The Video Streaming with RTSP (Real-Time Streaming Protocol) and RTP (Real-Time Transport Protocol) Lab is a comprehensive computer networking project that involves implementing a simple video streaming system. In this lab, you'll create a video server and client capable of streaming video content using RTSP for session control and RTP for transmitting the actual video data. This project provides hands-on experience with multimedia streaming and real-time communication protocols.

Project Title: Video Streaming with RTSP and RTP Lab in Python

Project Description:

Overview:

The goal of this project is to implement a basic video streaming system that uses RTSP for session control and RTP for transmitting video data. The server will stream video content to clients, and clients will display the streamed video.

Components:

1. **RTSP Server:**
    - The RTSP server handles session setup, control, and teardown.
    - Accepts RTSP requests from clients for starting, pausing, and stopping the video stream.
    - Uses RTSP to negotiate session parameters with clients.
2. **RTP Server:**
    - The RTP server transmits video data to clients in real-time.
    - Packs video frames into RTP packets and sends them to clients.
    - Uses UDP for transporting RTP packets.
3. **Video Source (Optional):**
    - Provide a source of video content for streaming.
    - This could be a pre-recorded video file or a live video feed.
4. **RTSP Client:**
    - The RTSP client sends RTSP requests to the server to initiate and control video streaming sessions.
    - Receives session parameters and negotiates with the server.
5. **RTP Client:**
    - The RTP client receives RTP packets from the server.
    - Decodes and displays video frames in real-time.

Requirements:

1. **Programming Language:**
    - Use Python for both the RTSP and RTP server implementations.
2. **RTSP Protocol:**
    - Implement the RTSP protocol for session control, including setup, play, pause, and teardown.

3. **RTP Protocol:**
    - Implement the RTP protocol for real-time transmission of video data.
4. **Video Encoding (Optional):**
    - Optionally, implement video encoding and decoding to pack and unpack video frames into RTP packets.
5. **RTSP Client:**
    - Implement the RTSP client to initiate and control video streaming sessions.
6. **RTP Client:**
    - Implement the RTP client to receive, decode, and display video frames.

Sample Workflow:

1. The user runs the RTSP server script.
2. The RTSP server initializes and awaits RTSP requests from clients.
3. The user runs the RTP server script.
4. The RTP server begins streaming video frames to waiting clients.
5. The user runs the RTSP client script, specifying the RTSP server's address.
6. The RTSP client sends RTSP requests to the server to initiate and control the video stream.
7. The RTSP client receives session parameters and negotiates with the server.
8. The user runs the RTP client script.
9. The RTP client receives and decodes RTP packets, displaying the video frames in real-time.

Additional Features (Optional):

- **Multiple Clients:** Extend the project to support multiple clients simultaneously.
- **Video Source Selection:** Allow users to choose different video sources for streaming.
- **Adaptive Streaming:** Implement adaptive streaming techniques to adjust video quality based on network conditions.

Note:

Before starting the implementation, ensure that you have a good understanding of the RTSP and RTP protocols, as well as basic concepts of video encoding and decoding. Familiarize yourself with relevant libraries or tools for handling multimedia data in Python, such as OpenCV. Consider error handling, performance optimization, and synchronization between the RTSP and RTP components for a more robust implementation.