Project Structure

We have two main files:

client.py: This script is responsible for interacting with the server and displaying the video stream.

server.py: This script hosts an HTTP API to control the video streaming process using GStreamer.

Before running the code, ensure you have the following installed:

Python 3.6+: Make sure you have Python installed on your system.

GStreamer: GStreamer multimedia framework and its plugins are required.

Python Libraries: Install required python libraries using pip.

Documentation for server.py Purpose:

This script implements a Flask server with a RESTful API to control a GStreamer-based video streaming pipeline.

Imports: flask: For creating the web server and API. flask_restful: For creating RESTful APIs.

subprocess: For running GStreamer commands. threading: For potentially handling tasks in separate threads (not used in this simple example).

gst_process Variable: A global variable to hold a reference to the currently running GStreamer process.

StreamControl Class: Inherits from flask_restful.Resource and handles API requests.

post() Method: Handles POST requests to /control endpoint. Parses the JSON request body to get the action (either "start" or "stop").

Start Action ("start"): Checks if gst_process is already running. If not, it constructs a GStreamer command to capture video using avfvideosrc, encode it with x264enc, and send it to UDP port 5000. It uses subprocess.Popen to start GStreamer as a background process. Returns a JSON response indicating success or if the stream was already running.

Stop Action ("stop"): Checks if a GStreamer process is running. If so, it terminates the process using gst_process.terminate(). Resets gst_process to None. Returns a JSON response indicating success or failure.

Invalid action: Handles cases for requests that do not include "start" or "stop" actions by returning a json response indicating invalid action.

get() Method: Handles GET requests to /control endpoint. Returns json response indicating if server is running or stopped.

api.add_resource(StreamControl, '/control'): Adds the StreamControl resource to the /control URL.

app.run(host="0.0.0.0", port=8590): Starts the Flask development server, making it accessible from any IP address and listens on port 8590.

Documentation for client.py Purpose:

This script interacts with the Flask server to control the video streaming and displays the received stream in a window.

Imports: gi: For GStreamer and GTK bindings.

os: To set enviroment variables.

Gst: For working with GStreamer pipelines.

Gtk: For creating graphical windows.

requests: To make HTTP requests.

Gst.init(None): Initializes the GStreamer library.

os.environ["GST_GL_API"] = "opengl": Sets the GStreamer OpenGL API environment variable.

view_stream() Function:

Creates a GStreamer pipeline: The pipeline takes data from the udp source on port 5000, converts it to H264 and then displays it.

Creates a GTK window: Sets up a window for video display. on_realize() Function: This function is connected to the realize signal of the GTK window to set the state of the gstreamer pipeline to playing when the window is realized.

on_close() Function: This function is connected to the delete-event signal of the GTK window to set the state of the gstreamer pipeline to null when the window is closed. Connects signals: The realize signal sets gstreamer to play on window realize, and the delete event is connected to stop the gstreamer pipeline on window close.

Gtk.main(): Starts the GTK main loop to display the window. Error Handling Includes a try-except for a Keyboard Interrupt.

Main Execution (if **name** == "**main**")

Presents a menu to the user to choose between start stream, stop stream, or view stream. Sends a POST request with action as "start" or "stop" to the Flask server to start or stop the video stream. Calls the view_stream() function to display the received stream in a window.

Setup and Usage Instructions Install Prerequisites:

Follow the steps in the "Prerequisites" section above to set up your environment. Save the Code:

python3 server.py

Save server.py and client.py into the same directory. Run the Server: python server.py

Run the client:

Python3 client.py

A prompt will appear in terminal asking to select an option.

Select "1" to start the stream and press enter, response will appear in the terminal. Select "3" to view the stream and press enter, the stream should appear in a new window. Select "2" to pause the stream. you can close the video window and continue to use other options if you would like.

Detailed Explanation of the GStreamer Pipelines:

Server (GStreamer Command):

gst-launch-1.0 avfvideosrc ! videoconvert ! x264enc tune=zerolatency ! rtph264pay config-interval=1 ! udpsink host=127.0.0.1 port=5000

avfvideosrc: Captures video from an Apple video framework device. This is appropriate for macOS. If on linux, use v4l2src.

videoconvert: Converts the video format to a format that the encoder can understand.

x264enc tune=zerolatency: Encodes the video using the H.264 format and optimizing for low latency.

rtph264pay config-interval=1: Packages the H.264 encoded video into RTP packets, sending configuration packets every 1 frame.

udpsink host=127.0.0.1 port=5000: Sends the RTP packets over UDP to the specified address and port.

Client (GStreamer Pipeline):

udpsrc port=5000 ! application/x-rtp,media=video,payload=96,clock-rate=90000 ! rtph264depay ! h264parse ! avdec_h264 ! videoconvert ! autovideosink udpsrc port=5000: Receives the UDP stream from port 5000.

application/x-rtp,media=video,payload=96,clock-rate=90000: Specifies the stream is video encoded as H.264.

rtph264depay: Depayloads the RTP packets, extracting the raw H.264 data. h264parse: Parses the raw H.264 stream for analysis.

avdec_h264: Decodes the H.264 data into raw video frames.

videoconvert: Converts the raw video to a format the videosink can understand.

autovideosink: Selects the appropriate video sink based on your system to display the video.