ASSIGNMENT 1

Course: Computer Network, Semester 2, 2020-2021

Objectives

We 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).

Requirements (8 points)

We will provide you code (*used as reference, you can modify it according to your needs*) that implements the RTSP protocol in the **server**, the RTP de-packetization in the client, and takes care of simply displaying the transmitted video.

Your tasks are to implement

- **the RTSP protocol in the client (named as Client.py),**
- **the RTP packetization in the server (named as RtpPacket.py),**
- **the customized interactive player for displaying the transmitted video.**

Description

Client, ClientLauncher

The ClientLauncher starts the Client (*to be implemented*) 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 be implemented*) 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 RTPpacketization (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.

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 lab, you do not need to implement any other reply codes. For more information about RTSP, please see RFC 2326.

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 assignment, 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
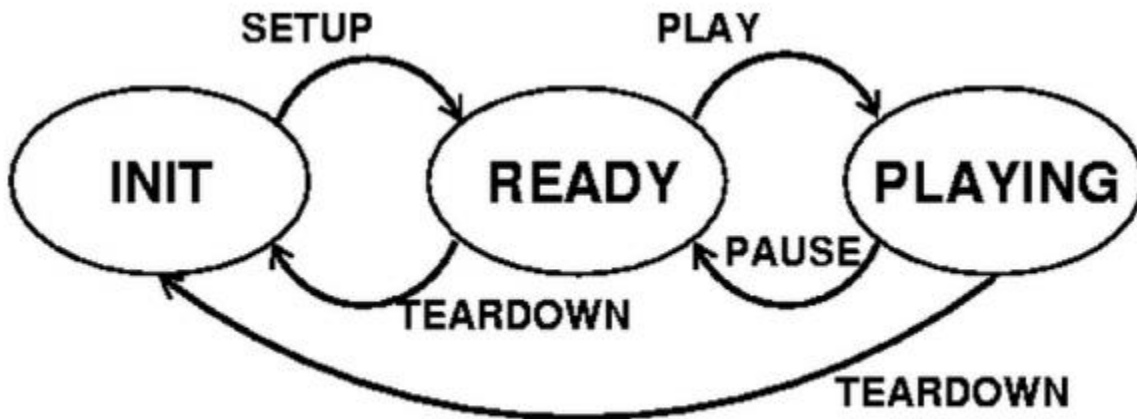C: CSeq: 5
C: Session: 123456

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 assignment, 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.

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 a 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 lab.

• Set payload type field (PT). In this lab 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             |
|                             ....                              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```
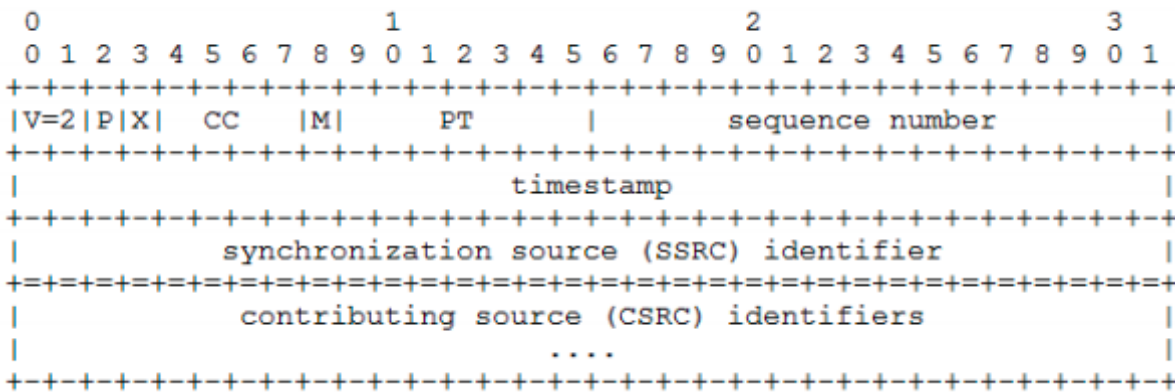
You must fill in the header fields in the header bytearray of the RtpPacket class. You will also need to copy the payload (given as argument data) to the RtpPacket's payload data field. The above diagram is in the network byte order (also known as big-endian). Python uses the same byte order, so you 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

Appendix

Assignment's proprietary MJPEG (Motion JPEG) format

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.

Extend (2 points)

1. Calculate the statistics about the session. You will need to calculate RTP packet loss rate, video data rate (in bits or bytes per second), and any other interesting statistics that you can think of.

2. 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?

3. Currently, the client and server only implement the minimum necessary RTSP interactions and PAUSE. Implement the method DESCRIBE which is used to pass information about the media stream. When the server receives a DESCRIBE-request, it sends back a session description file which tells the client what kinds of streams are in the session and what encodings are used.

4. Implement some additional functions for user interface such as: display video total time and remaining time, fast forward or backward video (or make a scroll bar for scrolling video if you can).

5. Add one more state to the client (for example SWITCH state) so that user can select another video from a list of videos received from server.

For *major student program*, you can embed your multimedia player into a Browser of choice. Test the use of this player by connecting to a video source.

Report

Hard copy (*needed for presentation and demo time*) and softcopy (*to be submitted to BKeL*) including

- Analysis of problem requirements
- Description of different functions for your application
- List of components
- Model and data flow
- Class diagram
- Implementation
- A summative evaluation of achieved results
- User manual
- Full source code (softcopy)

Group work

Maybe 2 or 3 or 4 students depending on the total number of students in Lab class. The Lab instructor will organize and announce the group member list.

Deadline

Sunday, November 14th, 2021.