NOVA SCHOOL OF
SCIENCE & TECHNOLOGY
DEPARTAMENTO DE INFORMÁTICA

# Streaming mJPEG over RTP

## (TPC3 – Redes de Computadores 2024/2025)

In this assignment you will implement a streaming video server and a client that coordinate their actions through a TCP socket and that exchange data using the Real-time Transfer Protocol (RTP); we will ignore the possibility of UDP datagram loss.

The application is composed by two processes:

- **Server**: that stores the video files in its disk and sends their contents in RTP packets
- **Client**: that will ask for a given file and, in case of success, will receive the RTP packets and play the file in a window using the TkInter software.

The window created by the client is as shown below:


The video to be displayed is composed by a sequence of isolated JPEG Images; this format is close to MJPEG (Multiple JPEG). The file used to test is called *movie.mpeg* and is in a proprietary format. It is a binary file with a sequence of parts, where each part is composed by:

- 5 bytes codifying an integer NB
- The codification of the JPEG image itself contained in NB bytes

You will implement a streaming video server and a client that coordinate their actions through a TCP socket and that exchange data using the Real-time Transfer Protocol (RTP); we will ignore the possibility of UDP datagram loss.

## 2. Actions of server and client

The server will be started with the command:

```
python3 Server.py TCPControlPort
```

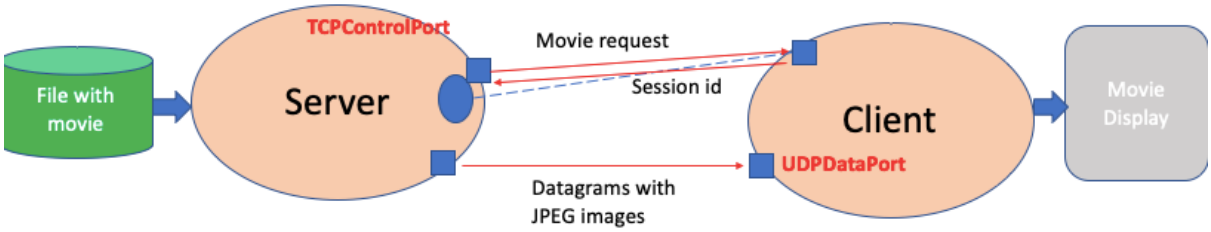
where *TCPControlPort* is the port where the server expects a video request.

the client with be stated with the command

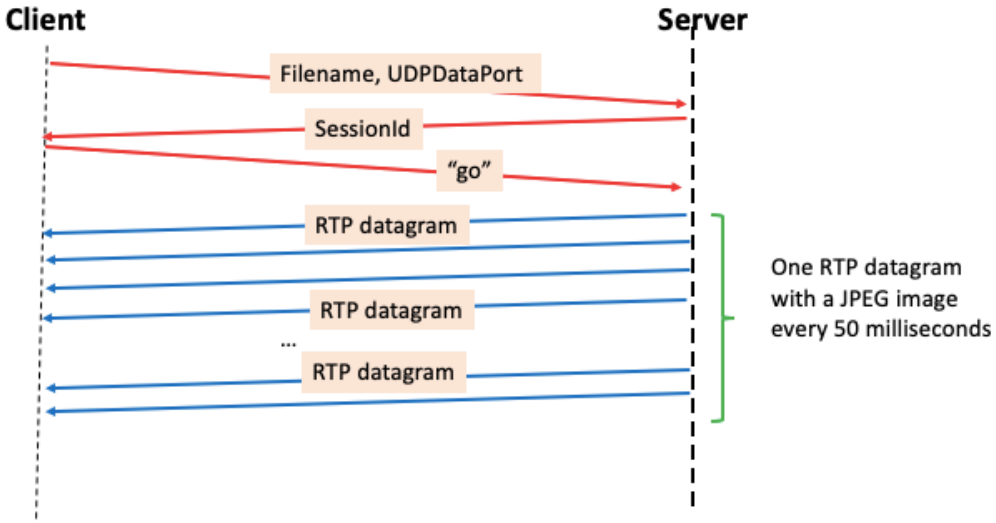
```
python3 Client.py hostname TCPControlPort UDPDataPort fileName
```

where (*hostname*, *TCPControlPort)* defines the location of the server's TCP endpoint, *UDPDataPort* is the UDP port where the client will receive the datagrams containing the video and *filename* is the name of the file in the server containing the video that will be played.

The following picture gives an overview of the interaction between server and client.




Here, we show all the messages exchanged between client and server on a successful display




The video display has two phases:

1. **Setup** – The client connects to the TCPControlPort and sends a message with `filename`, UDP`DataPort`, where f*ilename* is a string and *UDP`DataPort`* is an integer. If the sender has the file, it will answer with a message `with a SessionID` that is a random `number;` otherwise it

will reply with a `SessionId` of -1. Client replies with a message containing the string *"Go"* authorizing the server to start the sending of the movie.

2. **Transfer** – The sender will send the file in several UDP Datagrams. Each UDP datagram contains:

    a. A RTP Header described below
    b. A complete JPEG file

The datagrams are sent at rate of 20 datagrams per second.

## 3. The RTP header

The following picture shows the relevant parts of the RTP packet header:

**RTP packet header**

| bit offset | 0-1 | 2 | 3 | 4-7 | 8 | 9-15 | 16-31 |
|---|---|---|---|---|---|---|---|
| 0 | Version | P | X | CC | M | PT | Sequence Number |
| 32 | Timestamp | | | | | | |
| 64 | SSRC identifier | | | | | | |

In each RTP packet generated by the sender, the following fields must be filled

| | |
|---|---|
| RTP Version | 2 |
| Padding(P)<br>extension(X),<br>number of contributing sources (CC)<br>marker(M) | 0<br>0<br>0<br>0 |
| Payload type (PT) | 26 (MJPEG) |
| Sequence number | Starts at 1 and is incremented each time an image is sent |
| Timestamp | Use time stamp of Python's time module |
| SRSC Identifier | The SessionID obtained in Setup phase should be inserted here |

The client should perform the following checks in the RTP header:

- If the contents of RTP Version, P, X., CC, M, PT are the values expected
- If the sequence number is OK; this means that it should be greater than the last sequence number received
- If the Timestamp is reasonable
- If the SRSC value is the SessionId

**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 V=2, P=0, X=0 , CC = 3. In binary this would correspond to the following table

| Bits 7,6 | Bit 5 | Bit 4 | Bits 3- 0 |
|---|---|---|---|
| 1 0 | 0 | 0 | 0 0 1 1 |
| V=2 | P=0 | X=0 | CC=3 |

## 4. To do

Your task is to complete the code of two Python programs called *Client.py* and *Server.*py. Analyze the code files available in clip. You must complete the code.

***Client.py*** You have to fill the blanks corresponding to:

- Verification of RTP header contents as above.
- Writing the payload of the datagram in a temporary file

**Sender.py** You have to fill the blanks corresponding to the loop that:

- Fills an UDP datagram with:
  - The RTP Header
  - The payload that contains a JPEG file
- Sends the datagram

## 5. Delivery

The delivery should be done **before 11:00 on November 13, 2024**. The submission has two parts:

- a Google form containing the identification of the students that submit the work and questions about the functionality of the code
- The code developed will be uploaded through Moodle

Details will be sent later.