New York University
Computer Science Department
Courant Institute of Mathematical Sciences

Course Title: Data Communications & Networks **Course Number:** g22.2662-001
Instructor: Jean-Claude Franchitti **Session:** 10

Assignment 10: Final Project

I. Due

Sunday, May 15 2016 by midnight.

II. Objectives

1. Implement a streaming video server and client that communicate using the Real-Time Streaming Protocol (RSTP) and send data using the Real-Time Transfer Protocol (RTP)
2. Your client-server application focuses on implementing the RTSP protocol in the client and the RTP packetization in the server. Please note that the code that implements the RTSP protocol in the server, the RTP de-packetization in the client, and takes care of displaying the transmitted video is provided.
3. Your client-server application should implement the following additional features:
   a) Calculate statistics about the session including RTP packet loss rate, video data rate (in bits or bytes per second), and any other interesting statistics you can think of.
   b) As standard media player, such as RealPlayer or Windows Media Player have only 3 buttons for PLAY, PAUSE, and STOP (roughly corresponding to TEARDOWN), come up with and implement SETUP as part of the RTSP-interaction within your client (hint: figure out when the client sends the SETUP and whether it is appropriate to send TEARDOWN when the user clicks on the stop-button).
   c) 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 (note: currently the client and server only implement the minimum necessary RTSP interactions and PAUSE).
   d) Implement optimization as needed to handle congestion control and bandwidth allocation for packet video, network adaptive video coding and transport, and performance enhancements to improve user perceived quality.
   e) Optional (Extra Credit): Instead of the normal server given to you, use the class called FunkyServer (as well as the class FunkyServer$1.class), and

run your client-server application with “java FunkyServer server_port”. Comment on your observations, explain what happens and why, and fix the problem.

As always, the NYU and class policy about plagiarism must be followed in this project. If you use ANY code in your project that is not of your own creation, then you MUST attribute that code to the author, even if you modify it (ANY modification).

III. References

1. Slides and handouts posted on the course Web site
2. Textbook chapters as applicable
3. Sample code provided

IV. Software Required

1. Microsoft Word
2. Win Zip as necessary
3. Programming language and associated IDE of your choice
4. Network programming custom socket library of your choice

V. Assignment

1. Background:

This project on the implementation of a real-time streaming protocol (RTSP) server and client is based the RTSP and Real-time Transport Protocol (RTP) protocols cited in the course textbook ”Computer Networking: A Top-Down Approach” by Drs. Kurose and Ross. The project focuses on implementing and completing a streaming video server and client based on partial sample Java server and client code.

In this project, the server streams videos encoded into a proprietary MJPEG (Motion JPEG) 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.

From the sample code, the client uses the RTSP to control the actions (Setup, Play, Pause, or Teardown) of the server. And the server uses the real-time transport protocol (RTP) to packetize the video for transport over UDP to the client.

Understanding the difference between RTSP and RTP, proper manipulation of the RTP header fields, and the improving the display of the transmitted video are critical focus areas for this project.

2. Code Provided:

Four Java classes are provided to support this project:

Client
: This class implements the client and the user interface which you use to send RTSP commands and which is used to display the video. *You will need to implement the actions that are taken when the buttons are pressed*.

Server
: This class implements the server which responds to the RTSP requests and streams back the video. The RTSP interaction is already implemented and the server calls routines in the RTPpacket class to packetize the video data. *You will need to modify this class to implement optimization features as noted earlier.*

RTPpacket
: This class is used to handle the RTP packets. It has separate routines for handling the received packets at the client side which is given. *You will need to complete the first constructor of this class to implement RTP-packetization of the video data*. Note that the second constructor is used by the client to de-packetize the data.

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

```
java Server 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

```
java Client server_name server_port video_file
```

where `server_host` is the name of the machine where the server is running, `server_port` is the port the server is listening on, and `video_file` is the name of the file you want to request.

The client opens a connection to the server and pops up a window.

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

1. Client sends SETUP. This command is used to set up the session and transport parameters.
2. Client sends PLAY. This starts the playback.
3. Client may send PAUSE if it wants to pause during playback.
4. Client sends TEARDOWN. This terminates the session and closes the connection.

The server always replies to all the messages the client sends. The reply codes are roughly the same as in HTTP. The code 200 means that the request was successful. In this project, you do not need to implement any other reply codes.

For more information about RTSP, please see RFC 2326.

4. Implementing the Client Basic Features:

Your first task is to implement the RTSP on the client side. To do this, you need to complete the functions that are called when the user clicks on the buttons in the user interface. For each button in the interface there is a handler function in the code. You will need to implement the following actions in each handler function.

When the client starts, it also opens the RTSP socket to the server. Use this socket for sending all RTSP requests.

SETUP

- Create a socket for receiving RTP data and set the timeout on the socket to 5 milliseconds.
- Send SETUP request to server. You will need to insert the Transport header in which you specify the port for the RTP data socket you just created.
- Read reply from server and parse the Session header in the response to get the session ID.

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 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 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 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 is incremented by one for each request you send.

Example:

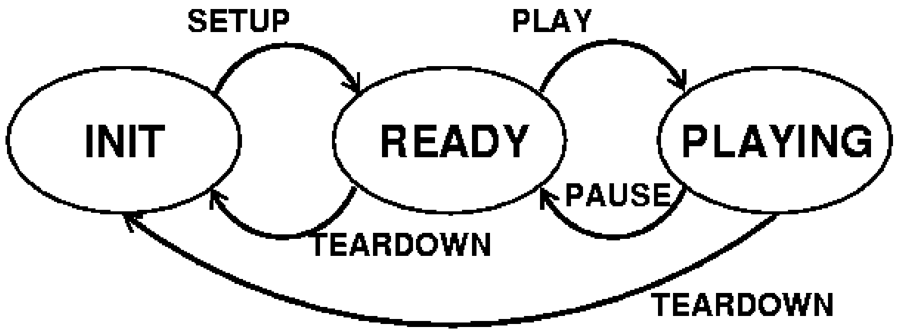
Here is a sample interaction between the client and server. The client's requests are marked with C: and server's replies with S:. In this project, both the client and the server are **very simple**. They do not have sophisticated parsing routines and they *expect the header fields to be in the order you see below*. That is, in a request, the first header is CSeq, and the second one is either Transport (for SETUP) or Session (for all other requests). In the reply, CSeq is again the first and Session is the second.

```
C: SETUP video_file.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 video_file.Mjpeg RTSP/1.0
C: CSeq: 2
C: Session: 123456

S: RTSP/1.0 200 OK
S: CSeq: 2
```

```
S: Session: 123456

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

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

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

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

C: TEARDOWN video_file.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. **Implementing the Server Basic Features:**

On the server you will first need to implement the packetization of the video data into RTP packets. For this 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, it starts a timer which is triggered every 100 ms. At these times the server will read one video frame from the file and send it to the client. The server creates an RTPpacket-object which is the RTP-encapsulation of the video frame.

The server calls the first constructor of the class RTPpacket to perform the encapsulation. 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)

1. Set the RTP-version field (V). You must set this to 2.
2. Set padding (P), extension (X), number of contributing sources (CC), and marker (M) fields. *These are all set to zero in this lab*.
3. Set payload type field (PT). In this lab we use MJPEG and the type for that is 26.
4. Set the sequence number. The server gives this the sequence number as the `Framenb` argument to the constructor.
5. Set the timestamp. The server gives this number as the `Time` argument to the constructor.
6. 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 in the array `header` of the RTPpacket-class. You will also need to copy the payload (given as argument `data`) to the variable `payload`. The length of the payload is given in the argument `data_length`.

The above diagram is in the network byte order (also known as big-endian). The Java Virtual Machine 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 an array of type `byte`, 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. In Java an int is 32 bits or 4 bytes.

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;
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. **Evaluation:**

This project is worth 100 points. You will be graded on both the completeness and accuracy of your program, as follows:

- Basic functionality [35 points]: Having basic functionality implemented for the client and the server.
- Statistics [15 points]: Having statistics about the session including RTP packet loss rate, video data rate (in bits or bytes per second), and any other interesting statistics you can think of.
- SETUP implementation [15 points]: Having SETUP designed and implemented as part of the RTSP-interaction within your client.
- DESCRIBE method implementation [15 points]: Having the method DESCRIBE implemented to send back a session description file which tells the client what kinds of streams are in the session and what encodings are used.
- Optimizations [15 points]: Having optimization implemented to handle congestion control and bandwidth allocation for packet video, network adaptive video coding and transport, and performance enhancements to improve user perceived quality.
- Style [5 points]:
  1. Coding Style: Well-structured, well documented, clean code, with well defined interfaces between components. Appropriate use of comments, clearly identified variables, constants, function names, etc.
  2. Assignment Layout:
     - o Assignment is neatly assembled on 8 1/2 by 11 paper
     - o Cover page with your name (last name first followed by a comma then first name), username and section number with a signed statement of independent effort is included
     - o Program and documentation submitted for Assignment #10 are satisfactory
     - o File name is correct.

Notice that many of the points are for basic functionality and simple congestion control. Make sure these work well before moving to more advanced functionality or worrying about corner-cases.

7. **What to Submit**

a. All of your source code files.
b. tests.txt/doc - file containing a brief description of your testing methods for the protocol interactions and other areas

c. vulnerabilities.txt/doc – file identifying at least one vulnerability in your current implementation
d. readme.txt: file containing a thorough description of your design and implementation. Please note that **all** code that you do not freshly write for this assignment must be clearly documented in this readme.txt file. The readme file must also clearly identify the name of the class that contains the main() method or program starting point.
e. Report document that describes your project briefly, explains your design, outlines some of the implementation details, and provides as assessment of what went well and not so well in your project. Problems should be clearly stated and solution approaches should be clearly documented (i.e., both current and new features that you implemented). You should also clearly document any simulation or modeling used in your approach and any evaluation metrics you used for comparative analysis of the current and new solutions. The format for your final project report should be similar to the standard conference paper formats and should include (at the very least) sections on Introduction, Related Work, Proposed Solution/Architecture/Algorithms, Simulation/Implementation, Results, and Conclusion.

Submissions that are incomplete or will not compile, link, or run according to your instructions will be returned to you and you will have to re-submit and accept an incomplete in the course and a loss of one half letter grade when you resubmit. No resubmitted assignment will be accepted after midnight on May 19, 2015.

8. **Email your project solution to the Instructor.**

VI. Deliverables

1. Electronic:

Your project submission archive file must be emailed to the grader. The file must be created and sent by the deadline. After the deadline, the project is late. The email clock is the official clock.

Your project submission archive file should contain the report file as well as your program source code packaged as a .jar file or as an equivalent binary file if you are not using Java. The various files submitted should be placed in the .jar file in a separate directory called "project documentation"

For Java users, to create the .jar file containing your Java source code (please do not include the class files), change your working directory to the directory where your Java files are located and execute the command:

```
jar cvf DCN-Spring2016-FinalProject-xxx.jar *.java
```

where **xxx** is **YOUR FULL STUDENT ID.**

Include the jar file in your project zip file and send the zip file as an email attachment to the grader.

You may send questions to the class mailing list ONLY.

2. Written:

   Printout of your project report.

   The cover page supplied on the next page must be the first page of your project report. Please fill in the blank area for each field.

   NOTE:

   The sequence of the report submission is:
   1. **Cover sheet**
   2. **Assignment Answer Sheet(s)**

VII. Sample Cover Sheet:

Name __________________________ Username: _______________ Date: _____________
(last name, first name)
Section: ____________

Assignment 10: Final Project

______ Basic functionality [Max 35 points]
______ Statistics [Max 15 points]
______ SETUP implementation [Max 15 points]
______ DESCRIBE method implementation [Max 15 points]
______ Optimizations [Max 15 points]
______ Style [Max 5 points]
______ Extra credit question [Max 15 points]

______Total [Max 100 points + Extra Credit points]

Total in points: _____________________

Total in extra credit points: _____________________

Professor's Comments:

Affirmation of my Independent Effort: _______________________________
(Sign here)