

VIETNAM NATIONAL UNIVERSITY - HCM
UNIVERSITY OF TECHNOLOGY
Faculty of Computer Science and Computer Engineering



COMPUTER NETWORK

Report

Real-Time Streaming Protocol

LECTURER : Nguyễn Lê Duy Lai
STUDENTS : Lâm Trịnh Long 1811044
 Nguyễn Khắc Hào 1852346

HO CHI MINH CITY, 11/2020



Contents

1	Requirement Analysis	2
1.1	Modules requirement	2
1.2	RtpPacketization	2
2	Functions Description	3
2.1	RTPPacket	3
2.2	Server / ServerWorker	3
2.3	Client / ClientLauncher :	4
3	Class diagram	6
4	A summative evaluation of achieved results	7
5	User Manual	9
6	Improving the implementation	9
6.1	Simplifying the SETUP button	9
6.2	File information, video Backward and Replay	10
6.2.1	VideoStream.py	10
6.2.2	ServerWorker.py	11
6.2.3	Client.py	11
6.3	Switching the video currently being streamed	11
6.4	Summary	11
6.5	Architectural overview	12



1 Requirement Analysis

This assignment is focused on implementing a video streaming service. The server and client communicate using Real-Time Streaming Protocol (RTSP), send video data using Real-time Transfer Protocol (RTP).

1.1 Modules requirement

Module	Requirements
ClientLauncher	Receive input of ip address, RTP port number, the server port number and the video file name. Send those data to the Client module.
Client	Establishing a connection to the server. Send and receive RTP packets from the server. Encode the RTSP requests for the Server while decode RTP packets from the server into usable video data. Show the video feed after being decoded.
Server	Receive a given port number. Open a socket and listen to client requests. Send requests to ServerWorker module for further processing.
ServerWorker	Receive clients' requests and response to them. Encode video data into RTP packets and send them to the client.
RTPPacket	Encode various metadata on the transfer protocol, the video, the server into the packet header (detailed below), and every single frame from the video into the payload for every packet.

1.2 RtpPacketization

Before being sent the data must first be fitted into the RTP packet format:

```

      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                |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|                               data payload                          |
|                               ....                                  |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
```

For this assignment we had:

- RTP version field $V = 2$
- Padding (P), extension (X), number of contributing sources (CC), and marker (M) field, all of these set to zero



- Payload type field(PT) = 26 for MJPEG
- Sequence number
- Timestamp
- Synchronization source identifier code(not important in this assignment because we only have 1 server)
- The video frame payload itself

2 Functions Description

2.1 RTPPacket

This module is solely for encoding and decoding messages into the RTP Packet format above for transporting.

2.2 Server / ServerWorker

The server software is capable of opening a socket to listen to clients and receive request packets through the use of RTSP/TCP session. At startup, the server will open up a socket at a certain port to start listening for any client

Then, the server is in charge of responding to various requests that might be sent to it. All of these responses include a sequence number, and a session ID for identifying between multiple sessions (i.e. multiple clients at once):

- When the server receive the SETUP-request from the client, the server start preparing for the video stream, getting the video's filename from the request. Following that, the server will response with a status code 200-OK if the video file is found and all is ready to start, status code 404-FILE_NOT_FOUND if the video file is not found and status code 500-CONNECTION_ERROR if there is problem in the connection. Including a session ID at the end.
- When the server receive the PLAY-request from the client, the server read 1 video frame from the file, create an RTPPacket carrying that video frame as payload with a header with metadata corresponding to the current Server-Client session. This repeats once every 50 milliseconds.
- When the server receive the PAUSE-request from the client, the server will stop the video frame packetization and stop creating RTPPackets for the client until further request.
- When the server receive the TEARDOWN-request from the client, the server will first send an ACK message, after which it close up the RSP socket and cease all connection to that client.

2.3 Client / ClientLauncher :

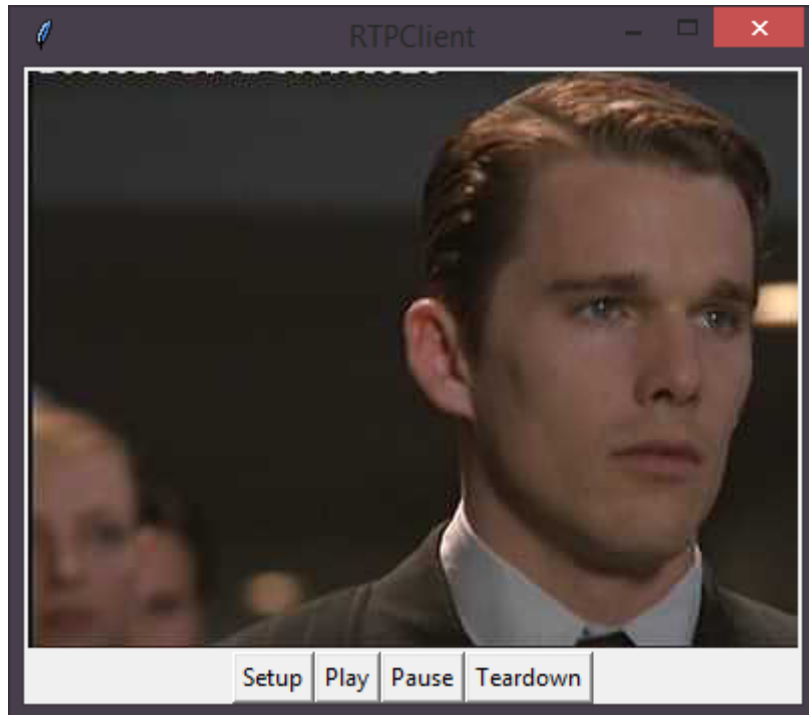


Figure 1: The client interface

When using the client, the user will see a small GUI with various buttons, each buttons have the following functions:

Setup :

- Send SETUP request to the server. In which contains the Transport header that 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. Insert the Session header and use the session ID acquired in the SETUP response.
- Read the server's response
- Listen to RTP packets from the server containing video frames, parse the video frames into motion video feed.

Pause :

- Send PAUSE request with session ID
- Read the server's response
- Halts the video feed

Teardown :

- Send TEARDOWN request with session ID
- Read the server's response
- Closes all connections and exit the client

The Client has various states throughout the video streaming session, according to what it requests and the server's response:

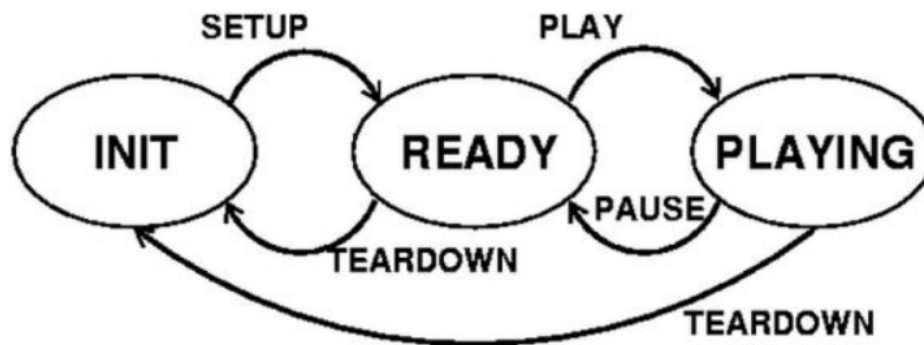


Figure 2: States of the client

3 Class diagram

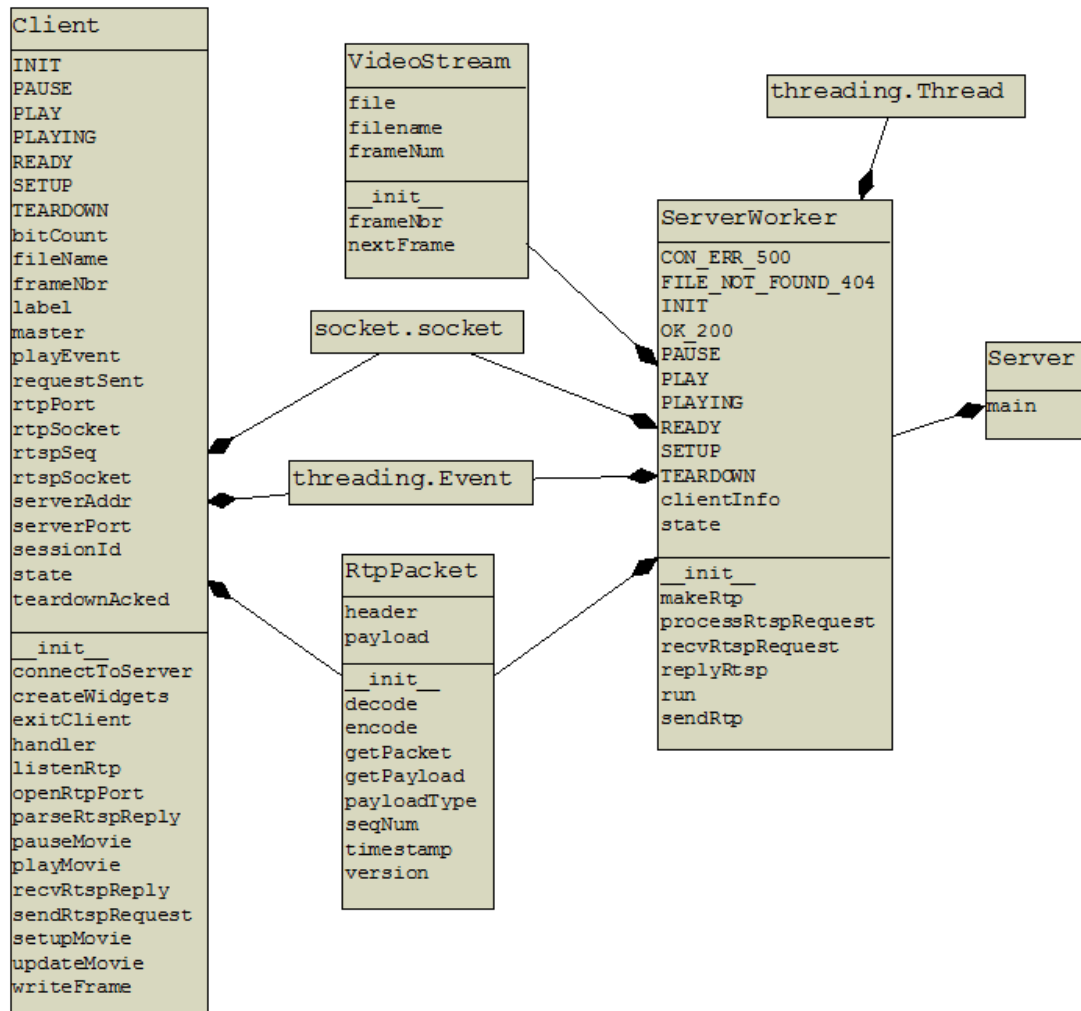


Figure 3: The class diagram

Here we can see the whole software hierarchy of classes and how each class composes of other classes, to function as a video streaming service.

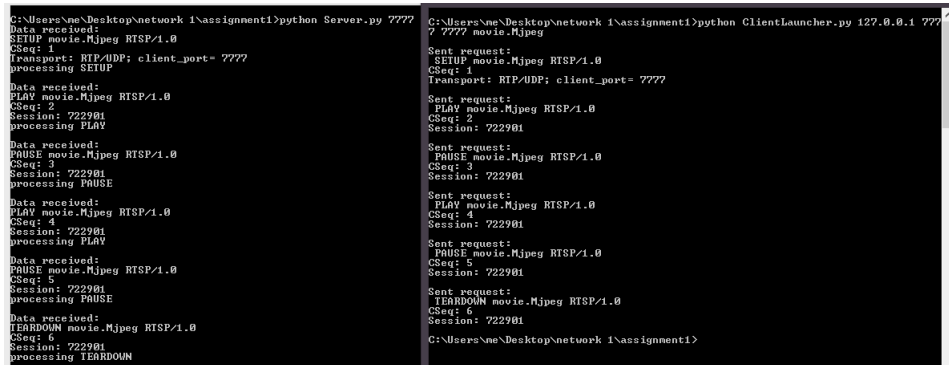
The server and client uses a socket object to communicate between each other.

The **ServerWorker** utilize **VideoStream** module to get video frames, packetize them via **RtpPacket** module, then the completed pack is sent to the **Client**.

The client that receive the packet then decode it into video frame and update the newest frame into the video feed.

4 A summative evaluation of achieved results

Request-reply and connection test:



```
C:\Users\me\Desktop\network\1\assignment1>python Server.py 7777
Data received:
SETUP movie.Mjpeg RTSP/1.0
CSeq: 1
Transport: RTP/UDP; client_port= 7777
processing SETUP
Data received:
PLAY movie.Mjpeg RTSP/1.0
CSeq: 2
Session: 722901
processing PLAY
Data received:
PAUSE movie.Mjpeg RTSP/1.0
CSeq: 3
Session: 722901
processing PAUSE
Data received:
PLAY movie.Mjpeg RTSP/1.0
CSeq: 4
Session: 722901
processing PLAY
Data received:
PAUSE movie.Mjpeg RTSP/1.0
CSeq: 5
Session: 722901
processing PAUSE
Data received:
TEARDOWN movie.Mjpeg RTSP/1.0
CSeq: 6
Session: 722901
processing TEARDOWN

C:\Users\me\Desktop\network\1\assignment1>python ClientLauncher.py 127.0.0.1 7777
7 7777 movie.Mjpeg
Sent request:
SETUP movie.Mjpeg RTSP/1.0
CSeq: 1
Transport: RTP/UDP; client_port= 7777
Sent request:
PLAY movie.Mjpeg RTSP/1.0
CSeq: 2
Session: 722901
Sent request:
PAUSE movie.Mjpeg RTSP/1.0
CSeq: 3
Session: 722901
Sent request:
PLAY movie.Mjpeg RTSP/1.0
CSeq: 4
Session: 722901
Sent request:
PAUSE movie.Mjpeg RTSP/1.0
CSeq: 5
Session: 722901
Sent request:
TEARDOWN movie.Mjpeg RTSP/1.0
CSeq: 6
Session: 722901
C:\Users\me\Desktop\network\1\assignment1>
```

Figure 4: A simple run

The server received every request and responses correctly to all of them (otherwise there would be an exception on client-side). And the basic function of video streaming is implemented.

Performance test :

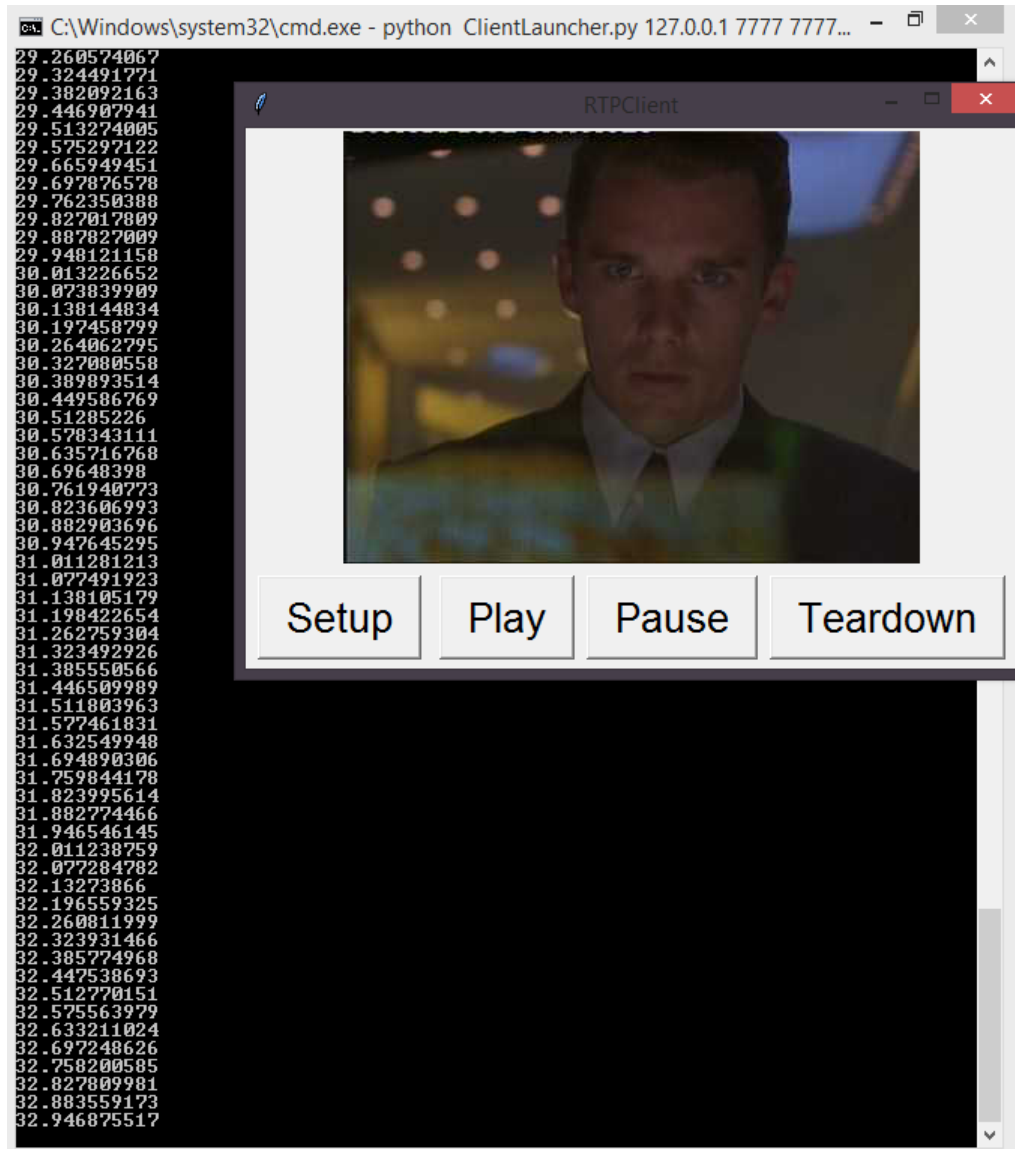


Figure 5: Time print test

Utilizing the *time* library, we ran the command of `print(time.perf_counter())`, placed in `updateMovie()` method of the client, we can get a printout of seconds from the launch of the client to the time a video frame is shown. From which we can calculate that the software averages around 16 frame per second. (Take into consideration that the extra functions for benchmarking the performance did make the software performs slower)



5 User Manual

In order to use this streaming service you need to follow these steps:

- Step 0 (On the server-side only) - Setting up a server :
Run the Server.py and enter a port using this command :

```
$python Server.py <PORT>
```

And give the process needed firewall rights and network rights. After which, the server is now all set for listening and responding to clients.

- Step 1 - Starting up the client: After filling in the need inputs of server IP address, port numbers and file name, start the client using this command:

```
$python ClientLauncher.py <Address> <S_port> <RTP_port> <Video_file>
```

With these corresponding fields:

<Address> The destination server IP address

<S_port> The destination server port number

<RTP_port> The port where the RTP packets will be received

<Video_file> The desired video filename

- Step 2 - Starting the video stream:
In sequence, click on the "Set up" button to initialize the connection, and click on "Play" to start the video on your client
- While playing the video, the user can click on "Pause" to temporary pauses the video. Click on "Teardown" or close the user interface if they wish to stop the video streaming service.

6 Improving the implementation

6.1 Simplifying the SETUP button

Despite SETUP being mandatory in an RTSP interaction, standard media players, such as RealPlayer or Windows Media Player, do not require the user to perform any setup action before being able to play video. This is because **the SETUP process can be performed automatically before the PLAY request is sent**, reducing the number of steps the user have to perform.

In the new implementation, we simply move the setupMovie() function call to the initialization process of the program, at the same time remove the Setup button since the user no longer need to consider about it.

6.2 File information, video Backward and Replay

Currently, the server and the client can only send and display video frames sequentially, and communicates using minimum necessary RTSP interactions and PAUSE. With the new implementation, we try to add a new method that allows the server to send more video information to the client, and communicates more interactively. In order to do this, we needed a new request type. Initially, we tried to make use of RTSP's DESCRIBE request. Unfortunately, due to DESCRIBE being quite complex and sends information which that we don't need:

(Example of an RTSP DESCRIBE reply:)

```
RTSP/1.0 200 OK
CSeq: 312
Date: 23 Jan 1997 15:35:06 GMT
Content-Type: application/sdp
Content-Length: 376
v=0
o=mhandley 2890844526 2890842807 IN IP4 126.16.64.4
s=SDP Seminar
i=A Seminar on the session description protocol
u=http://www.cs.ucl.ac.uk/staff/M.Handley/sdp.03.ps
e=mjh@isi.edu (Mark Handley)
c=IN IP4 224.2.17.12/127
t=2873397496 2873404696
a=recvonly
m=audio 3456 RTP/AVP 0
m=video 2232 RTP/AVP 31
m=whiteboard 32416 UDP WB
a=orient:portrait
```

We chose to implement a custom request instead. This request is called proprietary describe , or PDESCRIBE. An example of a PDESCRIBE reply is as follows:

```
RTSP/1.0 200 OK
CSeq: 312
Encoding-Type: MJPEG(MotionJPEG)
Content-Name: video2.Mjpeg
Content-Size: 15368
Content-Frames: 320
```

PDESCRIBE is much less complex than DESCRIBE, and only sends information we need. Due to PDESCRIBE being a custom request, it can only be used between the client and the server in this implementation.

With PDESCRIBE request defined, we moved on to implement it on our Server and Client. A total of 3 files were modified to support PDESCRIBE:

6.2.1 VideoStream.py

We modified the VideoStream to allow exporting more information of the video than just video frames. The class no longer reads video frames from the file sequentially every time a frame is

requested by the server, instead reads the complete content of the video file into a buffer and sends data from its buffer every time a frame is requested. Because of this, VideoStream can now report the video's size and total frame number **at the beginning of the streaming process**. It also can send any frame with a requested number of the video, which allows fast-forwarding the video. Although we didn't make use of this fast forwarding feature due to time constraints, we did make use of the new VideoStream and PDESCRIBE to implement a progress indicator in the video player.

6.2.2 ServerWorker.py

In addition to SETUP, PLAY, PAUSE and TEARDOWN, the new PDESCRIBE is now added to the Server's list of supported requests. Upon receiving PDESCRIBE request, the server constructs a reply message which contains the following information of the video file:

- Encoding type: The encoding type that the video has been encoded in.
- Video size: Total size of every frame of the video in bytes. (5-byte headers excluded)
- Video frames: total frame count of the video

6.2.3 Client.py

With the VideoStream and the Server now supporting PDESCRIBE, all the Client has to do is send a PDESCRIBE request and read the response. We implemented a new button which allows the user to ask the server about the information of the video file and receive a summary. The Client also sends a DESCRIBE request during the initialization process to gain more information like total video frames, which is necessary to implement features like replaying video and progress indicating.

With the new implementation, the Client's listenRTP process is divided into two independent part: listen for RTP packages and updating frames. The Client saves received frames into buffer and updates video whenever new frame is available. **When the RTP packet receiving process is stopped, the video player can still continue playing video.**

6.3 Switching the video currently being streamed

If desired, the user will be able to request a different movie from the server. This function is implemented as setting up a new session with the server, using a different socket and different session ID, also restarting the sequence number count.

6.4 Summary

In summary, we added the following features to the new implementation:

- No setup required: the player automatically setups upon launch.
- Video description: a new button to view information of the video being streamed.
- Video skipping: skips backward when streaming video
- Video replay: the client can still play video after the streaming session is over
- Progress indicator: shows how many frames is played and how many frames left.
- Video switching: switch to playing any video with a specified name

Here is the new player in action:

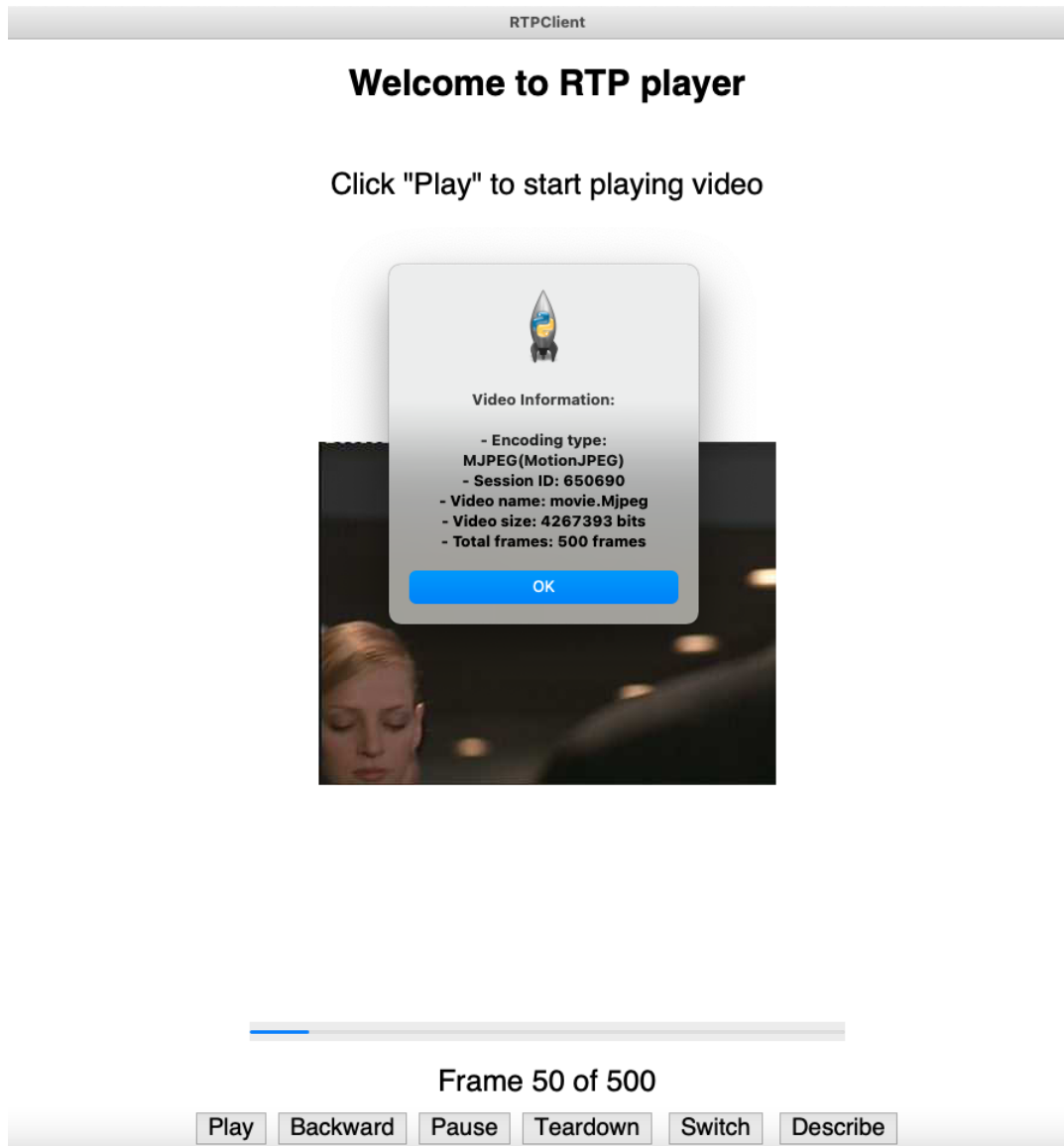


Figure 6: The new player with Describe, Backward and Switch

6.5 Architectural overview

Below are diagrams to better describe the architecture of the new streaming implementation:
How video is passed: The video is read by the VideoStream and stored in a buffer at during SETUP time, which will then be used by the server to extract video frames when a PLAY request is received. The video frames are encoded and sent to Client through RTP packets, then stored locally in the client's buffer for replaying and skipping backwards.

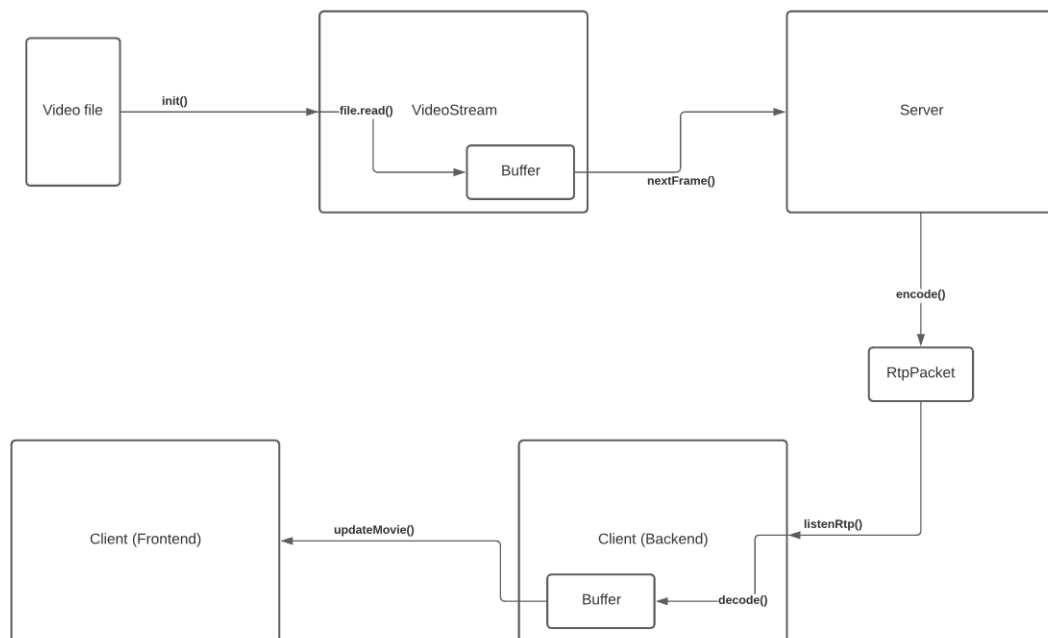


Figure 7: How video is transferred to the Client