RTSP (Real-Time Streaming Protocol) :

RTSP, or Real-Time Streaming Protocol, is a network control protocol designed for controlling the delivery of multimedia content over the Internet. It's primarily used for streaming media, such as audio and video, and enables clients to control the playback of media files stored on a server in real-time. RTSP operates on top of the transport layer protocols like TCP or UDP and typically uses port 554.

Key features of RTSP include:

1. Client-Server Model: RTSP follows a client-server architecture where a client establishes a session with a server to control the streaming media playback.
2. Session Control: RTSP provides methods for session control, such as SETUP, PLAY, PAUSE, and TEARDOWN, allowing clients to initiate, control, and terminate streaming sessions.
3. Media Content Description: RTSP supports the exchange of media content description information using SDP (Session Description Protocol), enabling clients to understand the characteristics of the media stream.
4. Stateful Protocol: RTSP is a stateful protocol, meaning that it maintains session state information between client and server throughout the duration of a streaming session.
5. Interoperability: RTSP is widely supported by various media players, servers, and streaming devices, making it an interoperable standard for streaming media delivery.

Now, let's delve into my RTSP streamer application:

RTSP Streamer Application :

My RTSP streamer application consists of several components that work together to enable the streaming of multimedia content from a server to multiple clients over a network. Here's an overview of the key components and how they function:

1. Server (server.py): The server component listens for incoming RTSP requests from clients and manages the streaming of multimedia content. It handles client connections, session setup, media streaming, and session teardown.
2. Server Worker (Serverworker.py): Each client connection is managed by a server worker instance. Server workers handle incoming RTSP requests from clients, such as SETUP, PLAY, PAUSE, and TEARDOWN, and coordinate the streaming of media content to clients.
3. Video Stream (VideoStream.py): The Video Stream class provides functionality to read frames of multimedia content from a file (e.g., movie.Mjpeg). It reads frame data from the file and provides methods to retrieve the next frame and frame number.

4. Client (client.py): The client component initiates connections to the server, sends RTSP requests to control the streaming session, and receives multimedia content from the server for playback.
5. Client Launcher (clientlauncher.py): The client launcher script provides a user-friendly interface for launching the client application. It prompts the user to enter the server address, port, and other parameters required to establish a connection and start streaming.
6. Multimedia Content (movie.Mjpeg): This file contains the multimedia content (e.g., video frames) that will be streamed from the server to clients. It's read by the Video Stream class for streaming.

Overall, my RTSP streamer application implements the RTSP protocol for session management and media streaming, allowing clients to control and stream multimedia content in real-time over a network. It provides a scalable and efficient solution for distributing multimedia content to multiple clients simultaneously.

Setup Instructions:

1. Open Command Prompt/Terminal.
2. Navigate to the directory containing your RTSP streamer application files.
3. Run the server by typing: python server.py 8000.
4. Open another Command Prompt/Terminal window.
5. Navigate to the same directory.
6. Run the client launcher by typing: python clientlauncher.py 127.0.0.1 8000 5001 movie.Mjpeg.
7. Follow the prompts in the client window to start streaming the video.