Program 3: RTP and RTSP Streaming Video

Dr. William Kreahling

October 23, 2013

Video Streaming with RTP and RTSP

Overview

For this program you will finish my implementation of a streaming client and server. These programs will communicate using the *Real-Time Streaming Protocol* and they will send data using the *Real-time Transfer Protocol*.

Client

Contains the code and the User Interface (UI) which you will use to send RTSP commands and display the video. In the Client class you will need to handle the action that occur when buttons are pressed.

Server

Implements the server which will respond to the RTSP commands and streams back the video. The RTSP interaction is mostly implmented; the server calls methods from the RTPpacket class to encapsulate the video data in an RTP packet.

RTPpacket

The class is used to handle the RTP pakcets. You will need to write the constructors and methods for this class.

VideoStream

This class reads the video file from the Disk. You will have to complete this class. This class will implement VideoInterface

Running the code

: When you finished you will run the code as follows:

```
java Server <serverPortNum>
```

Server port is the port you want the server to listen to for RTSP connections. In the real world the standard port is 554, but we cannot use that port, you must use a port greater than 1024.

Start the client wil the following command.

```
java Client <hostname> <rtpPort> <serverPortNum> <videoFileName>
```

- *hostname* is the name of the machine running the RTSP server.

- *rtpPort* is the port we will transmit the RTP packets on.
- *serverPortNum* is the port the RTSP server is listening on.
- *videoFileName* is the name of the video file. **These files are large. DO NOT copy them into your personal directories. Instead use the full pathname to the video file or create a soft link to the file. The files (movie.Mjpeg, farL.Mjpeg, and seventies.Mjpeg are located at /home/wck/public_html/movies**

When you open the Server and the client you will see windows like this:

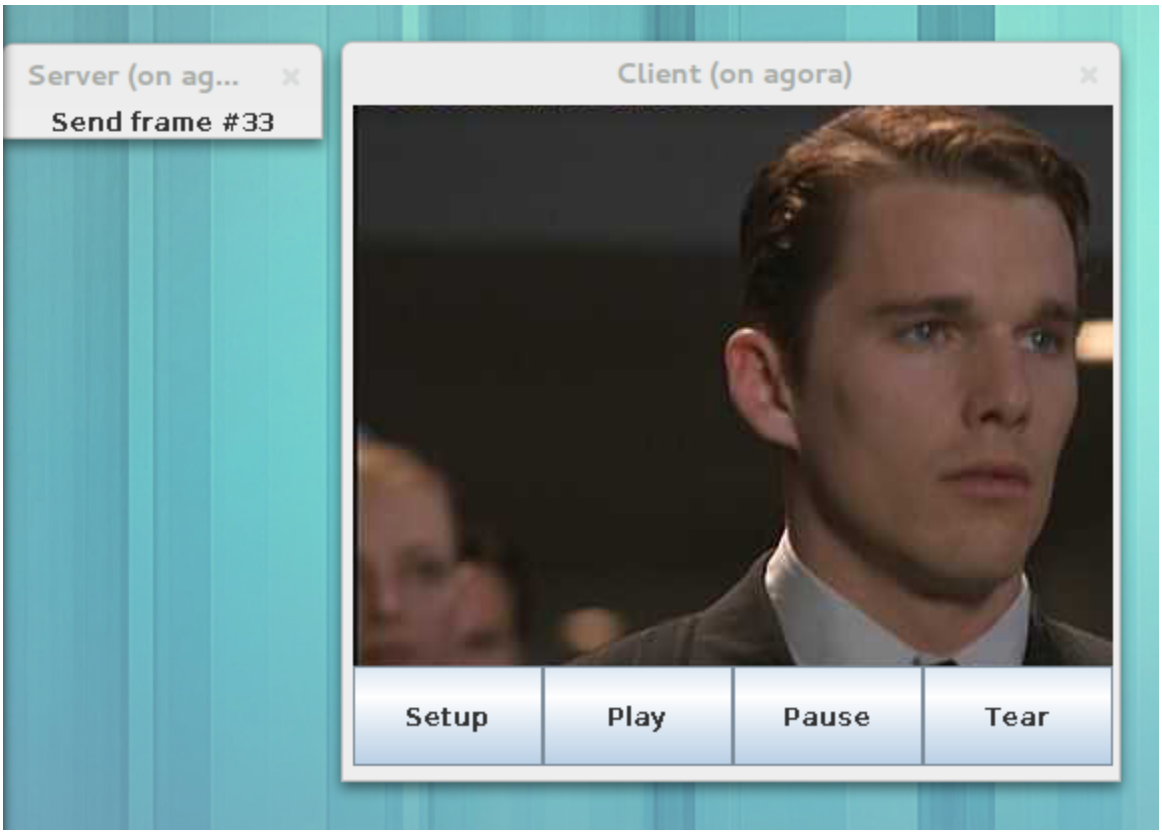


Figure 1: Client and Server after 33 frames have been streamed.

You send RTSP commands to the server by pressing buttons. An RTSP interation might go something like this:

1. Client sends SETUP. Sets up the session and transport parameters.
2. Client sends PLAY. Starts the playback.
3. Pause (optional) sent if the client wishes to pause playback.
4. Client sends TEARDOWN. This terminates and closes the connection.

The server *always* responds to client messages. The codes are roughly the same as in HTTP. For example, 200 means OKAY. In this assignment you only need to implement the OKAY code. See RFC 2326 for more information.

Client

For the client you need to complete the methdos that are called when ever a user clicks a button. Each button already has a handler and a method that is called when the button is clicked.

SETUP

Create a socket for the RTP Socket (set a small timeout) no greater than 10ms. Send a correctly formatted SETUP message, then read and parse the message returned by the server.

PLAY

Send a correctly formatted play request, then read and parse the server's response.

PAUSE

Send a correctly formatted pause request, then read and parse the server's response.

TEARDOWN

Send a correctly formatted teardown request, then read and parse the server's response.

Example

One possible session between client and server:

```
C: SETUP movie.Mjpeg RTSP/1.0
C: CSeq: 1
C: Transport: RTP/UDP; client_port= 5000

S: RTSP/1.0 200 OK
S: CSeq: 1
S: Session: 2113005544

C: PLAY movie.Mjpeg RTSP/1.0
C: CSeq: 2
C: Session: 2113005544

S: RTSP/1.0 200 OK
S: CSeq: 2
S: Session: 2113005544

C: PAUSE movie.Mjpeg RTSP/1.0
C: CSeq: 3
C: Session: 2113005544

S: RTSP/1.0 200 OK
S: CSeq: 3
```

```
S: Session: 2113005544

C: PLAY movie.Mjpeg RTSP/1.0
C: CSeq: 4
C: Session: 2113005544

S: RTSP/1.0 200 OK
S: CSeq: 4
S: Session: 2113005544

C: TEARDOWN movie.Mjpeg RTSP/1.0
C: CSeq: 5
C: Session: 2113005544

S: RTSP/1.0 200 OK
S: CSeq: 5
S: Session: 2113005544
```

Server

On the server side you will need to write the sendRtsptesponse method in Server.java

You will also need to write a class VideoStream that implements VideoInterface. Your constructor should accept a string representing the name of the file to stream.

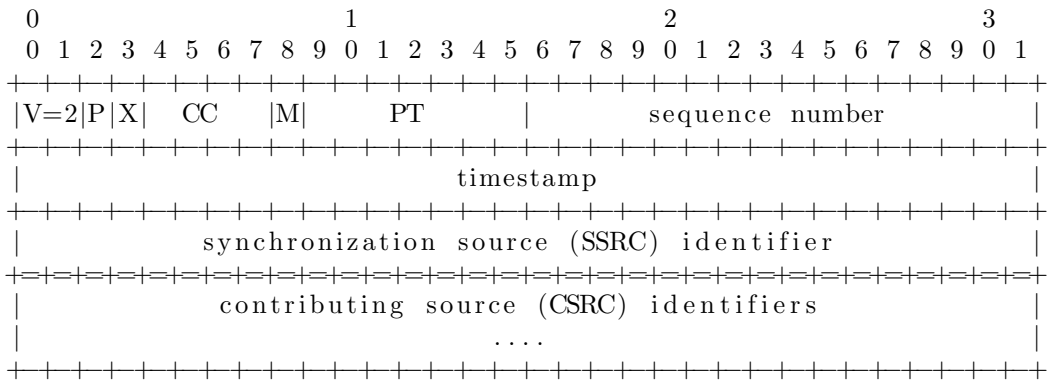
You will also need to complete a class to read the video from a file and a class to implment the packetization of the data into an RTP packet. You will need to create the packet, set the header fields and payload. We will assume one frame per packet. **This would not be the case in a real RTP server!**

When the server recieves a PLAY request it will start a timer that is triggered every 100ms. At these times the server will create an RTP packet and send it to the client. Do the following:

1. Set the RTP-version field (V) to 2.
2. Padding (P), Extension (X), number of contributing sources (CC), and Marker (M) fields are all not used for this assignment and thus set to zero.
3. The payload field (PT) should be set to the code for MJPEG which is 26.
4. Set the sequence number.
5. Set the timestamp. (This is not really used by our program, but it will contain a value)
6. Set the Synchronization Source Identifier (SSRC)

From RFC 3550:

The RTP header has the following format:



The first twelve octets are present in every RTP packet, while the list of CSRC identifiers is present only when inserted by a mixer.

Bit Twiddling

In the RTP packet header format the smaller bit numbers refer to high order bits. That is number 0 of a byte is 2^7 and bit number 7 is 1 or (2^0).

Because the RTP header fields are arrays of type `byte` you will need to set the header one byte at a time.

You may find the following operators useful:

- << shift bits left. num = foo << 4; Num gets the bits 4 high order bits from foo.
- >> shift bits right. num = foo >> 4; num gets the 4 low order bits from foo.
- & Perform a bitwise AND. num = 10 & 1; Num would get 1.

```
  010011
& 000001
--------
  000001
```

- | Perform a bitwise OR. num = 10 | 1; Num would get 11.

```
  010010
| 000001
--------
  010011
```

Hand-In Instructions

This assignment is due by 11:59 PM on Monday, November 4th. Submit only the Java source files. You must submit the assignment using the *handin* command on agora. Handin works as follows:

```
handin.<course#>.<section#>  <assignment#>  <files>
```

1 Notes on Collaboration

You are may work in groups of two for this assignment. **Please do not consult *anyone* other than me, or your teammate, on *any* aspect of this assignment**.