Name Qu, Ying Date: 05/18/2015

Section: 10

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: Ying Qu (Sign here)

• High level description of the application

This project requires me to 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).

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.

User 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.

• Design details

(1) I implement the streaming video server and client that communicate using RTSP and RTP. The Real Time Streaming Protocol (RTSP) is a network control protocol designed for use in entertainment and communications systems to control streaming media servers. The protocol is used for establishing and controlling media sessions between end points. The Real-time Transport Protocol (RTP) defines a standardized packet format for delivering audio and video over IP networks.

(2) I have four java classes (Client, Server, RTPpacket, VideoStream) in my application. My client-server application focuses on implementing the RTSP protocol in the client and the RTP packetization in the server. The server streams videos encoded into a proprietary MJPEG file format, which stores the video as concatenated JPEG-encoded images. 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.

• Implementation details

(1) Client

This class implements the client and the user interface which we use to send RTSP commands and which is used to display the video. I implement the actions that are taken when the buttons are pressed.

• SET UP

Create a socket for receiving RTP data and set the timeout on the socket to 5 milliseconds. Insert the Transport header in which I specify the port for the RTP data socket just created. Read reply from server and parse the Session header in the response to get the session ID.

I add the following code in the class setupButtonLister.

class setupButtonListener implements ActionListener{

public void actionPerformed(ActionEvent e){

//System.out.println("Setup Button pressed !");

if (state == INIT)

{

//Init non-blocking RTPsocket that will be used to receive data

try{

//construct a new DatagramSocket to receive RTP packets from the server, on port RTP\_RCV\_PORT

//RTPsocket = ...

RTPsocket = new DatagramSocket(RTP\_RCV\_PORT);

//set TimeOut value of the socket to 5msec.

//....

RTPsocket.setSoTimeout(5);

}

catch (SocketException se)

{

System.out.println("Socket exception: "+se);

System.exit(0);

}

//init RTSP sequence number

RTSPSeqNb = 1;

//Send SETUP message to the server

send\_RTSP\_request("SETUP");

//Wait for the response

if (parse\_server\_response() != 200)

System.out.println("Invalid Server Response");

else

{

//change RTSP state and print new state

//state = ....

//System.out.println("New RTSP state: ....");

state = READY;

System.out.println("New RTSP state:READY");

}

}//else if state != INIT then do nothing

}

}

• PLAY

This class should send PLAY request. I insert the Session header and use the session ID returned in the SETUP response. The playButtonListener class is as follows:

class playButtonListener implements ActionListener {

public void actionPerformed(ActionEvent e){

//System.out.println("Play Button pressed !");

if (state == READY)

{

//increase RTSP sequence number

//.....

RTSPSeqNb++;

//Send PLAY message to the server

send\_RTSP\_request("PLAY");

//Wait for the response

if (parse\_server\_response() != 200)

System.out.println("Invalid Server Response");

else

{

//change RTSP state and print out new state

//.....

// System.out.println("New RTSP state: ...")

state = PLAYING;

System.out.println("New RTSP state: PLAYING");

//start the timer

timer.start();

}

}//else if state != READY then do nothing

}

}

• PAUSE

This class sends PAUSE request. I insert the Session header and use the session ID returned in the SETUP response.

class pauseButtonListener implements ActionListener {

public void actionPerformed(ActionEvent e){

//System.out.println("Pause Button pressed !");

if (state == PLAYING)

{

//increase RTSP sequence number

//........

RTSPSeqNb++;

//Send PAUSE message to the server

send\_RTSP\_request("PAUSE");

//Wait for the response

if (parse\_server\_response() != 200)

System.out.println("Invalid Server Response");

else

{

//change RTSP state and print out new state

//........

//System.out.println("New RTSP state: ...");

state = READY;

System.out.println("New RTSP state:READY");

//stop the timer

timer.stop();

}

}

//else if state != PLAYING then do nothing

}

}

• TEARDOWN

This class should send TEARDOWN request. I insert the session header and use the session ID returned in the SETUP response.

class tearButtonListener implements ActionListener {

public void actionPerformed(ActionEvent e){

//System.out.println("Teardown Button pressed !");

//increase RTSP sequence number

// ..........

RTSPSeqNb++;

//Send TEARDOWN message to the server

send\_RTSP\_request("TEARDOWN");

//Wait for the response

if (parse\_server\_response() != 200)

System.out.println("Invalid Server Response");

else

{

//change RTSP state and print out new state

//........

//System.out.println("New RTSP state: ...");

state = INIT;

System.out.println("New RTSP state:INIT");

//stop the timer

timer.stop();

//exit

System.exit(0);

}

}

}

I have to insert the CSeq header in every request I send. The value of the CSeq header is a number which is incremented by one for each request I send.

I also insert code to the method send\_RTSP\_request.

private void send\_RTSP\_request(String request\_type)

{

try{

//Use the RTSPBufferedWriter to write to the RTSP socket

//write the request line:

//RTSPBufferedWriter.write(...);

RTSPBufferedWriter.write(request\_type+""+VideoFileName+"RTSP/1.0"+CRLF);

//write the CSeq line:

//......

RTSPBufferedWriter.write("CSeq:"+RTSPSeqNb+CRLF);

//check if request\_type is equal to "SETUP" and in this case write the Transport: line advertising to the server the port used to receive the RTP packets RTP\_RCV\_PORT

//if ....

//otherwise, write the Session line from the RTSPid field

//else ....

if ( 0 == (new String(request\_type)).compareTo("SETUP") )

RTSPBufferedWriter.write("Transport: RTP/UDP; client\_port= " + RTP\_RCV\_PORT + CRLF);

else

RTSPBufferedWriter.write("Session: " + RTSPid + "\n");

RTSPBufferedWriter.flush();

}

catch(Exception ex)

{

System.out.println("Exception caught: "+ex);

System.exit(0);

}

}

(2) RTPpacket

This class is used to handle the RTP packets. It has separate routines for handling the received packets at the client side which given. The second constructor is used by the client to de-packetize the data. I complete the first constructor of this class to implement RTP-packetization of the video data.

This first constructor is as follows.

//--------------------------

//Constructor of an RTPpacket object from header fields and payload bitstream

//--------------------------

public RTPpacket(int PType, int Framenb, int Time, byte[] data, int data\_length){

//fill by default header fields:

Version = 2;

Padding = 0;

Extension = 0;

CC = 0;

Marker = 0;

Ssrc = 0;

//fill changing header fields:

SequenceNumber = Framenb;

TimeStamp = Time;

PayloadType = PType;

//build the header bistream:

//--------------------------

header = new byte[HEADER\_SIZE];

//.............

//TO COMPLETE

//.............

//fill the header array of byte with RTP header fields

//header[0] = ...

// .....

header[0] = new Integer((Version << 6) | (Padding << 5) | (Extension << 4) | CC).byteValue();

header[1] = new Integer((Marker << 7) | PayloadType).byteValue();

header[2] = new Integer(SequenceNumber >> 8).byteValue();

header[3] = new Integer(SequenceNumber).byteValue();

for ( int i = 0; i < 4; i++ )

{

header[7 - i] = new Integer(TimeStamp >> (8 \* i)).byteValue();

header[11 - i] = new Integer(Ssrc >> (8 \* i)).byteValue();

}

//fill the payload bitstream:

//--------------------------

payload\_size = data\_length;

payload = new byte[data\_length];

//fill payload array of byte from data (given in parameter of the constructor)

//......

for ( int i = 0; i < data\_length; i++ )

{

payload[i] = data[i];

}

// ! Do not forget to uncomment method printheader() below !

}

(3) Server

This class implements the server which responds to the RTSP requests and streams back the video. I need to implement the packetization of the video data into RTP packets. I create the packet, set the fields in the packet header, and copy the payload into the packet. When the server receives the PLAY-request from the client, it starts a timer which is triggered every 100ms. 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.

(4) VideoStream

This class is used to read video data from the file on disk. I don’t make changes to this class.

4. Test results

I used Java IDE Eclipse to run write my program.

Run the program as follows:

First, start the server as follows:

It will pops up a window

Then, start the client

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

User can send RTSP commands to the server by pressing the buttons as shown. The client window plays the video as follows.

• What went well and not so well

In my program, the client can send RTSP commands to the server successfully and display the video. The server can respond to the RTSP requests and streams back the video.

The optimization part doesn’t work well. My program cannot successfully handle congestion control and bandwidth allocation for packet video.

• Timeline achieved

By May 13th, I finished reading the textbook about RTSP and RTP materials.

By May 15th, I finished the code for Client and Server class.

By May16th, I finished the code for RTPpacket class.

By May 18th, I successfully ran the program and finished the report.