# Computer Networks

# Socket Programming: Video Streaming with RTSP and RTP

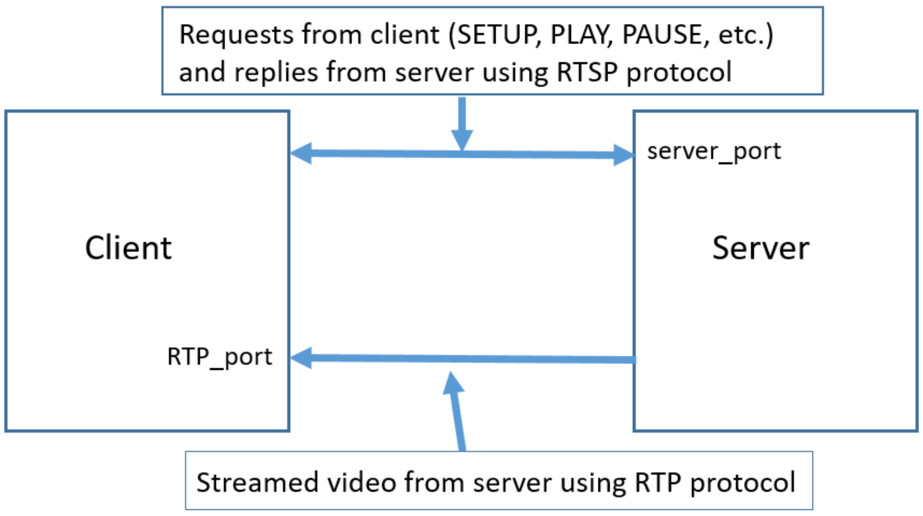
In this project, you will implement a streaming video server and client that communicate using the Real-Time Streaming Protocol (RTSP) and send data using the Real-time Transfer Protocol (RTP). Your task is to implement the RTSP protocol in the client and implement the RTP packetization in the server.

We will provide you code that implements the RTSP protocol in the server, the RTP de-packetization in the client, and takes care of displaying the transmitted video. If you prefer, you may propose to write your code in another programming language than Python, but that would have to be approved.

## 1. System overview

The video to be streamed resides on the server. The server listens at port server_port for requests from clients. When a client wants to receive the streamed video, the client sends the appropriate requests to the server and indicates to the server at which port it wants to receive the video. That port is RTP_port. The server replies to the request and sends the streamed video in RTP packets to port RTP_port.

The request/replies use the RTSP protocol, while the streamed video is carried using the RTP protocol.


Requests from client (SETUP, PLAY, PAUSE, etc.) and replies from server using RTSP protocol
server_port
Client
Server
RTP_port
Streamed video from server using RTP protocol

## 2. Code

### Client, ClientLauncher

The ClientLauncher starts the Client and the user interface which you use to send RTSP commands and which is used to display the video. In the Client class, you will need to implement the actions that are taken when the buttons are pressed. You do not need to modify the ClientLauncher module.

### ServerWorker, Server

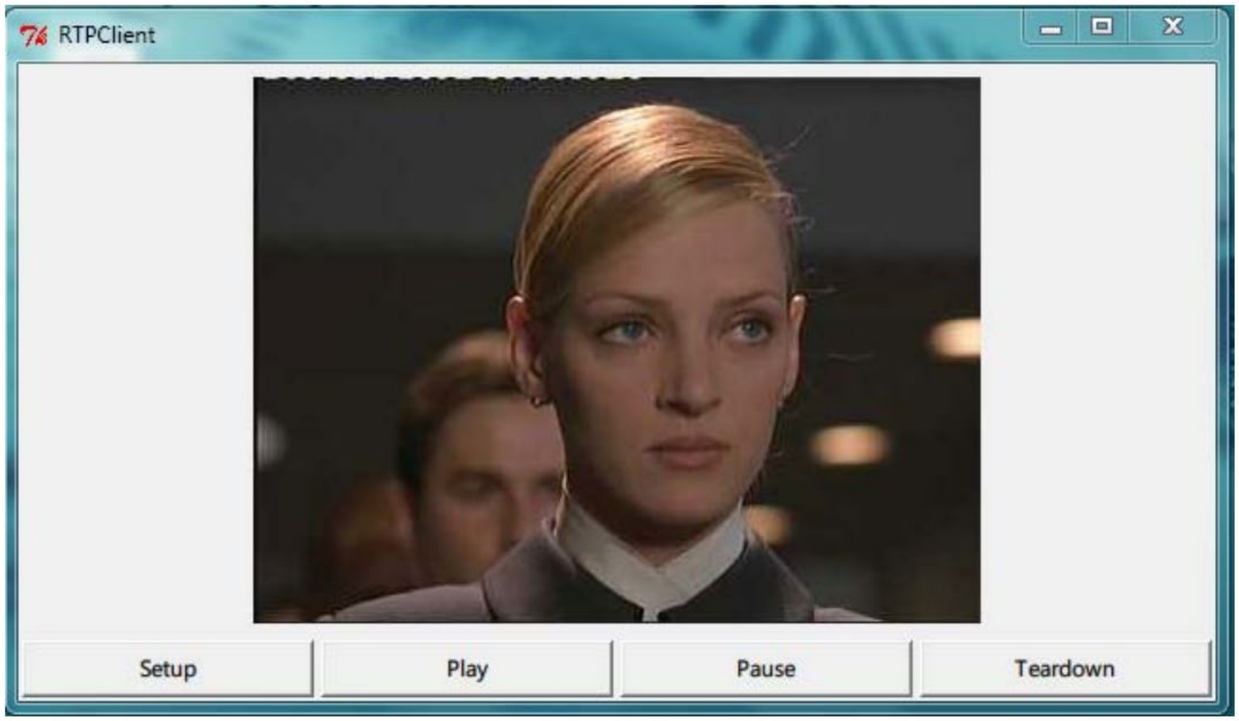
These two modules implement the server which responds to the RTSP requests and streams back the video. The RTSP interaction is already implemented and the ServerWorker calls methods from the RtpPacket class to packetize the video data. You do not need to modify these modules.

**RtpPacket**
This class is used to handle the RTP packets. It has separate methods for handling the received packets at the client side and you do not need to modify them. The Client also de-packetizes (decodes) the data and you do not need to modify this method. You will need to complete the implementation of video data RTP packetization (which is used by the server).

**VideoStream**
This class is used to read video data from the file on disk. You do not need to modify this class.

## 3. Running the code

After completing the code, you can run it as follows:
First, start the server with the command
```
python  Server.py  server_port
```
where server_port is the port your server listens to for incoming RTSP connections. The standard RTSP port is 554, but you will need to choose a port number greater than 1024.
Then, start the client with the command
```
python ClientLauncher.py   server_host   server_port   RTP_port   video_file
```
where server_host is the name of the machine where the server is running, server_port is the port where the server is listening on, RTP_port is the port where the RTP packets are received, and video_file is the name of the video file you want to request (we have provided one example file movie.Mjpeg). The file format is described in Appendix section.
The client opens a connection to the server and pops up a window like this:



You can send RTSP commands to the server by pressing the buttons. A normal RTSP interaction goes as follows:

1. The client sends SETUP. This command is used to set up the session and transport parameters.
2. The client sends PLAY. This command starts the playback.

3. The client may send PAUSE if it wants to pause during playback.
4. The client sends TEARDOWN. This command terminates the session and closes the connection.

The server always replies to all the messages that the client sends. The code 200 means that the request was successful while the codes 404 and 500 represent FILE_NOT_FOUND error and connection error respectively. In this project, you do not need to implement any other reply codes. For more information about RTSP, please see RFC 2326.

## 4. The Client

Your first task is to implement the RTSP protocol on the client side. To do this, you need to complete the functions that are called when the user clicks on the buttons on the user interface. You will need to implement the actions for the following request types. When the client starts, it also opens the RTSP socket to the server. Use this socket for sending all RTSP
requests.

### SETUP

• Send SETUP request to the server. You will need to insert the Transport header in which you specify the port for the RTP data socket you just created.
• Read the server's response and parse the Session header (from the response) to get the RTSP session ID.
• Create a datagram socket for receiving RTP data and set the timeout on the socket to 0.5 seconds.

### PLAY

• Send PLAY request. You must insert the Session header and use the session ID returned in the SETUP response. You must not put the Transport header in this request.
• Read the server's response.

### PAUSE

• Send PAUSE request. You must insert the Session header and use the session ID returned in the SETUP response. You must not put the Transport header in this request.
• Read the server's response.

### TEARDOWN

• Send TEARDOWN request. You must insert the Session header and use the session ID returned in the SETUP response. You must not put the Transport header in this request.
• Read the server's response.

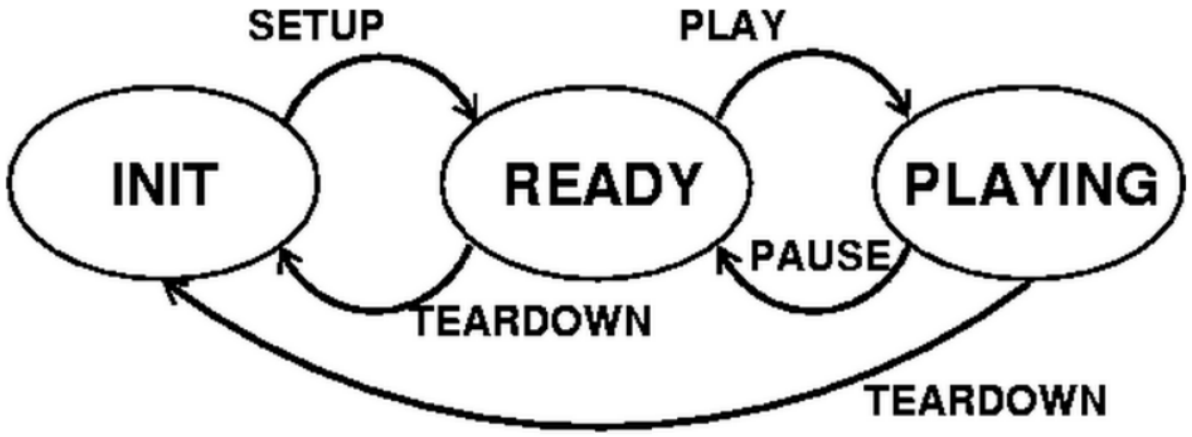
*Note: You must insert the CSeq header in every request you send. The value of the CSeq header is a number which starts at 1 and is incremented by one for each request you send.*

**Example**
Here is a sample interaction between the client and server. The client's requests are marked with C: and server's replies with S:. In this project both the client and the server do not use sophisticated parsing methods, and they expect the header fields to be in the order you see below.

```
C: SETUP movie.Mjpeg RTSP/1.0
C: CSeq: 1
C: Transport: RTP/UDP; client_port= 25000

S: RTSP/1.0 200 OK
S: CSeq: 1
S: Session: 123456
```

```
C: PLAY movie.Mjpeg RTSP/1.0
C: CSeq: 2
C: Session: 123456

S: RTSP/1.0 200 OK
S: CSeq: 2
S: Session: 123456

C: PAUSE movie.Mjpeg RTSP/1.0
C: CSeq: 3
C: Session: 123456

S: RTSP/1.0 200 OK
S: CSeq: 3
S: Session: 123456

C: PLAY movie.Mjpeg RTSP/1.0
C: CSeq: 4
C: Session: 123456

S: RTSP/1.0 200 OK
S: CSeq: 4
S: Session: 123456

C: TEARDOWN movie.Mjpeg RTSP/1.0
C: CSeq: 5
C: Session: 123456

S: RTSP/1.0 200 OK
S: CSeq: 5
S: Session: 123456
```

**Client State**
One of the key differences between HTTP and RTSP is that in RTSP each session has a state. In this project you will need to keep the client's state up-to-date. Client changes state when it receives a reply from the server according to the following state diagram.



## 5. The Server

On the server side, you will need to implement the packetization of the video data into RTP packets. You will need to create the packet, set the fields in the packet header and copy the payload (i.e., one video frame) into the packet.

When the server receives the PLAY-request from the client, the server reads one video frame from the file and creates an RtpPacket-object which is the RTP-encapsulation of the video frame. It then sends the frame to the client over UDP every 50 milliseconds.

For the encapsulation, the server calls the encode function of the RtpPacket class. Your task is to write this function. You will need to do the following: (the letters in parenthesis refer to the fields in the RTP packet format below).

• Set the RTP-version field (V). You must set this to 2.
• Set padding (P), extension (X), number of contributing sources (CC), and marker (M) fields.
These are all set to zero in this project.
• Set payload type field (PT). In this project we use MJPEG and the type for that is 26.
• Set the sequence number. The server gives this the sequence number as the frameNbr argument to the encode function.
• Set the timestamp using the Python's time module.
• Set the source identifier (SSRC). This field identifies the server. You can pick any integer value you like.
• Because we have no other contributing sources (field CC == 0), the CSRC-field does not exist.
The length of the packet header is therefore 12 bytes, or the first three lines from the diagram below.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|V=2|P|X|  CC   |M|     PT      |       sequence number         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                           timestamp                           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|           synchronization source (SSRC) identifier            |
+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
|            contributing source (CSRC) identifiers             |
|                             ....                              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

You must fill in the header fields in the header bytearray of the RtpPacket class. You will also need to copy the payload (given as argument data) to the RtpPacket's payload data field.

The above diagram is in the network byte order (also known as big-endian). Python uses the same byte order, so you do not need to transform your packet header into the network byte order.
For more details on RTP, please see RFC 1889.

**Twiddling the Bits**
Here are some examples on how to set and check individual bits or groups of bits. Note that in the RTP packet header format smaller bit-numbers refer to higher order bits, that is, bit number 0 of a byte is 2^7 and bit number 7 is 1 (or 2^0). In the examples below, the bit numbers refer to the numbers in the above diagram.

Because the header-field of the RtpPacket class is of type bytearray, you will need to set the header one byte at a time, that is, in groups of 8 bits. The first byte has bits 0-7, the second byte has bits 8-15, and so on.

To set bit number n in variable mybyte of type byte:
mybyte = mybyte | 1 << (7 - n)

To set bits n and n + 1 to the value of foo in variable mybyte:
mybyte = mybyte | foo << (7 - n)
Note that foo must have a value that can be expressed with 2 bits, that is, 0, 1, 2, or 3.

To copy a 16-bit integer foo into 2 bytes, b1 and b2:
b1 = (foo >> 8) & 0xFF
b2 = foo & 0xFF

After this, b1 will have the 8 high-order bits of foo and b2 will have the 8 low-order bits of foo.
You can copy a 32-bit integer into 4 bytes in a similar way.
**Bit Example**
Suppose we want to fill in the first byte of the RTP packet header with the following values:
V = 2
P = 0
X = 0
CC = 3

In binary this would be represented as
1 0 | 0 | 0 | 0 0 1 1
V=2 P X CC = 3
2^7 . . . . . . . . 2^0

## 6. Required Extensions

In addition to the above, you are required to implement the following.

### Statistics Analysis

Calculate statistics about the session. Statistics include RTP packet loss rate, video data rate (in bits or bytes per second), jitter and any other interesting statistics you can think of. To that end, you need to analyze the code and insert the necessary hooks to collect the data. You will need to implement a process emulating the network impairments with settable RTP packet loss ratio and jitter.

### Three-Button User Interface

The user interface on the RTPClient has 4 buttons for the 4 actions. If you compare this to a standard media player, such as RealPlayer or Windows Media Player, you can see that they have only 3 buttons for the same actions: PLAY, PAUSE, and STOP (roughly corresponding to TEARDOWN). There is no SETUP button available to the user. Given that SETUP is mandatory in an RTSP-interaction, how would you implement that in a media player? When does the client send the SETUP? Come up with a solution and implement it. Also, is it appropriate to send TEARDOWN when the user clicks on the STOP button?

## 7. Display of Statistics

Your code must display the following statistics. You may display them at the end of the session (when the user hits TEARDOWN/STOP), or whenever the user hits PAUSE, or continously.

- Number of lost packets
- Number of packets received

- Packet loss rate = (number of lost packets)/(total number of packets) - (should be close to the set % of packet loss)
- Number of bytes received
- Play time: Should not include the times during which the playback is paused
- Video data rate = (number of bytes received)/(play time)
- TotalJitter, as defined in the Implementation Notes below
- AverageJitter, as defined in the Implementation Notes below

## 8. Implementation Notes

### Packet Loss Rate

Use a random nunber generator at the server to decide whether an outgoing packet should be artificially dropped. The client detects a packet loss if there is a gap in the sequence number. The settable parameter is the % of packets lost.

### Video Data Rate

Count the number of bytes received and divide by the play time. The play time should not include the time periods during which the video is not played. The clock should be started when the user hits PLAY, suspended when the user hits PAUSE/TEARDOWN/STOP and resumed when the user hits PLAY.

### Jitter

To simulate jitter, sleep the thread at the server for randJitter msec, where randJitter is an integer randomly chosen from [0, MAX_JITTER] for each packet. The settable parameter is MAX_JITTER.

At the client, calculate the jitter as follows. Initialize totalJitter = 0

Define interPacketSpacing(n) = arrivalTimeOfPacket(n) – arrivalTimeOfPreviousPacket

When packet(n) is received, calculate interPacketSpacing (n), and update

totalJitter = totalJitter + jitterIncrement,

where jitterIncrement = abs(interPacketSpacing (n) - interPacketSpacing (n-1)), where abs() is the absolute value

The averageJitter is calculated as totalJitter/(number_of_packets)

<u>Notes</u>:
interPacketSpacing (n) is not valid and should not be calculated if packet(n) is received after a packet loss. That is, if RTP sequence number of packet(n) > RTP sequence number of packet(n-1) + 1

interPacketSpacing (n) is not valid and should not be calculated if packet(n) is the first packet received after the user hits PLAY.

jitterIncrement = abs(interPacketSpacing (n) - interPacketSpacing (n-1)) should be calculated only if interPacketSpacing (n) and interPacketSpacing (n-1) are valid and have been calculated. totalJitter should be incremented only in that case.

The granularity of the time measurements should be at least to the msec. You may use datetime.now() or a similar method to get the arrival times. datetime.now() gives time down to the microsecond. However, you need to account for the possible microsecond counter wrap around when you calculate jitterIncrement.

## 9. Validation Scenarios

In order to validate your code, use the following scenarios.

## Four-Button User Interface

### a. Successful operation of all buttons

| Step | Action | Expected behavior |
|---|---|---|
| 0 | Start with both client and server not connected | |
| 1 | Start the server and the client | The client must open a connection to the server and a window with the 4 button options available must be displayed once the connection is established successfully. |
| 2 | User clicks on the SETUP button | Client sends SETUP to server. The server must respond with OK if the request is successful and send the RTSP session ID in the response. The correct CSeq header and transport header must be sent by the client. |
| 3 | User clicks on the PLAY button | Client sends PLAY to server. The server must read one frame at a time, create an RTP-encapsulation of the frame and send it to the client over UDP. The correct CSeq header and session ID must be seen in the client's request and the server's response. |
| 4 | User clicks on the PAUSE button | Client sends PAUSE to server. The server must stop sending any frames to the client but the connection must be open and the file should pause after the most recent frame was recent. The correct CSeq header and session ID must be seen in the client's request and the server's response. |
| 5 | User clicks on the PLAY button after the PAUSE button | Client sends PLAY to server. The server must resume reading frames from where it was paused earlier and start sending one frame at a time with the encapsulation of each frame. The correct CSeq header and session ID must be seen in the client's request and the server's response. |
| 6 | User clicks on TEARDOWN button after PAUSE/PLAY | Client sends TEARDOWN to server. The server must close the connection to the client immediately upon receiving this request. |

### b. Error scenarios

| Step | Action | Expected Behavior |
|---|---|---|
| 0 | Start with steps 0 to 2 of the successful scenario but with a different executable server (Build another version of the executable with a hard coded incorrect file name) | The server should not be able to find the file requested, and hence will send back an error message with code 404 to the client and the client should then display this to the user. |

## Three-Button User Interface

As part of this assignment, you have to develop the validation scenario(s) for the 3-button interface.

## 10. What to Turn In

1. A proposed action plan by the due date specified in “ProjectTimeline” in “Projects Overview”.

2. A proposed outline of the research paper for approval by the due date specified in “ProjectTimeline” in “Projects Overview”.

3. A team report by date in “ProjectTimeline” in “Projects Overview”.

   a) Including the complete code for the four-button user interface and the three-button user interface. The complete code includes the pieces you modified or wrote and the pieces you did not have to modify. Include a README file that describes how to compile/run the program.
   b) Providing a background summary of RTP and RTSP and how they are utilized in streaming
   c) Describing your hardware setup and configuration
   d) Including the design document of your code (e.g. Protocol State Diagrams, Sequence Diagram or any other info that helps to understand what has been done). The document should describe the various design choices you have made regarding how to implement the 3-button interface, how to implement the settable packet loss/jitter at the server and how to measure the loss rate, jitter and video data rate at the client.
   e) Providing screenshots showing the user interface, three-button and four-button
   f) Providing results of the statistics analysis about the session: RTP packet loss rate, etc.
   g) Describing what issues, if any, the team encountered during the project, how the team overcame the issues and what the team learned from the project. You can also provide suggestions on how the projects in Computer Networks could be improved in the future.
   h) Including a video clip to demo the code running

4. Individual reports, one for each team member by the due date specified in “ProjectTimeline in “Projects Overview””. The individual report is confidential and not shared with the other team members.

   a) If you, as an individual team member, have anything specific to add to 1.f) in the team report, please do it in your individual report. Describe what issues, if any, you, as an individual team member, encountered during the project, how you overcame the issues and what you learned from the project (this is not necessarily just about the topic, could be related to teamwork, etc.). You can also provide suggestions on how the projects in Computer Networks could be improved in the future. This complements the team report with any individual viewpoint not included in the team report.
   b) Describe what each team member (including yourself) did and contributed to the project, and assign a numerical score from 1 to 10 to each team member, including yourself. 1 is the poorest, and 10 is the best..

Note: There will be a separate session (taking place outside of lecture hours) for you to demo your code is running and answer questions about your code and design. The code demo sessions will take place towards the end of the semester.

## 11. Grading criteria

General Note: The following are criteria used to come up with a team grade. Your final individual project score is not necessarily the team grade, it may be flexed up or down, depending on your individual contribution to the team and the quality of your individual report.

### Coding (60%)

Source code of your well-structured and well-documented program. Include comments on your codes for clarification. In addition, your project has to meet the validation scenarios. Here is the weight detail for each part:

4-button interface: 30%   Statistical analysis: 10% . For each of these, **you should be able to demonstrate good understanding of the code and be able to answer specific questions on the code and design**.

3-button interface and student developed validation scenarios: 20%. For each of these, **you should be able to demonstrate good understanding of the code and be able to answer specific questions on the code and design.**

### Documents (40%)

**Group Report (40%)**
The group report will be evaluated not only on its content, but also the professionalism of its appearance.

## 12. Appendix

1. Project's proprietary MJPEG (Motion JPEG) format
In this project, the server streams a video which has been encoded into a proprietary MJPEG file format. This format stores the video as concatenated JPEG-encoded images, with each image being preceded by a 5-Byte header which indicates the bit size of the image. The server parses the bitstream of the MJPEG file to extract the JPEG images on the fly. The server sends the images to the client at periodic intervals. The client then displays the individual JPEG images as they arrive from the server.