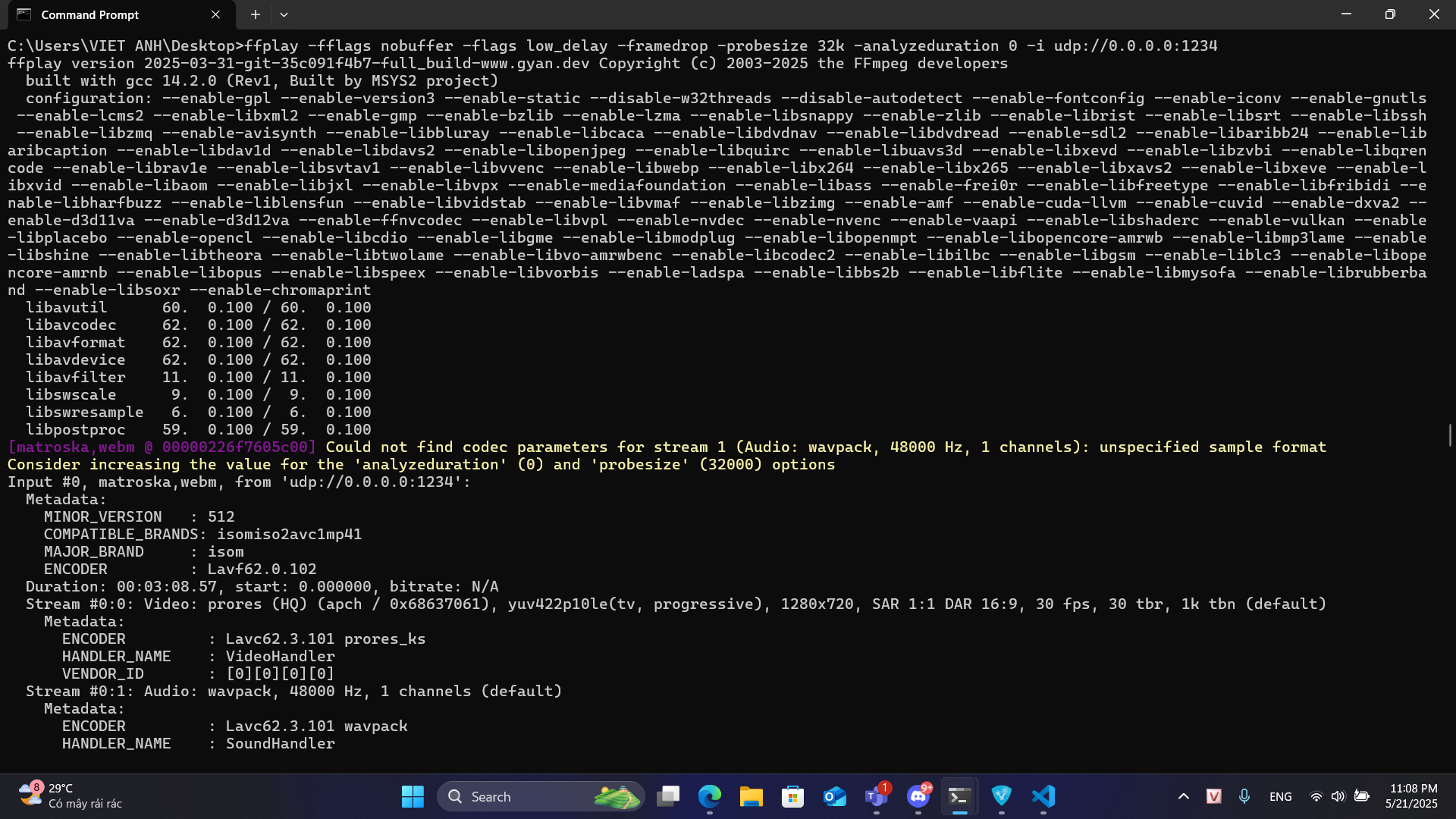
At computer 2

This is IP address of computer 2



Then, start running the terminal at the computer 2:

ffplay -fflags nobuffer -flags low\_delay -framedrop -probesize 32k -analyzeduration 0 -i udp://0.0.0.0:1234



**‘ffplay’ :** Runs the FFmpeg media player (FFplay) to play the incoming stream.

**‘-fflags nobuffer’ :** Disables internal input buffering. Helps reduce latency, especially for live streams.

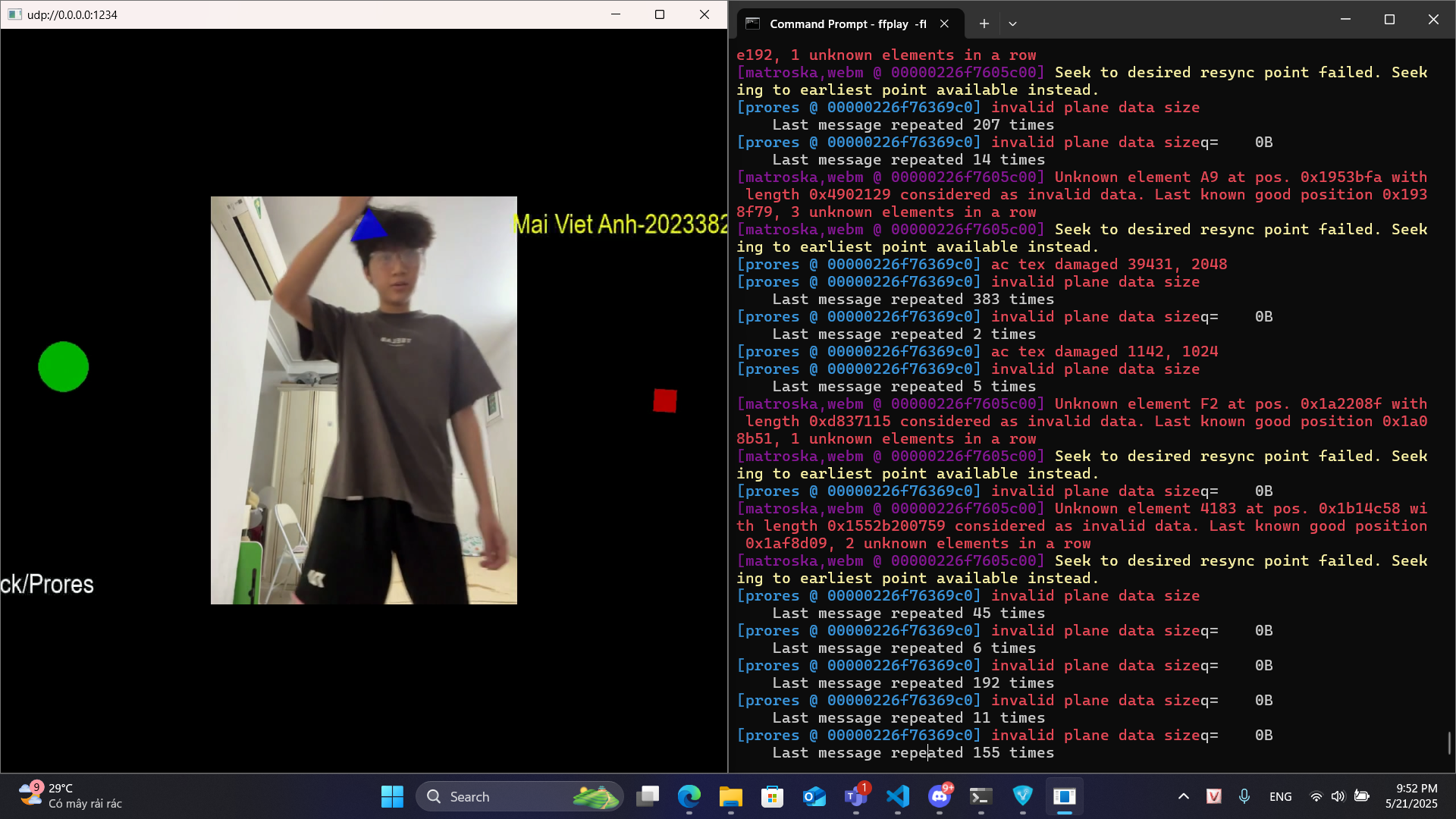
**‘-flags low\_delay’:** Tells the decoder to minimize delay. Useful in real-time applications.

**‘-framedrop’:** Drops frames if decoding/rendering cannot keep up in real time to avoid visible lag.

**‘-probesize 32k’:** Limits the amount of data probed (32 kilobytes) to detect stream format. Speeds up stream start.

**‘-analyzeduration 0’ :** Disables stream analysis duration — further reduces initial delay

**‘-i udp://0.0.0.0:1234’:** Tells FFplay to listen for a UDP stream on port 1234



**VI. Quality Assessment with PSNR and SSIM**

**Assigned Member:** Trần Phú Nghĩa - 20233871

**Description:**

This task involves assessing the quality of received videos by calculating Peak Signal-to-Noise Ratio (PSNR) and Structural Similarity Index (SSIM) metrics, comparing the reconstructed videos (after transmission over Wi-Fi with varying bitrate and packet loss rates) to the original video.

1. **PSNR calculation**ffmpeg -i received\_wavpack1.mkv -i output.mkv -lavfi psnr -f null -

**Code explanation:**

ffmpeg: Launches the FFmpeg tool

-i received\_wavpack1.mkv: First input video (degraded video to evaluate)

-i output.mkv: Second input video (original high-quality reference)

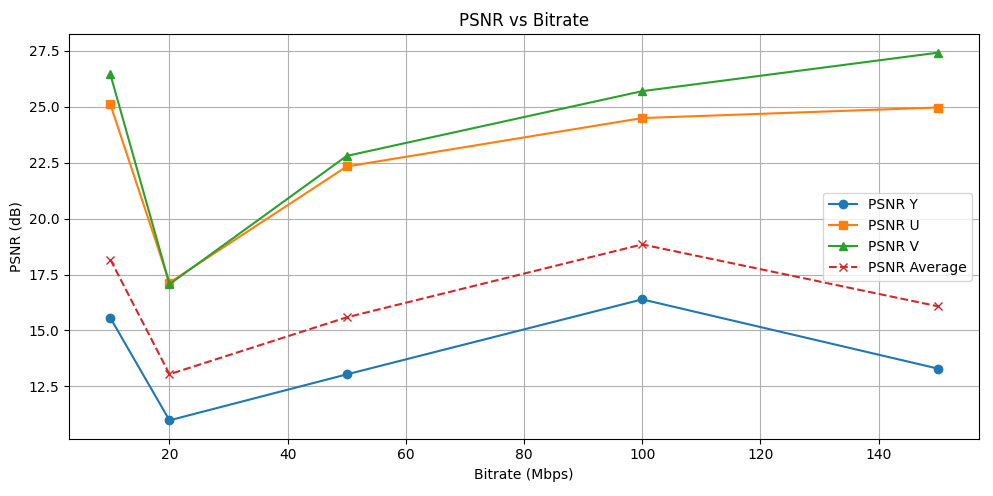
-lavfi psnr: Apply the psnr filter from FFmpeg's libavfilter to compare the inputs

-f null: Don't output any actual video file – discard it

-: Indicates output to standard output (combined with -f null)

**Result:**

| **Test** | **Bitrate (Mbps)** | **Packet Loss (%)** | **Y** | **U** | **V** | **Average PSNR** |
| --- | --- | --- | --- | --- | --- | --- |
| 1 | 10 | 1 | 15.557 | 25.099 | 26.443 | 18.168 |
| 2 | 20 | 4 | 10.981 | 17.135 | 17.065 | 13.042 |
| 3 | 50 | 2 | 13.036 | 22.328 | 22.801 | 15.586 |
| 4 | 100 | 1 | 16.387 | 24.493 | 25.695 | 18.844 |
| 5 | 150 | 0.5 | 13.298 | 24.964 | 27.413 | 16.082 |

****

**Comment:**

* The lowest quality is observed at 20 Mbps, 4% packet loss, where the PSNR average drops to 13.04 dB, indicating high signal degradation.
* Best PSNR average is at 100 Mbps, 1% loss, peaking at 18.84 dB, showing the best trade-off between bitrate and transmission reliability.
* Interestingly, 150 Mbps with 0.5% loss performs worse than 100 Mbps, suggesting that packet loss can outweigh the benefit of higher bitrate.
* Across all entries, the PSNR Y component (brightness/luma) is consistently lower than U and V, proving that human-perceptual sensitivity to brightness makes it more vulnerable to distortion.

1. **SSIM calculation**

ffmpeg -i received\_wavpack1.mkv -i output.mkv -lavfi ssim -f null -

**Explanation:**

ffmpeg: Runs FFmpeg

-i received\_wavpack1.mkv: First input (test video)

-i output.mkv: Second input (original video)

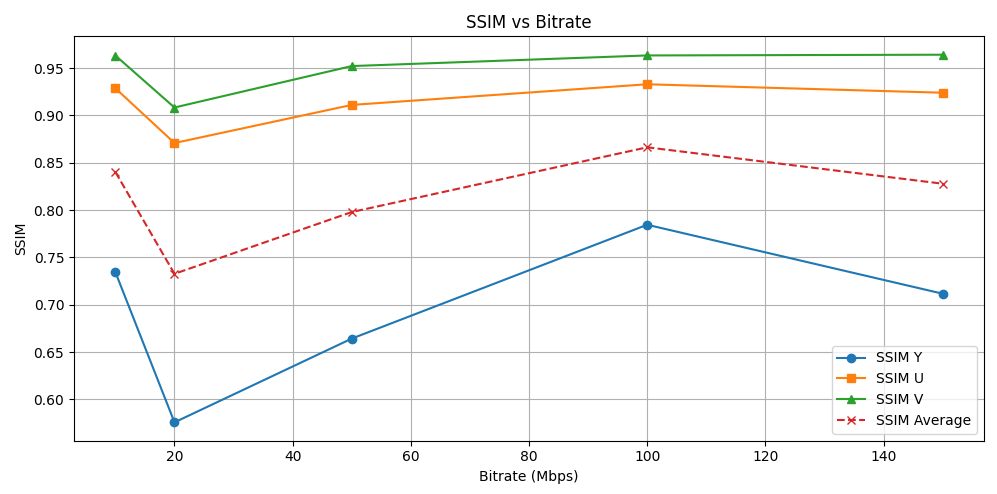
-lavfi ssim: Apply the ssim (Structural SIMilarity) filter

-f null : Discard any actual video output

-: Send output to console (no file created)

**Result:**

| **Test** | **Bitrate (Mbps)** | **Packet Loss (%)** | **Y** | **U** | **V** | **Average PSNR** |
| --- | --- | --- | --- | --- | --- | --- |
| 1 | 10 | 1 | 0.7345 | 0.9287 | 0.9634 | 0.8403 |
| 2 | 20 | 4 | 0.5758 | 0.8709 | 0.9084 | 0.7327 |
| 3 | 50 | 2 | 0.6643 | 0.9112 | 0.9522 | 0.7980 |
| 4 | 100 | 1 | 0.7845 | 0.9330 | 0.9634 | 0.8664 |
| 5 | 150 | 0.5 | 0.7117 | 0.9241 | 0.9642 | 0.8279 |

****

### **Comment:**

* SSIM confirms PSNR trends: lowest quality at 20 Mbps with 4% loss (avg SSIM = 0.7327), highest at 100 Mbps with 1% loss (0.8664).
* 150 Mbps with 0.5% packet loss still sees a drop in SSIM to 0.8279, again showing packet loss is more harmful than low bitrate.
* SSIM Y is always the lowest among the three channels, confirming that structural similarity in luminance is more susceptible to error.
* From a perceptual quality standpoint, SSIM > 0.80 indicates acceptable quality, so only wavpack2 falls below this mark.

**Source code to plot PSNR and SSIM:**

**import matplotlib.pyplot as plt**

**# Bitrates**

**bitrates = [10, 20, 50, 100, 150]**

**# PSNR data**

**psnr\_y = [13.826, 11.259, 13.494, 17.415, 16.222]**

**psnr\_u = [24.406, 17.220, 22.003, 28.381, 26.260]**

**psnr\_v = [25.493, 17.141, 22.443, 29.130, 28.634]**

**psnr\_avg = [16.511, 13.280, 15.957, 20.116, 18.905]**

**# SSIM data**

**ssim\_y = [0.679, 0.551, 0.662, 0.833, 0.747]**

**ssim\_u = [0.920, 0.858, 0.903, 0.964, 0.931]**

**ssim\_v = [0.958, 0.895, 0.938, 0.979, 0.968]**

**ssim\_avg = [0.809, 0.714, 0.791, 0.902, 0.848]**

**# Plot PSNR**

**plt.figure(figsize=(10, 6))**

**plt.plot(bitrates, psnr\_y, marker='o', label='PSNR Y')**

**plt.plot(bitrates, psnr\_u, marker='s', label='PSNR U')**

**plt.plot(bitrates, psnr\_v, marker='^', label='PSNR V')**

**plt.plot(bitrates, psnr\_avg, marker='x', linestyle='--', label='PSNR Average')**

**plt.title('PSNR vs Bitrate')**

**plt.xlabel('Bitrate (Mbps)')**

**plt.ylabel('PSNR (dB)')**

**plt.grid(True)**

**plt.legend()**

**plt.tight\_layout()**

**plt.show()**

**# Plot SSIM**

**plt.figure(figsize=(10, 6))**

**plt.plot(bitrates, ssim\_y, marker='o', label='SSIM Y')**

**plt.plot(bitrates, ssim\_u, marker='s', label='SSIM U')**

**plt.plot(bitrates, ssim\_v, marker='^', label='SSIM V')**

**plt.plot(bitrates, ssim\_avg, marker='x', linestyle='--', label='SSIM Average')**

**plt.title('SSIM vs Bitrate')**

**plt.xlabel('Bitrate (Mbps)')**

**plt.ylabel('SSIM')**

**plt.grid(True)**

**plt.legend()**

**plt.tight\_layout()**

**plt.show()**

**Conclusion:**

1. The critical insight is that packet loss is the most influential factor in quality degradation. Even high bitrates cannot compensate for high packet loss.
2. The optimal transmission quality is achieved at 100 Mbps with 1% packet loss, providing both high PSNR and high SSIM.
3. Both PSNR and SSIM consistently show that luma (Y component) is more affected than chroma (U and V), which is important for encoder tuning and optimization.
4. Increasing bitrate above 100 Mbps shows diminishing returns, especially when network conditions are unstable.
5. In real-time video streaming or conferencing systems, balancing bitrate with packet reliability is more effective than simply maximizing bitrate.