VIETNAM NATIONAL UNIVERSITY, HO CHI MINH CITY
UNIVERSITY OF TECHNOLOGY
FACULTY OF COMPUTER SCIENCE AND ENGINEERING

Computer Network (CO3093)

Assignment 1

Video Streaming

Lecturer:	- Nguyễn Mạnh Thìn
Student:	- Nguyễn Minh Hùng 1952737
	- Nguyễn Bá Minh Hưng 1952748
	- Đinh Hoàng Anh 1952553
	- Phạm Nhật Hoàng 1952703

HO CHI MINH CITY, OCTOBER 2021

Member list & Workload

No.	Fullname	Student ID	Problems	Percentage of work
1	Nguyễn Minh Hùng	1952737	...	25%
2	Nguyễn Bá Minh Hưng	1952748	...	25%
3	Đinh Hoàng Anh	1952553	...	25%
4	Phạm Nhật Hoàng	1952703	...	25%

Contents

1 Analysis of problem requirements

1.1 Introduction

Multimedia communication gradually becomes more and more important as we are witnessing the tremendous growth of these applications on the Internet.

Live view technology makes it possible to watch videos without downloading the entire video.

RTSP is a protocol which will be used to communicate between the client and the server.

1.2 Real Time Streaming Protocol (RTSP)

Real Time Streaming Protocol (RTSP) establishes and controls either a single or several time-synchronized streams of continuous media such as audio and video. It does not typically deliver the continuous streams itself, although interleaving of the continuous media stream with the control stream is possible In other words, RTSP acts as a "network remote control" for multimedia servers.

1.3 Real Time Transfer Protocol (RTP)

Real Time Transfer Protocol (RTP) is a network standard designed for transmitting audio or video data that is optimized for consistent delivery of live data. It is used in internet telephony, Voice over IP and video telecommunication. It can be used for one-on-one calls (unicast) or in one-to-many conferences (multicast).

1.4 Module Analysis

Client, ClientLauncher
These two classes will implement the user interface and client functions.

ServerWorker, Server
These two classes will implement the server function, respond to RTSP commands and play video.

RtpPacket
This class will handle RTP packets (including encode, decode).

VideoStreaming
This class will read the uploaded video file.

1.5 Target

Implement the RTSP protocol at the client.

Implement RTP encapsulation at the server.

2 Function Description

Client

- createWidget: Create UI for application (buttons, video,...)
- setupMovie: send RTSP SETUP request to server.
- exitClient: send RTSP TEARDOWN request to server, exit the application.
- pauseMovie: send RTSP PAUSE request to server for pausing video.
- playMovie: send RTSP PLAY request to server for playing video.
- listenRtp: Listen for RTP packet (receive and decode), update frame number.
- writeFrame: write dataframe of video into image file.
- updateMovie: Update and display image file.
- connectToServer: connect to server.
- sendRtspRequest: send RTSP request to server.
- recvRtspReply: receive RTSP Response from server and decode packet. Exit application if the request is TEARDOWN
- parseRtspReply: parse the RTSP Reply from server.
- openRtpPort: create a connection port between client and server.
- handler: Handler on explicitly closing the GUI window.

RtpPacket

- encode: encode the RTP packet with header fields and payload.
- decode: decode the RTP packet, split into two parts header and payload.
- version: return version of RTP packet.
- seqNum: return sequence number of RTP packet.
- timestamp: return timestamp.
- payloadType: return payload type.

- getPayload: return payload.
- getPacket: return packet.

ServerWorker

- run: run the server.
- recvRtspRequest: receive request from client and decode packet.
- processTrspRequest: process request data.
- makeRtp: packetize the video data and send to client.
- replyRtsp: Send reply to the client . ("200OK" if success, "404 NOT FOUND" if cannot find file, "500 CONNECTION ERROR" if getting a connection error.

VideoStream

- NextFrame: return next frame.
- frameNbr: return frame number.

3 List of components

The client/server model features two major components: a server and a client. These two components interact with each other through a network connection using RTSP protocol. The communication is unidirectional: The client issues a request to the server, and after processing the request the server returns a response. There could be multiple client components issuing requests to a server that is passively waiting for them. Hence, the important operations in the client-server paradigm are request, accept (client side), and listen and response (server side).

4 Application Flow

- Run Server.py on Command Prompt at the directory containing the application to start server. Command: python Server.py server_port. Server will initiate socket initialization RTSP/TCP, assign the socket with the IP address and port we provided, then start listen for connections from the server.

- Run ClientLaucher.py on Command Prompt at the directory containing the application to launch Client. Command: **python ClientLauncher.py server_host server_port PRT_port video_file**. ClientLaucher.py will accept user-supplied parameters, call Client. At the Client, it will proceed to initialize the status, parameters, create a socket to connect to Server through the address and port provided.

- The server may accept the connection when it receives a request from the client. In ServerWorker, it will initialize Server states, parameters and ready to receive requests RTSP from Client. At the same time, Client will initialize user interface.

- After initializing Server and Client to connect to each other via RTSP/TCP protocol. Client will send to Server commands such as **SETUP**, **PLAY**, **PAUSE**, **TEARDOWN** through RTSP protocol, these commands will tell the Server the next action it needs to do.

- These parameters will be response from Server to Client via RTSP Protocol such as: **OK_200**, **FILE_NOT_FOUND_404**, **CON_ERR_500** to verified with the Client if the Server received its request correctly.

- After the Client receives the Server's reply, it will change its status accordingly into: **READY**, **PLAYING**.

- If a **SETUP** request is sent to the Server, the RTSP **SETUP** packet will include:

- **SETUP** request.
- Video name to play.
- Protocol type: RTSP/1.0
- Number of RTSP Packet Sequence begin at 1
- Transport Protocol: RTP/UDP
- RTP port for video streaming

Information of requestSent will be also recorded.

- When the Server receives the **SETUP** request, it will randomly assign the Client a Specific Session Number. If there is an error with the request or the server state, it will return the **ERROR** packet to the Client. If the request is correct, the server will open the video file specified in the **SETUP** request and initialize the video frame number its to 0.

- If the request is processed correctly, the Server will reply back **OK_200** to the Client and set its STATE to **READY**.

- The Client will perform a loop to receive the Server's RTSP Reply.

- Then it proceeds to parse the RTSP response from the server, get Session Number previously initialized by the Server for the Client and if the response packet is **SETUP** command, Client will set the status (STATE) to **READY**. Then it opens an RtpPort to receive the incoming video stream .

- Initialize RTP socket to receive RTP packet from server, set timeout for socket to 0.5 seconds. Bind for socket address using provided RTP port by the user, if it fails, a message window pops up.

- Then, if a **PLAY** request is sent to the Server, the RTSP Packet Sequence Number is incremented by 1 and the RTSP **PLAY** packet will include:

- **SETUP** request.
- Video name to play.
- Protocol type: RTSP/1.0
- Number of RTSP Packet Sequence
- SessionId (randomly initialized from ServerWorker and returned **SETUP** request)

Information of requestSent will be also recorded.

- When receiving the request, the Server will create a Socket to transmit RTP over UDP and initiates a threading to send RTP packets.

- VideoStream.py will help cut video files into separate frames and put each frame into the RTP datagram.

- The implementation of RTP packet encryption include the header and payload fields will be performed at RtpPacket.py, they are inserted into the RTP packet via bitwise operations

- The minimum size of an RTP packet header is 12 bytes. After the header main is the extension header and it is not necessary to have this header. Detail in a header will be shown as follow:

- Version (2 bits): Indicates the version of this protocol.
- P (Padding) (1 bit): Indicates the number of expansion bytes to add to the end of the RTP packet. For example, in case we want to use encryption algorithms, we can add a number of bytes to the end of the packet to encode the frame in transit.
- X (Extension) (1bit): Indicates that an extension header is added after the main header or not.

- CC (CSRC Count) (4 bits): Contains the CSRC identifier number indicate the fixed size header.
- M (Marker) (1 bit): Indicates the level of the application and is defined by a profile. If it is set, it means that the current data has been calculated suitably.
- PT (Payload Type) (7 bits): Indicates the format of the video file. This is a specification defined by an RTP profile.
- Sequence Number (16 bits) : the number of frames. And it will be increased by 1 unit for each RTP packet before sending and used by the receiver to detect lost packets and possibly recover the packet with that sequence number.
- Timestamp (32 bits): to be used to notify the receiver to re-transmit the frame within the appropriate time interval.
- Identifier for the streaming source. Each source that allows streaming video will be identified by a unique RTP session.

- Finally, the RTP packet will include a header and a frame of the video sent to the RTP port on the client side.

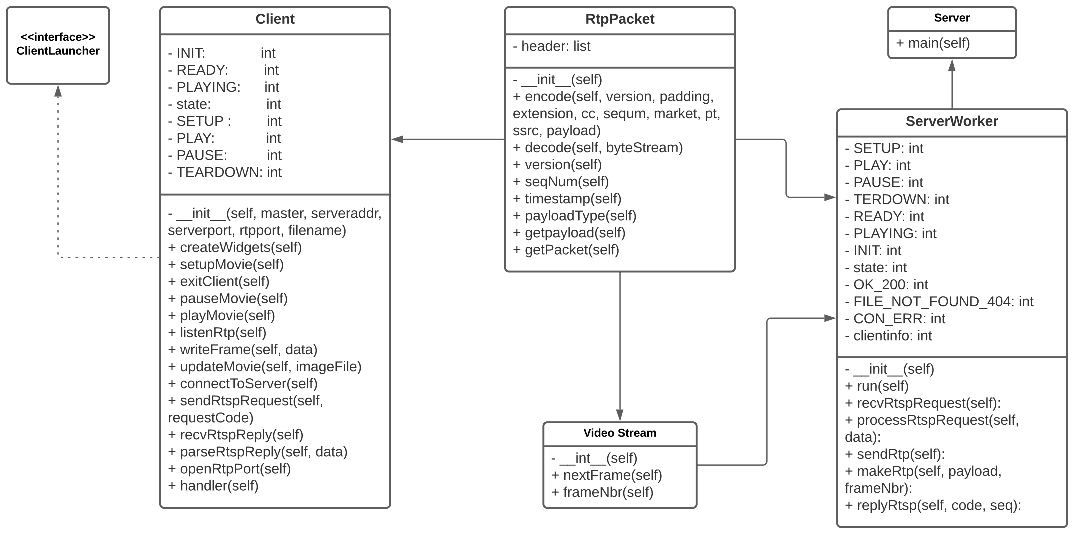
- The client will receive RTP server packets, then proceed to decode the packet to get the header and frame.

- The frames will then be displayed on the UI.

- If the "PAUSE" command is sent from the Client to the Server, it will prevent the Server from sending frames to the Client.

- If the "TEARDOWN" command is sent from Client to Server, it will also prevent the Server from sending video frames to the Client and close the Client's connection as well.

5 Class Diagram



6 A summative evaluation of achieved results

6.1 Interface

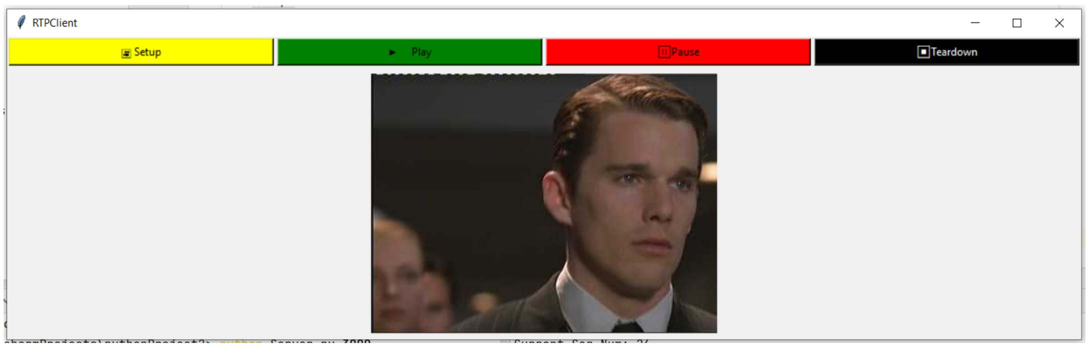
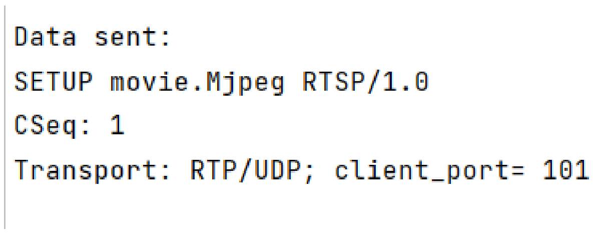


Figure 1: Interface

6.2 SETUP

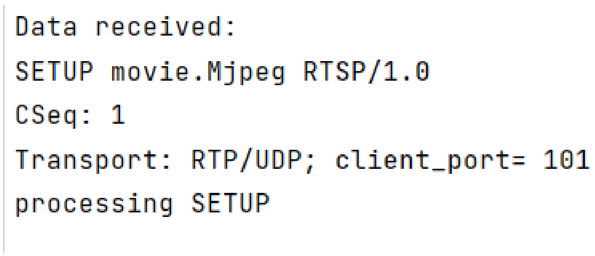
Client Terminal



```
Data sent:
SETUP movie.Mjpeg RTSP/1.0
CSeq: 1
Transport: RTP/UDP; client_port= 101
```

Figure 2: Interface

Server Terminal

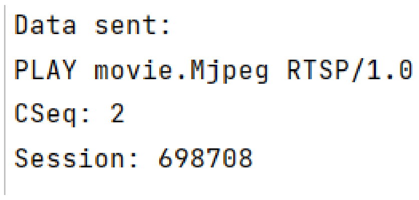


```
Data received:
SETUP movie.Mjpeg RTSP/1.0
CSeq: 1
Transport: RTP/UDP; client_port= 101
processing SETUP
```

Figure 3: Interface

6.3 PLAY

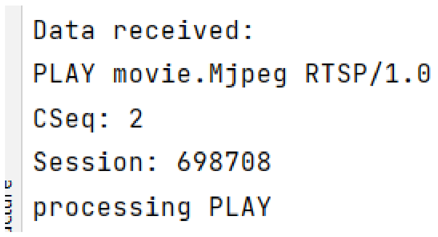
Client Terminal



```
Data sent:
PLAY movie.Mjpeg RTSP/1.0
CSeq: 2
Session: 698708
```

Figure 4: Interface

Server Terminal

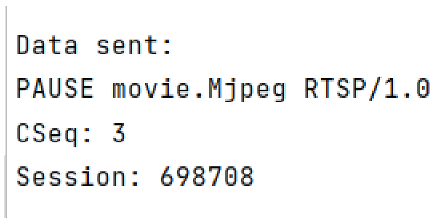


```
Data received:
PLAY movie.Mjpeg RTSP/1.0
CSeq: 2
Session: 698708
processing PLAY
```

Figure 5: Interface

6.4 PAUSE

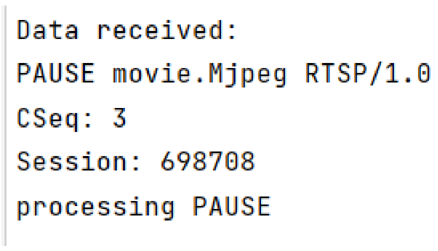
Client Terminal



```
Data sent:
PAUSE movie.Mjpeg RTSP/1.0
CSeq: 3
Session: 698708
```

Figure 6: Interface

Server Terminal

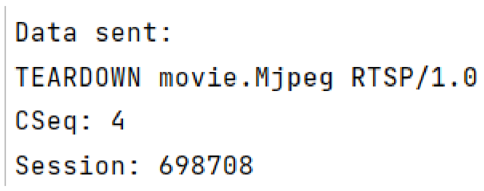


```
Data received:
PAUSE movie.Mjpeg RTSP/1.0
CSeq: 3
Session: 698708
processing PAUSE
```

Figure 7: Interface

6.5 TEARDOWN

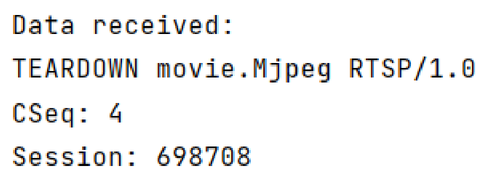
Client Terminal



```
Data sent:
TEARDOWN movie.Mjpeg RTSP/1.0
CSeq: 4
Session: 698708
```

Figure 8: Interface

Server Terminal



```
Data received:
TEARDOWN movie.Mjpeg RTSP/1.0
CSeq: 4
Session: 698708
```

Figure 9: Interface

7 User Manual

Firstly, we have to split the terminal into 2 section, one is for running the server, the other is for running the application.

7.1 Start the server

```
python Server.py 3000
```

where 3000 is the port the server listens to for incoming RTSP connections ($>$ 1024).

7.2 Run the client

```
python ClientLauncher.py localhost 3000 101 movie.Mjpeg
```

where `localhost` is the hostname of the current device, 3000 is the server port, 101 is the RTP Port which receive RTP packet, `movie.Mjpeg` is the video file name.

8 Using the application

4 main functions of the application:

- `Setup`: Initialize the video.
- `Play`: play the video
- `Pause`: pause the video, resume by click on button play.
- `Teardown`: end the video and exit the application.

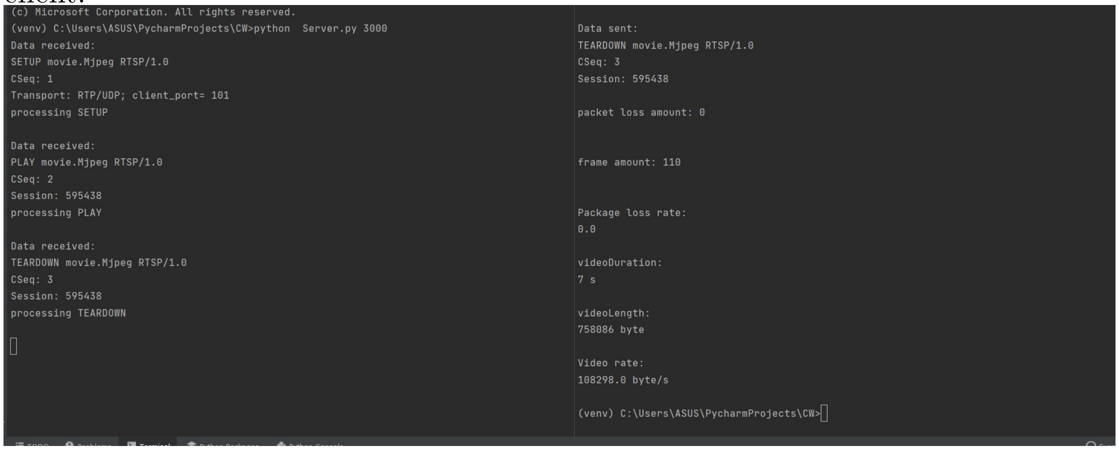
9 Extend

9.1 Statistics about the session

Calculate session statistics, perform RTP packet loss rate calculation and data transfer rate of video (in bytes per second):

```python
    def exitClient(self):
        """Teardown button handler."""
        self.sendRtspRequest(self.TEARDOWN)
        rate = float(self.lostFrameCounter) / float(self.frameNbr)
        print("\npacket loss amount: " + str(self.lostFrameCounter) + "\n")
        print("\nframe amount: " + str(self.frameNbr) + "\n")
        print('\nPackage loss rate:\n' + str(rate))
        videoDuration = self.endTimestamp - self.beginTimestamp  # in second
        videoRate = float(self.videoSize) / float(videoDuration)
        print('\nvideoDuration:\n' + str(videoDuration) + ' s')
        print('\nvideoLength:\n' + str(self.videoSize) + ' byte')
        print('\nVideo rate:\n' + str(videoRate) + ' byte/s')
        self.master.destroy()  # Close the GUI
        os.remove(CACHE_FILE_NAME + str(self.sessionId) + CACHE_FILE_EXT)  #
    Delete the cache image from video
```

Calculation results are displayed after the session is over and displayed on the client:



```
(c) Microsoft Corporation. All rights reserved.
(venv) C:\Users\ASUS\PycharmProjects\CW>python  Server.py 3000
Data received:
SETUP movie.Mjpeg RTSP/1.0
CSeq: 1
Transport: RTP/UDP; client_port= 101
processing SETUP

Data received:
PLAY movie.Mjpeg RTSP/1.0
CSeq: 2
Session: 595438
processing PLAY

Data received:
TEARDOWN movie.Mjpeg RTSP/1.0
CSeq: 3
Session: 595438
processing TEARDOWN
```

```
Data sent:
TEARDOWN movie.Mjpeg RTSP/1.0
CSeq: 3
Session: 595438

packet loss amount: 0


frame amount: 110


Package loss rate:
0.0

videoDuration:
7 s

videoLength:
758086 byte

Video rate:
108298.0 byte/s

(venv) C:\Users\ASUS\PycharmProjects\CW>
```

9.2 Removing SETUP button

The user interface on the RTPClient has 4 buttons for the 4 actions but standard of the media player such as RealPlayer or Windows Media Player has only 3 buttons which are: PLAY, PAUSE and TEARDOWN, no SETUP button. The solution given is when the user presses PLAY, program will process two features SETUP and PLAY:

1. Insert the header that you specify the port for the Socket datagram RTP that

you just created. The client will get the server response and get the ID of the RTSP instance.

2. The server creates a Socket to transmit RTP over UDP and initiates a step to send the video packet.

3. Client of STATE will switch from INIT to READY to PLAYING.

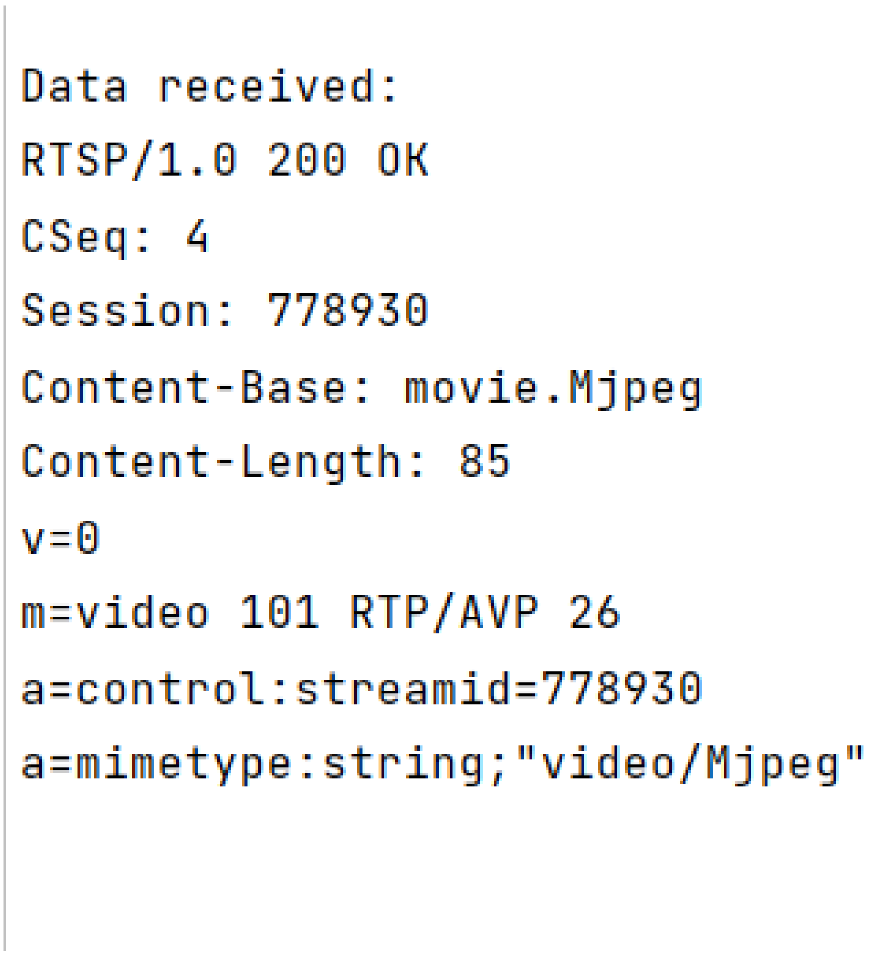
```
def playMovie(self):
    if self.state == self.INIT:
        self.sendRtspRequest(self.SETUP)
    """Play button handler."""
    if self.state == self.READY:
        threading.Thread(target=self.listenRtp).start()
        self.playEvent = threading.Event()
        self.playEvent.clear()
        self.sendRtspRequest(self.PLAY)
```

9.3 Method DESCRIBE

In addition to the RTSP and PAUSE interactions, DESCRIBE is used to transmit information about the video stream. When the server receives the DESCRIBE request, it sends back a file describing the session - telling the client what type of stream in the session and what encoding was used.
Specifically, when the server sends a response to the client, there are statistics about:

- RTSP port of the server.
- Video encoding of video transmitted over RTP - RTSP ID of session has been established.

```
def describeMovie(self):
    """Describe button handler."""
    if self.state != self.INIT:
        self.info = True
        self.sendRtspRequest(self.DESCRIBE)
```



```
Data received:
RTSP/1.0 200 OK
CSeq: 4
Session: 778930
Content-Base: movie.Mjpeg
Content-Length: 85
v=0
m=video 101 RTP/AVP 26
a=control:streamid=778930
a=mimetype:string;"video/Mjpeg"
```

10 Source code and References

- Source code: https://github.com/dinhhoanganh2001/asm1_networking.
- References:
  - RTP, https://en.wikipedia.org/wiki/Real-time_Transport_Protocol
  - RTSP, https://en.wikipedia.org/wiki/Real_Time_Streaming_Protocol