RTSP Streaming Server with Optional Object Detection and MQTT Integration

Deniz Doğa Yanar - 2001336

Acar Kaan Şenel - 2104099

Table of Contents

Introduction

This report shows how to create an RTSP streaming server using GStreamer and Flask, incorporating other optional features such as object detection via OpenCV and YOLO and also MQTT for remote control. The system is defined to offer a more flexible streaming system with REST APIs and messaging that can be applied across a diverse range of applications such as surveillance, remote monitoring, and real-time video processing.

System Architecture

The system architecture comprises several interconnected components:

- **GStreamer**: Handles video streaming and processing.
- **Flask**: Provides a RESTful API for controlling the RTSP server.
- **OpenCV & YOLO**: Optional modules for real-time object detection.
- **MQTT**: Optional messaging protocol for remote control and status updates.
- **Flask Application**: Serves as the central controller orchestrating all functionalities.

Key Components

1. Initialization

Purpose: Set up the environment, import necessary libraries, and initialize global variables.

Details:

- **Library Imports**: The script imports essential libraries including `os`, `threading`, `time`, `subprocess`, `signal`, `GStreamer` (`Gst`), `GstRtspServer`, `Flask`, and optional modules like `paho.mqtt.client`, `cv2`, and `numpy`.
- **GStreamer Initialization**: `Gst.init(None)` initializes the GStreamer library.
- **Flask App Initialization**: `app = Flask(__name__)` creates a Flask application instance.
- **Global Variables**:
    - `rtsp_server`: Holds the RTSP server instance.
    - `main_loop`: Manages the GLib main loop.
    - `pipeline_lock`: Ensures thread-safe operations on the pipeline.
    - `object_detection_enabled`: Toggles YOLO-based detection.
    - `VIDEO_FILE_PATH`: Specifies the video file to stream.
- **Optional Components**:
    - **MQTT**: Checks for MQTT library availability and sets related variables.
    - **OpenCV**: Checks for OpenCV and NumPy availability for object detection.

2. Object Detection with OpenCV and YOLO

Purpose: Integrate real-time object detection into the video stream using YOLO models.

Details:

- **Configuration Paths**:
    - `YOLO_CONFIG_PATH`: Path to the YOLO configuration file.
    - `YOLO_WEIGHTS_PATH`: Path to the YOLO weights file.
    - `YOLO_NAMES_PATH`: Optional path to class label names.
- **Model Loading**:
    - The `load_yolo_model()` function initializes the YOLO network using OpenCV's DNN module.
    - It sets the preferable backend and target for inference.
    - Loads class names if provided.
- **Integration with GStreamer**:
    - Utilizes an `appsink` to intercept video frames for processing.
    - Processes frames with YOLO, draws bounding boxes, and reinserts them into the stream.

3. GStreamer RTSP Server

Purpose: Set up and manage the RTSP streaming server using GStreamer.

Details:

- **Custom Media Factory**:
  - `CustomRtspMediaFactory` class inherits from `GstRtspServer.RTSPMediaFactory`.
  - Constructs a GStreamer pipeline based on whether object detection is enabled.
  - For object detection:
    - Uses `appsink` to process frames with YOLO.
    - Employs `appsrc` to inject modified frames back into the stream.
  - Without object detection:
    - Sets up a simple pipeline with `filesrc`, `decodebin`, `videoconvert`, `x264enc`, and `rtph264pay`.

4. Appsink Callback for Object Detection

Purpose: Handle incoming video frames, perform object detection, and modify frames accordingly.

Details:

- **Callback Function**:
  - `on_new_sample(sink, pipeline)`: Triggered for each new frame.
  - Retrieves the frame, maps buffer data, and converts it to a NumPy array.
  - Runs YOLO detection to identify objects and draw bounding boxes.
  - Converts the processed frame back to a GStreamer buffer and pushes it to `appsrc`.
- **YOLO Processing**:
  - Utilizes blob creation, forward pass through the network, and non-max suppression to filter detections.
  - Annotates frames with bounding boxes and class labels.

5. RTSP Server Control

Purpose: Provide functions to start and stop the RTSP server.

Details:

- **Starting the Server**:
  - `start_rtsp_server()`: Initializes the RTSP server, sets up mount points, and starts the GLib main loop in a separate thread.
- **Stopping the Server**:
  - `stop_rtsp_server()`: Quits the main loop and cleans up the RTSP server instance.

6. Flask REST API

Purpose: Expose RESTful endpoints to control the RTSP server and object detection features.

Details:

- **Endpoints**:
  - `/start` (POST): Starts the RTSP server.
  - `/stop` (POST): Stops the RTSP server.
  - `/status` (GET): Retrieves the current status of the RTSP server.
  - `/enable_detection` (POST): Enables YOLO-based object detection.
  - `/disable_detection` (POST): Disables object detection.
- **Responses**:
  - Returns JSON responses indicating the status of operations or errors if any.

7. MQTT Integration

Purpose: Allow remote control and status updates via MQTT messaging protocol.

Details:

- **MQTT Setup**:
  - Checks for MQTT library availability.
  - Configures MQTT client with broker details and topic subscriptions.
- **Callbacks**:
  - `on_mqtt_connect`: Subscribes to control topics upon successful connection.
  - `on_mqtt_message`: Handles incoming MQTT messages to start or stop the RTSP server and publishes status updates.
- **Operation**:
  - Runs MQTT network loop in a separate daemon thread to handle messages asynchronously.

8. Main Entry Point

Purpose: Define the script's entry points for running the server and an optional client for testing.

Details:

- **Server Execution**:
  - When run as the main program, initializes Flask and optionally MQTT.
  - Starts the Flask application to listen for API requests.
- **Client Execution**:
  - Contains a `main()` function that acts as a client to receive and display the RTSP stream using OpenCV.
  - Connects to the RTSP server, displays the stream in a window, and allows exiting with the 'q' key.
- **Execution Flow**:
  - The `main()` function is protected with `if __name__ == "__main__":` to ensure proper execution context.

Usage

1. **Prerequisites**:
    - Ensure Python 3.x is installed.
    - Required Libraries: `flask paho-mqtt opencv-python numpy pygobject`
    - Install GStreamer and its RTSP server component.
    - (Optional) Obtain YOLO configuration, weights, and class names files.
2. **Configuration**:
    - Update `VIDEO_FILE_PATH` with the path to the video file to stream.
    - If using object detection, set the correct paths for YOLO files.
    - Configure MQTT broker details if MQTT integration is desired.
3. **Running the Server**:

    ```
    python server.py
    ```

    The Flask API will be accessible at `http://<server_ip>:5000`.

4. **Using the REST API**:

Start Streaming:
```
curl -X POST http://<server_ip>:5000/start
```

Stop Streaming:
```
curl -X POST http://<server_ip>:5000/stop
```

Check Status:
```
curl http://<server_ip>:5000/status
```

Enable Object Detection:
```
curl -X POST http://<server_ip>:5000/enable_detection
```

Disable Object Detection:

```
curl -X POST http://<server_ip>:5000/disable_detection
```

5. **Accessing the RTSP Stream**:

Use an RTSP-compatible client (e.g., VLC) to access:
`rtsp://<server_ip>:8554/test`

6. **MQTT Control**:

    - Publish messages (`start`, `stop`) to the `stream/control` topic to control the server remotely.
    - Subscribe to `stream/status` for status updates.

Dependencies

- **Core Libraries**:
    - `os`, `threading`, `time`, `subprocess`, `signal`
    - `gi` (GObject Introspection)
    - `Flask`
- **Optional Libraries**:
    - `paho.mqtt.client` for MQTT integration.
    - `cv2` (OpenCV) and `numpy` for object detection.
- **GStreamer**:
    - Must be installed on the system along with necessary plugins for video processing and RTSP server functionality.

Conclusion

The given Python program provides a complete solution for a highly flexible RTSP streaming server that can serve as an object detector and be controlled remotely via MQTT. It efficiently handles all media assets using GStreamer, uses Flask to expose the server functionalities through RESTful APIs, and can use additional optional modules such as OpenCV and MQTT, thus making this design sufficiently extensible. It can accommodate a variety of applications-from security to remote analytics-for both video inputs and outputs.

Future enhancements could include:

- **Dynamic Pipeline Configuration**: Allowing on-the-fly changes to the GStreamer pipeline without restarting the server.
- **Enhanced Security**: Implementing authentication and encryption for both the REST API and RTSP streams.
- **Scalability**: Optimizing for handling multiple streams and higher resolutions.
- **Advanced Object Detection**: Incorporating more sophisticated models or allowing user-defined models.

Overall, the system demonstrates a robust integration of multimedia streaming with modern web and messaging technologies.