

UNIVERSITY OF BRIDGEPORT

Department Of Computer Science and Engineering



Final Project

Advanced Problems Solution (in Java)

Submitted by:

Utpal Rana
ID # 559536

Advisor : Prof. Gohnsin Liu

INDEX

		Page no.
Assignment 1	Multi threaded Web Server - Part A	
1.1	Introduction	4
1.2	Description	4
1.3	Code	5
1.4	Result	7
Assignment 2	Multi threaded Web Server - Part B	
2.1	Introduction	8
2.2	Description	8
2.3	Code	8
2.4	Result	11
Assignment 3	Implementing a Reliable Transport Protocol	
1.1	Introduction	13
1.2	Description	13
1.3	Code	18
1.4	Result	18
Assignment 4	Implementing an Algorithm (a distributed asynchronous distance vector routing for the network)	
1.1	Introduction	19
1.2	Description	19
1.3	Code	22
1.4	Result	22

Assignment 5

Streaming Video with RTSP and RTP

1.1	Introduction	23
1.2	Description	23
1.3	Code	29
1.4	Result	29

Introduction :

The Multi threaded web server is developed in java which is capable of processing multiple simultaneous service requests in parallel. This server displays the contents of HTTP request message that it receives. The server implements HTTP 1.0 (as defined in RFC 1945) ,where separate HTTP requests are send for each component of the Web Page.

Description:

The Multi threaded web server processes each incoming request which take place inside a separate thread of execution. This allows the server to service multiple clients in parallel.

When we create a new thread of execution, we need to pass to the Thread 's constructor an instance of some class that implements the runnable interface.

The server program begins by creating a new ServerSocket object to listen on some specific port.

ServerSocket is a java.net class that provides a system-independent implementation of the server side of a client/server socket connection. It waits for requests to come in over the network. It performs some operation based on that request, and then possibly returns a result to the requester. The constructor for ServerSocket throws an exception if it can't listen on the specified port (for example, the port is already being used).

When connection request is received, HttpRequest object is created, passing to its constructor a reference to the socket object that represents established connection with client.

In order to have the HttpRequest Object handle the incoming Http service request in a separate thread , a new Thread object is created with reference of HttpRequest object is passed to it, then thread's start() method is called.

After the new thread has been created and started,execution in the main thread returns to the top of message processing loop. The main thread will then block, waiting for another TCP connection request,while the new thread continue running. When another TCP connection request is received, the main thread goes through the same process of thread creation regardless of whether the previous thread has finished exectuion or is still running.This completes the code in Main() method.

The HttpRequest has CRLF variable which is created with a carriage return(CR) and a line feed(LF).

This CRLF variable is used to terminate earch line of the server's response message. The HttpRequest has socket variable which will be used to store a reference to the connection socket , which is passed to the constructor of this class.

HttpRequest implements Runnable interface. it has run() method which is triggered when thread's start() method is called. The processing code in the run() method must catch all exceptions within run() method through try/catch block.

In the HTTPRequest class , BufferedReader and InputStreamReader filters are wrapped around input stream.

Once the server receives the client request, the request can be read in by input stream. The readLine() method of BufferedReader extracts characters from input stream until it reaches end of line character. The HTTP request line is extracted first from the message header. After obtaining request line, the headerline is extracted and displayed on the console. The Streams and socket are closed at the end of the processing.

Code :

```
import java.net.ServerSocket;
import java.net.Socket;

public final class WebServer {

    public static void main(String[] args) throws Exception{
        int port =2222;

        ServerSocket ServSocket = new ServerSocket (port);
        //          open listen socket
        while (true)
        {
//          LISTEN for a tcp conn
            Socket req= ServSocket.accept();
            HttpRequest hreq= new HttpRequest(req);
            Thread th= new Thread(hreq);
            th.start();

        }
    }
}
```

WebServer.java

```
import java.io.*;
import java.net.Socket;

public class HttpRequest implements Runnable{

    final static String CRLF="\r\n";
    Socket socket;
    public HttpRequest (Socket socket) throws Exception
    {
        this.socket=socket;
    }

    public void run() {

        try{
            processRequest();
        }catch(Exception e)
        {

```

```

        System.out.println(e);
    }
}
private void processRequest() throws Exception
{
//        DataOutputStream os= new DataOutputStream(socket.getInputStream());
//        setup input stream filters

        BufferedReader br= new BufferedReader(new
InputStreamReader(socket.getInputStream()));
        String requestLine=null;

//        System.out.println();
//        System.out.println(requestLine);

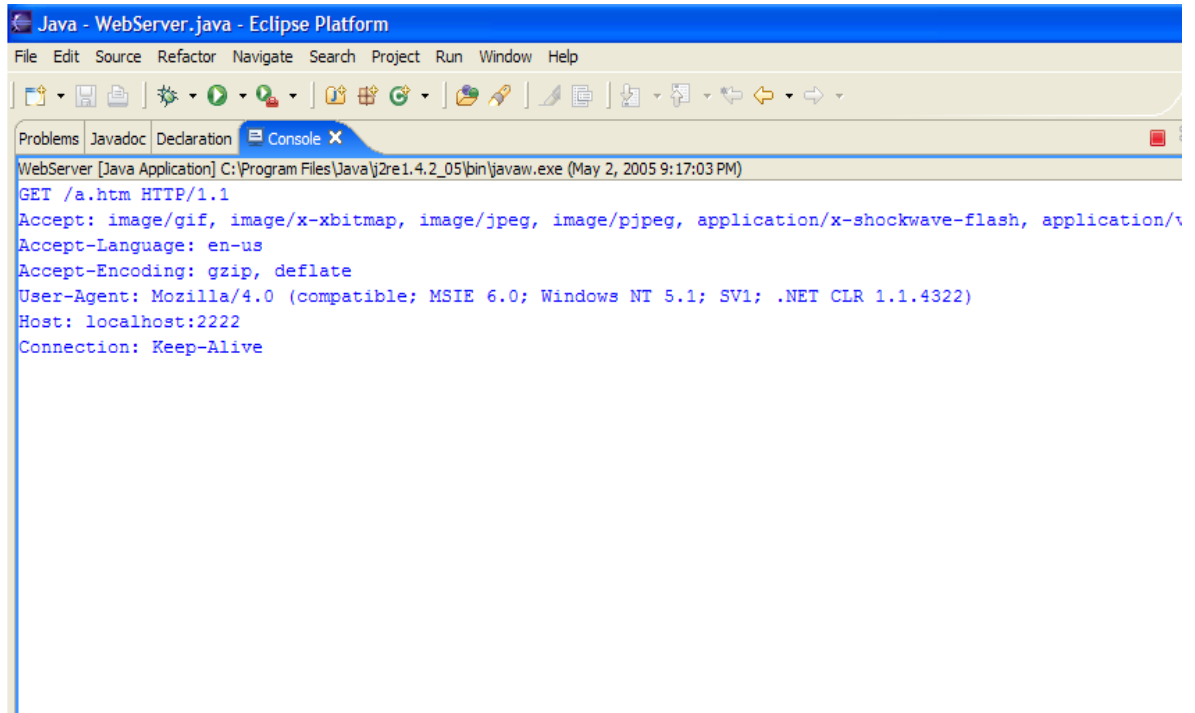
        String headerline=null ;
        while ((headerline=br.readLine()).length()!=0)
        {
            System.out.println(headerline);
        }
//os.close();
br.close();
socket.close();

    }
}

```

HttpRequest.java

Results:



The screenshot shows the Eclipse IDE interface with the console window open. The title bar reads "Java - WebServer.java - Eclipse Platform". The menu bar includes "File", "Edit", "Source", "Refactor", "Navigate", "Search", "Project", "Run", "Window", and "Help". The toolbar contains various icons for file operations and development tools. The console window shows the following output:

```
WebServer [Java Application] C:\Program Files\Java\j2re1.4.2_05\bin\javaw.exe (May 2, 2005 9:17:03 PM)
GET /a.htm HTTP/1.1
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, application/x-shockwave-flash, application/v
Accept-Language: en-us
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1; .NET CLR 1.1.4322)
Host: localhost:2222
Connection: Keep-Alive
```

Assignment 2 Multi threaded Web Server - Part B

Introduction :

In this part B of developing Multi threaded web server, instead of simply terminating thread after displaying the browser's HTTP request message, the request is analysed and appropriate message is sent to browser.

Description:

When client makes a request for some file, the request is passed to `HttpRequest` object. In the `processRequest()` method of `HttpRequest` class, requested file name is extracted from the request line using `StringTokenizer` class. The file will be opened using `FileInputStream` class. If `FileInputStream` object doesn't find the file it throws `FileNotFoundException`. which will be caught using `try/catch` block. and appropriate error response message can be sent to the user instead of trying to send non existent file.

If `FileInputStream` finds the requested file, it returns the file handle to the instance variable.

There are three parts to the response message : status line, response headers and entity body. The status line and response headers are terminated by the character sequence `CRLF`.

In case of request of non existent file, '404 not found' is returned in status line of response message and error message is included in the form of HTML document in the entity body.

when the file exists, the file's MIME type is determined in `contentType()` method, and appropriate MIME type is sent to the client. After sending status line, header line and entity body in output stream, the stream and socket are closed.

Code :

```
import java.net.ServerSocket;
import java.net.Socket;

public class WebServer {

    public static void main(String[] args) throws Exception{
        int port =2222;

        ServerSocket ServSocket = new ServerSocket (port);
        //          open listen socket
        while (true)
        {
            //          LISTEN for a tcp conn
            Socket req= ServSocket.accept();
            HttpRequest hreq= new HttpRequest(req);
            Thread th= new Thread(hreq);
            th.start();

        }

    }
}
```


WebServer.java

```
import java.io.*;
import java.net.Socket;
import java.util.StringTokenizer;

public class HttpRequest implements Runnable{

    final static String CRLF="\r\n";
    Socket socket;
    public HttpRequest (Socket socket) throws Exception
    {
        this.socket=socket;
    }

    public void run() {

        try{
            processRequest();
        }catch(Exception e)
        {
            System.out.println(e);
        }
    }

    private void processRequest() throws Exception
    {

//          DataOutputStream os= new DataOutputStream(socket.getInputStream());
//          setup input stream filters

        DataOutputStream os=new
DataOutputStream(socket.getOutputStream());

        BufferedReader br= new BufferedReader(new
InputStreamReader(socket.getInputStream()));
        String requestLine=null;

//          System.out.println();
//          System.out.println(requestLine);

        String headerline=null ;
        boolean fileExists= true;
        FileInputStream fis= null;
        String file=null;
        if ((headerline=br.readLine()).length()!= 0)
        {

            System.out.println(headerline);

            StringTokenizer tokens= new StringTokenizer(headerline);
            tokens.nextToken();
            file= tokens.nextToken();
```

```

        file="."+file;
        System.out.println("file is "+file);
        try{

            fis= new FileInputStream (file);
        }catch(FileNotFoundException e)
        {
            fileExists=false;
        }
    }
    String status= null;
    String content= null;

    String body= null;
    if(fileExists)
    {
        status="HTTP/1.0 200 OK\r\n";
        content="Content-Type: "+contentType(file)+CRLF;
    }
    else
    {
        status="HTTP/1.1 404 File Not Found\r\n";

        content= "Content-Type: text/html\r\n";
        body= "File Is Not Found.";

    }

    os.writeBytes(status);
    os.writeBytes(content);
    os.writeBytes(CRLF);

    if(fileExists)
    {
        sendBytes(fis,os);
        fis.close();
    }else{
        os.writeBytes(body);

    }
    os.close();
    br.close();
    socket.close();
System.out.println("REQUEST Ends");
//
    os.close();
    //br.close();
    //cket.close();

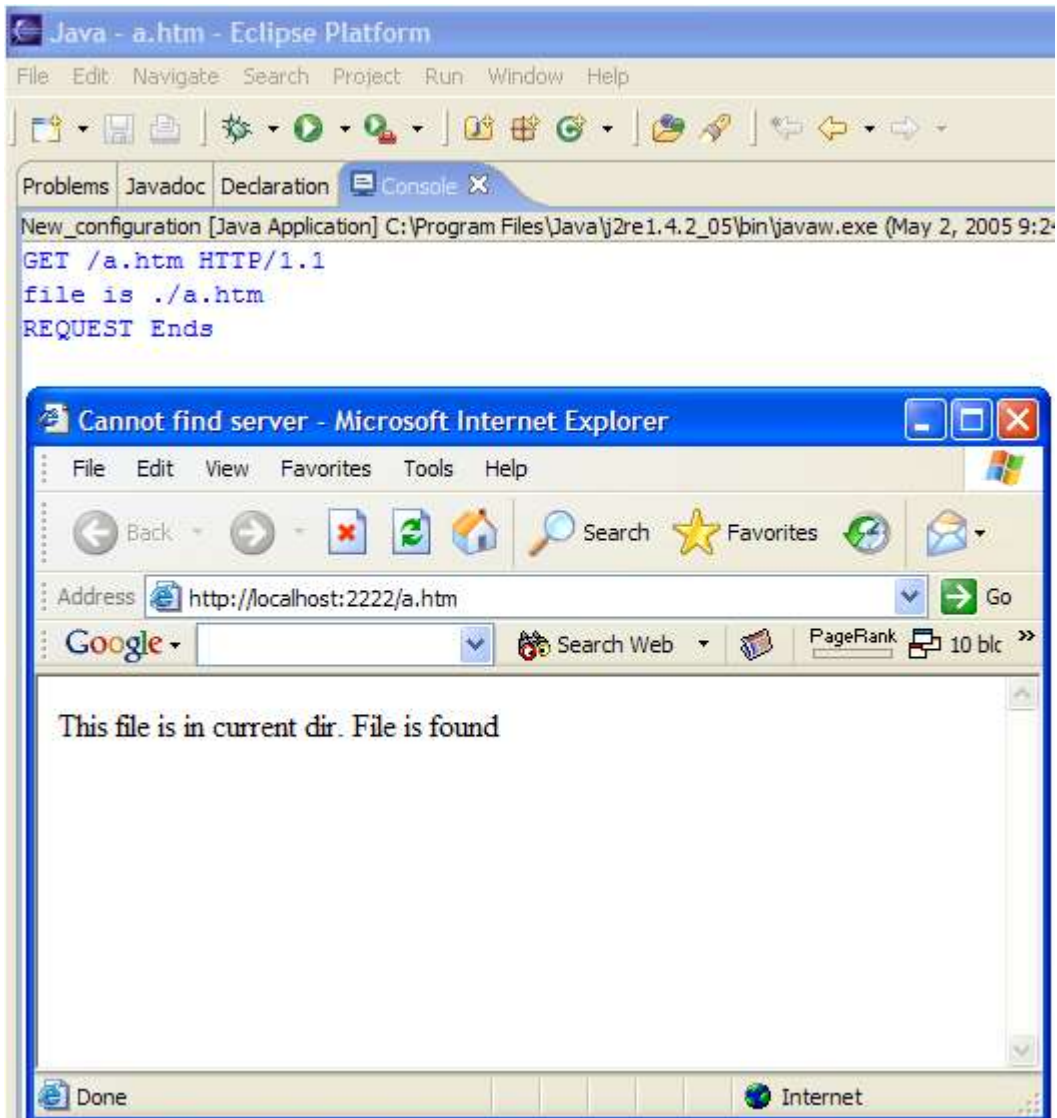
}

```

```
void sendBytes(FileInputStream fis,OutputStream os) throws Exception
{
byte[] buffer= new byte[1024];
int bytes= 0;
while ((bytes=fis.read(buffer))!=-1)
{
os.write(buffer,0,bytes);
}
}
private static String contentType(String fileName)
{
if (fileName.endsWith(".htm")|| fileName.endsWith(".html"))
{
return "text/html";
}
if(fileName.endsWith(".jpg")){
return "image/jpeg";
}
if(fileName.endsWith(".gif")){
return "image/gif";
}
return "application/octet-stream";
}
}
```

HttpRequest.java

Results:



Assignment 3 Implementing a Reliable Transport Protocol

Overview

In this laboratory programming assignment, you will be writing the sending and receiving transport-level code for implementing a simple reliable data transfer protocol. There are two versions of this lab, the Alternating-Bit-Protocol version and the Go-Back-N version. This lab should be **fun** since your implementation will differ very little from what would be required in a real-world situation.

Since you probably don't have standalone machines (with an OS that you can modify), your code will have to execute in a simulated hardware/software environment. However, the programming interface provided to your routines, i.e., the code that would call your entities from above and from below is very close to what is done in an actual UNIX environment. (Indeed, the software interfaces described in this programming assignment are much more realistic than the infinite loop senders and receivers that many texts describe). Stopping/starting of timers are also simulated, and timer interrupts will cause your timer handling routine to be activated.

The routines you will write

The procedures you will write are for the sending entity (A) and the receiving entity (B). Only unidirectional transfer of data (from A to B) is required. Of course, the B side will have to send packets to A to acknowledge (positively or negatively) receipt of data. Your routines are to be implemented in the form of the procedures described below. These procedures will be called by (and will call) procedures that I have written which emulate a network environment. The overall structure of the environment is shown in [Figure Lab.3-1](#) (structure of the emulated environment):

The unit of data passed between the upper layers and your protocols is a *message*, which is declared as:

```
struct msg {
    char data[20];
};
```

This declaration, and all other data structure and emulator routines, as well as stub routines (i.e., those you are to complete) are in the file, **prog2.c**, described later. Your sending entity will thus receive data in 20-byte chunks from layer5; your receiving entity should deliver 20-byte chunks of correctly received data to layer5 at the receiving side.

The unit of data passed between your routines and the network layer is the *packet*, which is declared as:

```
struct pkt {
    int seqnum;
    int acknum;
    int checksum;
    char payload[20];
};
```

Your routines will fill in the payload field from the message data passed down from layer5. The other packet fields will be used by your protocols to insure reliable delivery, as we've seen in class.

The routines you will write are detailed below. As noted above, such procedures in real-life would be part of the operating system, and would be called by other procedures in the operating system.

- **A_output(message)**, where message is a structure of type msg, containing data to be sent to the B-side. This routine will be called whenever the upper layer at the sending side (A) has a message to send. It is the job of your protocol to insure that the data in such a message is delivered in-order, and correctly, to the receiving side upper layer.
- **A_input(packet)**, where packet is a structure of type pkt. This routine will be called whenever a packet sent from the B-side (i.e., as a result of a tolayer3() being done by a B-side procedure) arrives at the A-side. packet is the (possibly corrupted) packet sent from the B-side.
- **A_timerinterrupt()** This routine will be called when A's timer expires (thus generating a timer interrupt). You'll probably want to use this routine to control the retransmission of packets. See starttimer() and stoptimer() below for how the timer is started and stopped.
- **A_init()** This routine will be called once, before any of your other A-side routines are called. It can be used to do any required initialization.
- **B_input(packet)**, where packet is a structure of type pkt. This routine will be called whenever a packet sent from the A-side (i.e., as a result of a tolayer3() being done by a A-side procedure) arrives at the B-side. packet is the (possibly corrupted) packet sent from the A-side.
- **B_init()** This routine will be called once, before any of your other B-side routines are called. It can be used to do any required initialization.

Software Interfaces

The procedures described above are the ones that you will write. I have written the following routines which can be called by your routines:

- **starttimer(calling_entity,increment)**, where calling_entity is either 0 (for starting the A-side timer) or 1 (for starting the B side timer), and increment is a *float* value indicating the amount of time that will pass before the timer interrupts. A's timer should only be started (or stopped) by A-side routines, and similarly for the B-side timer. To give you an idea of the appropriate increment value to use: a packet sent into the network takes an average of 5 time units to arrive at the other side when there are no other messages in the medium.
- **stoptimer(calling_entity)**, where calling_entity is either 0 (for stopping the A-side timer) or 1 (for stopping the B side timer).
- **tolayer3(calling_entity,packet)**, where calling_entity is either 0 (for the A-side send) or 1 (for the B side send), and packet is a structure of type pkt. Calling this routine will cause the packet to be sent into the network, destined for the other entity.
- **tolayer5(calling_entity,message)**, where calling_entity is either 0 (for A-side delivery to layer 5) or 1 (for B-side delivery to layer 5), and message is a structure of type msg. With unidirectional data transfer, you would only be calling this with calling_entity equal to 1 (delivery to the B-side). Calling this routine will cause data to be passed up to layer 5.

The simulated network environment

A call to procedure `tolayer3()` sends packets into the medium (i.e., into the network layer). Your procedures `A_input()` and `B_input()` are called when a packet is to be delivered from the medium to your protocol layer.

The medium is capable of corrupting and losing packets. It will not reorder packets. When you compile your procedures and my procedures together and run the resulting program, you will be asked to specify values regarding the simulated network environment:

- **Number of messages to simulate.** My emulator (and your routines) will stop as soon as this number of messages have been passed down from layer 5, regardless of whether or not all of the messages have been correctly delivered. Thus, you need **not** worry about undelivered or unACK'ed messages still in your sender when the emulator stops. Note that if you set this value to 1, your program will terminate immediately, before the message is delivered to the other side. Thus, this value should always be greater than 1.
- **Loss.** You are asked to specify a packet loss probability. A value of 0.1 would mean that one in ten packets (on average) are lost.
- **Corruption.** You are asked to specify a packet loss probability. A value of 0.2 would mean that one in five packets (on average) are corrupted. Note that the contents of payload, sequence, ack, or checksum fields can be corrupted. Your checksum should thus include the data, sequence, and ack fields.
- **Tracing.** Setting a tracing value of 1 or 2 will print out useful information about what is going on inside the emulation (e.g., what's happening to packets and timers). A tracing value of 0 will turn this off. A tracing value greater than 2 will display all sorts of odd messages that are for my own emulator-debugging purposes. A tracing value of 2 may be helpful to you in debugging your code. You should keep in mind that *real* implementors do not have underlying networks that provide such nice information about what is going to happen to their packets!
- **Average time between messages from sender's layer5.** You can set this value to any non-zero, positive value. Note that the smaller the value you choose, the faster packets will be arriving to your sender.

The Alternating-Bit-Protocol Version of this lab.

You are to write the procedures, `A_output()`, `A_input()`, `A_timerinterrupt()`, `A_init()`, `B_input()`, and `B_init()` which together will implement a stop-and-wait (i.e., the alternating bit protocol, which we referred to as `rdt3.0` in the text) unidirectional transfer of data from the A-side to the B-side.

Your protocol should use both ACK and NACK messages.

You should choose a very large value for the average time between messages from sender's layer5, so that your sender is never called while it still has an outstanding, unacknowledged message it is trying to send to the receiver. I'd suggest you choose a value of 1000. You should also perform a check in your sender to make sure that when `A_output()` is called, there is no message currently in transit. If there is, you can simply ignore (drop) the data being passed to the `A_output()` routine.

You should put your procedures in a file called `prog2.c`. You will need the initial version of this file, containing the emulation routines we have written for you, and the stubs for your procedures. You can obtain this program from <http://gaia.cs.umass.edu/kurose/transport/prog2.c>.

This lab can be completed on any machine supporting C. It makes no use of UNIX features. (You can simply copy the prog2.c file to whatever machine and OS you choose).

We recommend that you should hand in a code listing, a design document, and sample output. For your sample output, your procedures might print out a message whenever an event occurs at your sender or receiver (a message/packet arrival, or a timer interrupt) as well as any action taken in response. You might want to hand in output for a run up to the point (approximately) when 10 messages have been ACK'ed correctly at the receiver, a loss probability of 0.1, and a corruption probability of 0.3, and a trace level of 2. You might want to annotate your printout with a colored pen showing how your protocol correctly recovered from packet loss and corruption.

Make sure you read the "helpful hints" for this lab following the description of the Go_Back-N version of this lab.

The Go-Back-N version of this lab.

You are to write the procedures, A_output(),A_input(),A_timerinterrupt(),A_init(),B_input(), and B_init() which together will implement a Go-Back-N unidirectional transfer of data from the A-side to the B-side, with a window size of 8. Your protocol should use both ACK and NACK messages. Consult the alternating-bit-protocol version of this lab above for information about how to obtain the network emulator.

We would **STRONGLY** recommend that you first implement the easier lab (Alternating Bit) and then extend your code to implement the harder lab (Go-Back-N). Believe me - it will **not** be time wasted! However, some new considerations for your Go-Back-N code (which do not apply to the Alternating Bit protocol) are:

- **A_output(message)**, where message is a structure of type msg, containing data to be sent to the B-side.

Your A_output() routine will now sometimes be called when there are outstanding, unacknowledged messages in the medium - implying that you will have to buffer multiple messages in your sender. Also, you'll also need buffering in your sender because of the nature of Go-Back-N: sometimes your sender will be called but it won't be able to send the new message because the new message falls outside of the window.

Rather than have you worry about buffering an arbitrary number of messages, it will be OK for you to have some finite, maximum number of buffers available at your sender (say for 50 messages) and have your sender simply abort (give up and exit) should all 50 buffers be in use at one point (Note: using the values given below, this should never happen!) In the ``real-world," of course, one would have to come up with a more elegant solution to the finite buffer problem!

- **A_timerinterrupt()** This routine will be called when A's timer expires (thus generating a timer interrupt). Remember that you've only got one timer, and may have many outstanding, unacknowledged packets in the medium, so you'll have to think a bit about how to use this single timer.

Consult the Alternating-bit-protocol version of this lab above for a general description of what you might want to hand in. You might want to hand in output for a run that was long enough so

that at least 20 messages were successfully transferred from sender to receiver (i.e., the sender receives ACK for these messages) transfers, a loss probability of 0.2, and a corruption probability of 0.2, and a trace level of 2, and a mean time between arrivals of 10. You might want to annotate parts of your printout with a colored pen showing how your protocol correctly recovered from packet loss and corruption.

For **extra credit**, you can implement bidirectional transfer of messages. In this case, entities A and B operate as both a sender and receiver. You may also piggyback acknowledgments on data packets (or you can choose not to do so). To get my emulator to deliver messages from layer 5 to your B_output() routine, you will need to change the declared value of BIDIRECTIONAL from 0 to 1.

Helpful Hints and the like

- **Checksumming.** You can use whatever approach for checksumming you want. Remember that the sequence number and ack field can also be corrupted. We would suggest a TCP-like checksum, which consists of the sum of the (integer) sequence and ack field values, added to a character-by-character sum of the payload field of the packet (i.e., treat each character as if it were an 8 bit integer and just add them together).
- Note that any shared "state" among your routines needs to be in the form of global variables. Note also that any information that your procedures need to save from one invocation to the next must also be a global (or static) variable. For example, your routines will need to keep a copy of a packet for possible retransmission. It would probably be a good idea for such a data structure to be a global variable in your code. Note, however, that if one of your global variables is used by your sender side, that variable should **NOT** be accessed by the receiving side entity, since in real life, communicating entities connected only by a communication channel can not share global variables.
- There is a float global variable called *time* that you can access from within your code to help you out with your diagnostics msgs.
- **START SIMPLE.** Set the probabilities of loss and corruption to zero and test out your routines. Better yet, design and implement your procedures for the case of no loss and no corruption, and get them working first. Then handle the case of one of these probabilities being non-zero, and then finally both being non-zero.
- **Debugging.** We'd recommend that you set the tracing level to 2 and put LOTS of printf's in your code while your debugging your procedures.
- **Random Numbers.** The emulator generates packet loss and errors using a random number generator. Our past experience is that random number generators can vary widely from one machine to another. You may need to modify the random number generation code in the emulator we have supplied you. Our emulation routines have a test to see if the random number generator on your machine will work with our code. If you get an error message:

It is likely that random number generation on your machine is different from what this emulator expects. Please take a look at the routine jimsrand() in the emulator code. Sorry.

then you'll know you'll need to look at how random numbers are generated in the routine jimsrand(); see the comments in that routine.

Code :

```

package simulator;

import java.util.Date;

public class Event {
    Date eventTime;
    int eventType;
    int eventity;
    Packet packet;
    public String toString ()
    {
        if(packet!=null)
        {
            return "eventTime: "+eventTime+"| eventType: "+      eventType+ "| eventity: "+eventity+"|
Packet : "+packet.toString();
        }
        else{
            return "eventTime: "+eventTime+"| eventType: "+ eventType+ "| eventity: "+eventity;
        }

    }

    public int getEventity() {
        return eventity;
    }
    public void setEventity(int eventity) {
        this.eventity = eventity;
    }
    public Date getEventTime() {
        return eventTime;
    }
    public void setEventTime(Date eventTime) {
        this.eventTime = eventTime;
    }
    public int getEventType() {
        return eventType;
    }
    public void setEventType(int eventType) {
        this.eventType = eventType;
    }
    public Packet getPacket() {
        return packet;
    }
    public void setPacket(Packet packet) {
        this.packet = packet;
    }
}

```

Event.java

```

/*
 * Created on Oct 23, 2005
 *
 * TODO To change the template for this generated file go to
 * Window - Preferences - Java - Code Style - Code Templates
 */
package simulator;

public class Message {
    public char[] data= new char[20];
    public String toString()
    {

```

```
        return (new String (data));
    }
}
```

Message.java

```
/*
 * Created on Oct 23, 2005
 *
 * TODO To change the template for this generated file go to
 * Window - Preferences - Java - Code Style - Code Templates
 */
package simulator;

import java.io.Serializable;

public class Packet implements Cloneable, Serializable {
    private int seqNum;
    private int ackNum;
    private int checksum;
    public char[] payload = new char[20];
    public int getAckNum() {
        return ackNum;
    }
    public void setAckNum(int ackNum) {
        this.ackNum = ackNum;
    }
    public int getChecksum() {
        return checksum;
    }
    public void setChecksum(int checksum) {
        this.checksum = checksum;
    }
    public int getSeqNum() {
        return seqNum;
    }
    public void setSeqNum(int seqNum) {
        this.seqNum = seqNum;
    }
    public int calcChecksum()
    {
        return 100;
    }

    public Object clone() {

        Cloneable packet = new Packet();

        return packet;
    }
    public String toString()
    {
        return " seqNum: "+seqNum+"| ackNum: "+ackNum+"| checksum: "+checksum + "| data : " +
new String(payload);
    }
}
```

Packet.java

```
/*
 * Created on Oct 23, 2005
```

```

*
* TODO To change the template for this generated file go to
* Window - Preferences - Java - Code Style - Code Templates
*/
package simulator;

import java.util.ArrayList;
import java.util.Date;
import java.util.Iterator;
import java.util.Random;

public class Simulator {
    public final static boolean BIDIRECTIONAL = false;
    public final static int TIMER_INTERRUPT = 0;
    public final static int FROM_LAYER5 = 1;
    public final static int FROM_LAYER3 = 2;
    public final static int OFF = 0;
    public final static int ON = 1;
    public final static int A = 0;
    public final static int B = 1;

    private ArrayList eventList = null;
    private Event event = null;

    private int traceLevel = 1;
    private int nSim = 0;
    private int nSimMax = 0;
    private Date time;
    private double lossProbability;
    private double corruptProbability;
    private double lambda;
    private int nToLayer3;
    private int nLost;
    private int nCorrupt;

    public Simulator(){
        eventList = new ArrayList();
    }

    public Event getNextEvent(){
        Event tmpEvent = null;
        Iterator iterator = eventList.iterator();
        if (iterator.hasNext()){
            tmpEvent = (Event) iterator.next();
            eventList.remove(tmpEvent);
            return tmpEvent;
        }
        return null;
    }

    public void removeEvent(){
    }

    public void printEvent(Event event){
        System.out.println("The Event is :"+ event.toString());
    }

    public void printMessage(Message msg){

```

```

        System.out.println("The message is :"+msg.toString());

    }

    public void printPacket(Packet pkt){

        System.out.println("The packet is :"+pkt.toString());

    }

    private void aInit(){ }
    private void bInit(){ }

    private void aOutput(Message msg){
        //String msgStr = "Simulator is working";
        // for(int i=0;i<msgStr.length();i++) msg.data[i]=msgStr.charAt(i);
        Packet packet = new Packet();
        packet.setSeqNum(1);
        packet.setChecksum(packet.calcChecksum());
        packet.setAckNum(1);
        packet.payload = msg.data;
        toLayer3(A, packet);
    }

    private void bInput(Packet packet){
        System.out.println("Student.bInput() : =>");
        printPacket(packet);
    }
    private void bOutput(Message msg){ }

    private void aInput(Packet packet){ }

private void addEvent(Event event){
    if (getTraceLevel() > 2){
        System.out.println("Simulator.addEvent() =>");
        printEvent(event);
    }

    Iterator itr = eventList.iterator();
    Event tmpEvent = null;
    int idx =0;
    while (itr.hasNext()){
        tmpEvent = (Event) itr.next();
        if (event.getEventTime().getTime() < tmpEvent.getEventTime().getTime()){
            idx=eventList.indexOf(tmpEvent);
            break;
        }
    }
    eventList.add(idx,event);
    //printEventList();
}
    private void aTimerInterrupt(){ }
    private void bTimerInterrupt(){ }

    private void fatalError(){ }

    private double getJimRand(){
        Random r = new Random();
        return r.nextDouble();
    }

    private void genNextArrival(){
        long x;

```

```

Event Oevent=null;
if (this.getTraceLevel()>2)
    System.out.println("    GENERATE NEXT ARRIVAL: creating new arrival\n");
x = (new Double(this.getLambda()*this.getJimRand()*2)).longValue();
Oevent= new Event();

//ISSUE
Oevent.setEventTime(new Date(this.getTime().getTime()+x));

Oevent.setEventType(this.FROM_LAYER5);
Oevent.setEventivity(this.A);

addEvent(Oevent);
}
private void printEventList(){
    System.out.println("Simulator.printEventList() => ");
    Iterator itr = eventList.iterator();
    Event tmpEvent = null;
    while (itr.hasNext()){
        tmpEvent = (Event) itr.next();
        printEvent(tmpEvent);
    }
}

private void startTimer(int aORb, int increment){
    if (getTraceLevel() >2){
        System.out.println("Simulator.startTimer() => starting timer at " + time.getTime());
    }
    Iterator itr = eventList.iterator();
    Event tmpEvent = null;
    while (itr.hasNext()){
        tmpEvent = (Event) itr.next();
        if (tmpEvent.getEventivity() == aORb && tmpEvent.getEventType() ==
TIMER_INTERRUPT){
            System.out.println("Simulator.startTimer() => WARN: Attempt to start the
timer that has already started. ");
            if (getTraceLevel() >2){
                printEvent(tmpEvent);
            }
            return;
        }
    }
    Event event = new Event();
    event.setEventivity(aORb);
    event.setEventTime(new Date(time.getTime() + (long) increment));
    event.setEventType(TIMER_INTERRUPT);
    addEvent(event);
}
private void stopTimer(int aORb){
    if (getTraceLevel() >2){
        System.out.println("Simulator.stopTimer() => stopping timer at " + time.getTime());
    }
    Iterator itr = eventList.iterator();
    Event tmpEvent = null;
    while (itr.hasNext()){
        tmpEvent = (Event) itr.next();
        if (tmpEvent.getEventivity() == aORb && tmpEvent.getEventType() ==
TIMER_INTERRUPT){
            eventList.remove(tmpEvent);
            if (getTraceLevel() >2){
                System.out.println("Simulator.stopTimer() => removed interrupt
event ");

```

```

        printEvent(tmpEvent);
    }
    return;
}
}
System.out.println("Unable to stop the timer. It wasn't running!!!!");
}

private void toLayer3(int aORb, Packet packet){
    if ( getTraceLevel() > 2){
        System.out.println("Simulator.toLayer3() => Entered :");
        printPacket(packet);
    }

    nToLayer3++;

    if (loose()) return;

    corrupt(packet);

    // Create Event
    Event event = new Event();
    event.setEventity((aORb+1) %2);
    event.eventType = FROM_LAYER3;
    event.setPacket(packet);
    Date lastTime = new Date( getEventLastTime(FROM_LAYER3, event.getEventity()).getTime()
+ 1 + (long)(9* getJimRand() ) );
    event.setEventTime(lastTime);

    if ( getTraceLevel() > 2){
        System.out.println("Simulator.toLayer3() => Exited :");
        printPacket(packet);
    }

    addEvent(event);
}

private Date getEventLastTime(int layer, int eventity){
    Event tmpEvent = null;
    Date lastTime = time;
    Iterator itr = eventList.iterator();
    while (itr.hasNext()){
        tmpEvent = (Event) itr.next();
        if (tmpEvent.getEventType() == FROM_LAYER3 && tmpEvent.getEventity()
== event.getEventity())
            lastTime = tmpEvent.getEventTime();
    }
    return lastTime;
}

private void toLayer5(int aORb, char[] data){ }

private boolean loose(){
    if (getJimRand() < getLossProbability()) {
        nLost++;
        if (getTraceLevel() > 0){
            System.out.println("Simulator.toLayer3() => packet being lost");
        }
        return true;
    }
}

```

```

        return false;
    }

    private void corrupt(Packet packet){
        double jRand = getJimRand();
        if (getJimRand() < getCorruptProbability() {
            nCorrupt++;
            if ( jRand < .75){
                packet.payload[0]='Z';
            } else if ( jRand < .875){
                packet.setSeqNum(9999);
            } else {
                packet.setChecksum(99999);
            }
            if (getTraceLevel() >2){
                System.out.println("Simulator.toLayer3() =>: packet is being
corrupted");
                printPacket(packet);
            }
        }
    }

    private void init(){
        this.setNSimMax(2);
        this.setLossProbability(0.5);
        this.setCorruptProbability(0.0);
        this.setLambda(0.2);
        this.setTraceLevel(3);

        nToLayer3 = 0;
        nLost = 0;
        nCorrupt = 0;
        time=new Date();
        // time=0.0;          /* initialize time to 0.0 */
        genNextArrival();
    }

    public void run(){
        Date time = null;
        int i,j;
        Message msg2give = new Message();
        Packet pkt2give = new Packet();

        init();
        aInit();
        bInit();
        while (((event = getNextEvent()) != null) ){           // runs in a infinite loop
            System.out.println("nSim is :"+nSim);
            if (getTraceLevel() > 0) {
                System.out.println("MAINLOOP: ");
            }
            printEvent(event);
        }

        time = event.getEventTime();

        switch(event.getEventType()){
            case FROM_LAYER5:

                j = nSim % 26;

                for (i=0;i<20;i++){
                    msg2give.data[i] = (char) (j+97);
                }
        }
    }

```



```

        if (getTraceLevel() > 0) {
            System.out.println("  MAINLOOP : data give to student :");
            printMessage(msg2give);
        }

        nSim++;
        if (nSim < nSimMax)
        {
            genNextArrival();
        }

        if (event.getEntity() == A)
            aOutput(msg2give);
        else
            bOutput(msg2give);

        break;
    case FROM_LAYER3:
        pkt2give = event.getPacket();
        if (event.getEntity() == A)
            aInput(pkt2give);
        else
            bInput(pkt2give);

        break;

    case TIMER_INTERRUPT:
        if (event.getEntity() == A)
            aTimerInterrupt();
        else
            bTimerInterrupt();
        break;

    default :
        fatalError();
        break;
    }
}

terminate:
    System.out.println("Simulator terminated !");
}

public double getCorruptProbability() {
    return corruptProbability;
}

public void setCorruptProbability(double corruptProbability) {
    this.corruptProbability = corruptProbability;
}

public double getLambda() {
    return lambda;
}

public void setLambda(double lambda) {
    this.lambda = lambda;
}

public double getLossProbability() {
    return lossProbability;
}

public void setLossProbability(double lossProbability) {
    this.lossProbability = lossProbability;
}
}

```

```

    public int getNSimMax() {
        return nSimMax;
    }
    public void setNSimMax(int nsimmax) {
        this.nSimMax = nsimmax;
    }
    public Date getTime() {
        return time;
    }
    public void setTime(Date time) {
        this.time = time;
    }

    public int getTraceLevel() {
        return traceLevel;
    }
    public void setTraceLevel(int traceLevel) {
        this.traceLevel = traceLevel;
    }
}

```

Simulator.java

```

* Created on Oct 23, 2005
*
* TODO To change the template for this generated file go to
* Window - Preferences - Java - Code Style - Code Templates
*/
package test;

import simulator.Simulator;

public class TestSimulator {

    public static void main(String[] args) {
        Simulator sim= new Simulator();
        sim.run();
    }
}

```

TestSimulator.java

Results:

Following is the Output of the Simulation (nSimMax=2, LossProbability=0.5, CorruptProbability=0.0, Lambda=0.2,TraceLevel=3)

```

-----
GENERATE NEXT ARRIVAL: creating new arrival
Simulator.addEvent() =>
The Event is :eventTime: Mon Oct 24 23:09:12 EDT 2005| eventType: 1| eventity: 0
nSim is :0
MAINLOOP:
The Event is :eventTime: Mon Oct 24 23:09:12 EDT 2005| eventType: 1| eventity: 0
  MAINLOOP : data give to student :
The message is :aaaaaaaaaaaaaaaaaaaaa
  GENERATE NEXT ARRIVAL: creating new arrival
Simulator.addEvent() =>
The Event is :eventTime: Mon Oct 24 23:09:12 EDT 2005| eventType: 1| eventity: 0
Simulator.toLayer3() => Entered :
The packet is : seqNum: 1| ackNum: 1| checkSum: 100| data : aaaaaaaaaaaaaaaaaaaaaa

```

```
Simulator.toLayer3() => Exited :
The packet is : seqNum: 1| ackNum: 1| checkSum: 100| data : aaaaaaaaaaaaaaaaaa
Simulator.addEvent() =>
The Event is :eventTime: Mon Oct 24 23:09:12 EDT 2005| eventType: 2| eventity: 1| Packet : seqNum: 1| ackNum:
1| checkSum: 100| data : aaaaaaaaaaaaaaaaaa
nSim is :1
MAINLOOP:
The Event is :eventTime: Mon Oct 24 23:09:12 EDT 2005| eventType: 2| eventity: 1| Packet : seqNum: 1| ackNum:
1| checkSum: 100| data : aaaaaaaaaaaaaaaaaa
Student.bInput() : =>
The packet is : seqNum: 1| ackNum: 1| checkSum: 100| data : aaaaaaaaaaaaaaaaaa
nSim is :1
MAINLOOP:
The Event is :eventTime: Mon Oct 24 23:09:12 EDT 2005| eventType: 1| eventity: 0
  MAINLOOP : data give to student :
The message is :bbbbbbbbbbbbbbbbbbbb
Simulator.toLayer3() => Entered :
The packet is : seqNum: 1| ackNum: 1| checkSum: 100| data : bbbbbbbbbbbbbbbbbbb
Simulator.toLayer3() => Exited :
The packet is : seqNum: 1| ackNum: 1| checkSum: 100| data : bbbbbbbbbbbbbbbbbbb
Simulator.addEvent() =>
The Event is :eventTime: Mon Oct 24 23:09:12 EDT 2005| eventType: 2| eventity: 1| Packet : seqNum: 1| ackNum:
1| checkSum: 100| data : bbbbbbbbbbbbbbbbbbb
nSim is :2
MAINLOOP:
The Event is :eventTime: Mon Oct 24 23:09:12 EDT 2005| eventType: 2| eventity: 1| Packet : seqNum: 1| ackNum:
1| checkSum: 100| data : bbbbbbbbbbbbbbbbbbb
Student.bInput() : =>
The packet is : seqNum: 1| ackNum: 1| checkSum: 100| data : bbbbbbbbbbbbbbbbbbb
Simulator terminated !
=====
```

Assignment 4 Implementing an Algorithm (a distributed asynchronous distance vector routing for the network)

Overview

In this lab, you will be writing a "distributed" set of procedures that implement a distributed asynchronous distance vector routing for the network.

The Basic Assignment

The routines you will write For the basic part of the assignment, you are to write the following routines which will "execute" asynchronously within the emulated environment that we have written for this assignment.

For node 0, you will write the routines:

- `rtinit0()` This routine will be called once at the beginning of the emulation. `rtinit0()` has no arguments. It should initialize the distance table in node 0 to reflect the direct costs of 1, 3, and 7 to nodes 1, 2, and 3, respectively. In Figure 1, all links are bi-directional and the costs in both directions are identical. After initializing the distance table, and any other data structures needed by your node 0 routines, it should then send its directly-connected neighbors (in this case, 1, 2 and 3) the cost of its minimum cost paths to all other network nodes. This minimum cost information is sent to neighboring nodes in a *routing packet* by calling the routine `tolayer2()`, as described below. The format of the routing packet is also described below.
- `rtupdate0(struct rtpkt *rcvdpkt)`. This routine will be called when node 0 receives a routing packet that was sent to it by one of its directly connected neighbors. The parameter `*rcvdpkt` is a pointer to the packet that was received.

rtupdate0() is the "heart" of the distance vector algorithm. The values it receives in a routing packet from some other node *i* contain *i*'s current shortest path costs to all other network nodes. rtupdate0() uses these received values to update its own distance table (as specified by the distance vector algorithm). If its own minimum cost to another node changes as a result of the update, node 0 informs its directly connected neighbors of this change in minimum cost by sending them a routing packet. Recall that in the distance vector algorithm, only directly connected nodes will exchange routing packets. Thus nodes 1 and 2 will communicate with each other, but nodes 1 and 3 will not communicate with each other.

As we saw in class, the distance table inside each node is the principal data structure used by the distance vector algorithm. You will find it convenient to declare the distance table as a 4-by-4 array of int's, where entry [i,j] in the distance table in node 0 is node 0's currently computed cost to node *i* via direct neighbor *j*. If 0 is not directly connected to *j*, you can ignore this entry. We will use the convention that the integer value 999 is "infinity."

Similar routines are defined for nodes 1, 2 and 3. Thus, you will write 8 procedures in all: rtinit0(), rtinit1(), rtinit2(), rtinit3(), rtupdate0(), rtupdate1(), rtupdate2(), rtupdate3()

Software Interfaces

The procedures described above are the ones that you will write. We have written the following routines that can be called by your routines:

tolayer2(struct rtpkt pkt2send)

where rtpkt is the following structure, which is already declared for you. The procedure tolayer2() is defined in the file [prog3.c](#)

```
extern struct rtpkt {
    int sourceid; /* id of node sending this pkt, 0, 1, 2, or 3 */
    int destid; /* id of router to which pkt being sent
                (must be an immediate neighbor) */
    int mincost[4]; /* min cost to node 0 ... 3 */
};
```

Note that tolayer2() is passed a structure, not a pointer to a structure.

printdt0()

will pretty print the distance table for node 0. It is passed a pointer to a structure of type distance_table. printdt0() and the structure declaration for the node 0 distance table are declared in the file node0.c. Similar pretty-print routines are defined for you in the files node1.c, node2.c node3.c.

The simulated network environment

Your procedures rtinit0(), rtinit1(), rtinit2(), rtinit3() and rtupdate0(), rtupdate1(), rtupdate2(), rtupdate3() send routing packets (whose format is described above) into the medium. The medium will deliver packets in-order, and without loss to the specified destination. Only directly-connected nodes can communicate. The delay between sender and receiver is variable (and unknown).

When you compile your procedures and my procedures together and run the resulting program, you will be asked to specify only one value regarding the simulated network environment:

- **Tracing.** Setting a tracing value of 1 or 2 will print out useful information about what is going on inside the emulation (e.g., what's happening to packets and timers). A tracing value of 0 will turn this off. A tracing value greater than 2 will display all sorts of odd messages that are for my own emulator-debugging purposes.

A tracing value of 2 may be helpful to you in debugging your code. You should keep in mind that *real* implementors do not have underlying networks that provide such nice information about what is going to happen to their packets!

The Basic Assignment

You are to write the procedures `rtinit0()`, `rtinit1()`, `rtinit2()`, `rtinit3()` and `rtupdate0()`, `rtupdate1()`, `rtupdate2()`, `rtupdate3()` which together will implement a distributed, asynchronous computation of the distance tables for the topology and costs shown in Figure 1.

You should put your procedures for nodes 0 through 3 in files called `node0.c`, `node3.c`. You are **NOT** allowed to declare any global variables that are visible outside of a given C file (e.g., any global variables you define in `node0.c` may only be accessed inside `node0.c`). This is to force you to abide by the coding conventions that you would have to adopt if you were really running the procedures in four distinct nodes. To compile your routines: `cc prog3.c node0.c node1.c node2.c node3.c`. Prototype versions of these files are here: [node0.c](#), [node1.c](#), [node2.c](#), [node3.c](#). You can pick up a copy of the file `prog3.c` at <http://gaia.cs.umass.edu/kurose/network/prog3.c>.

This assignment can be completed on any machine supporting C. It makes no use of UNIX features.

As always, most instructors would expect you to hand in a code listing, a design document, and sample output.

For your sample output, your procedures should print out a message whenever your `rtinit0()`, `rtinit1()`, `rtinit2()`, `rtinit3()` or `rtupdate0()`, `rtupdate1()`, `rtupdate2()`, `rtupdate3()` procedures are called, giving the time (available via my global variable `clocktime`). For `rtupdate0()`, `rtupdate1()`, `rtupdate2()`, `rtupdate3()` you should print the identity of the sender of the routing packet that is being passed to your routine, whether or not the distance table is updated, the contents of the distance table (you can use my pretty-print routines), and a description of any messages sent to neighboring nodes as a result of any distance table updates.

The sample output should be an output listing with a `TRACE` value of 2. Highlight the final distance table produced in each node. Your program will run until there are no more routing packets in-transit in the network, at which point our emulator will terminate.

The Advanced Assignment

You are to write two procedures, `rtlinkhandler0(int linkid, int newcost)` and `rtlinkhandler1(int linkid, int newcost)`, which will be called if (and when) the cost of the link between 0 and 1 changes. These routines should be defined in the files `node0.c` and `node1.c`, respectively. The routines will be passed the name (id) of the neighboring node on the other side of the link whose cost has changed, and the new cost of the link. Note that when a link cost changes, these routines will have to update the distance table and may (or may not) have to send updated routing packets to neighboring nodes.

In order to complete the advanced part of the assignment, you will need to change the value of the constant LINKCHANGES (line 3 in prog3.c) to 1. FYI, the cost of the link will change from 1 to 20 at time 10000 and then change back to 1 at time 20000. Your routines will be invoked at these times.

We would again **STRONGLY** recommend that you first implement the undergraduate assignment and then extend your code to implement the graduate assignment. It will **not** be time wasted. (Believe me, I learned this the hard way!)

Code :

```
import java.util.Date;

public class Event {
    Date eventTime;
    int eventType;
    int eventity;
    RoutingPacket packet;
    public String toString ()
    {
        if(packet!=null)
        {
            return "eventTime: "+eventTime+"| eventType: "+      eventType+ "| eventity: "+eventity+"|
Packet : "+packet.toString();
        }
        else{
            return "eventTime: "+eventTime+"| eventType: "+ eventType+ "| eventity: "+eventity;
        }
    }

    public int getEventity() {
        return eventity;
    }
    public void setEventity(int eventity) {
        this.eventity = eventity;
    }
    public Date getEventTime() {
        return eventTime;
    }
    public void setEventTime(Date eventTime) {
        this.eventTime = eventTime;
    }
    public int getEventType() {
        return eventType;
    }
    public void setEventType(int eventType) {
        this.eventType = eventType;
    }
    public RoutingPacket getPacket() {
        return packet;
    }
    public void setPacket(RoutingPacket packet) {
        this.packet = packet;
    }
}
```

Event.java

```
import java.util.ArrayList;

public class Node {
    int costs[][]=new int [4][4];

    int mincosts[]=new int [4];
    ArrayList packets=new ArrayList();
    Simulator sim=null;
    int dest[]=null;
    /**
     * @param args
     */

    public RoutingPacket creatertpkt(int srcid, int destid,int mincosts[])
    {
        RoutingPacket oRPacket =new RoutingPacket();
        oRPacket.setDestid(destid);
        oRPacket.setSourceid(srcid);
        oRPacket.setMincost(mincosts);
    }
    return oRPacket;
}
void rinit(int srcid,int dest[], int initcosts[][],int initMinCosts[],Simulator s)
{
    System.out.println("Entering rinit\n");

    costs=initcosts;
    mincosts=initMinCosts;
    sim=s;
    this.dest=dest;
    for(int i=0;i<dest.length;i++)
        sim.toLayer2(creatertpkt(srcid,dest[i],mincosts));

        //packets.add(creatertpkt(srcid,dest[i],mincosts));
    printTable();
}
public void rtupdate(RoutingPacket oReceivedPacket)
{
    System.out.println("Entering rtupdate");
    int destid, i, j;
    int srcid = oReceivedPacket.sourceid;

    for(destid=0; destid < 4; destid++)
    {
        int mincosts12[ ] = { 999, 999, 999, 999};

        //if there is new min distance exists
        if(oReceivedPacket.mincost[destid]!=999)
        {
            //update distance table

            this.costs[destid][srcid] = this.costs[srcid][srcid] + oReceivedPacket.mincost[destid];
            //find new min for that destination cost
            if( this.costs[destid][srcid] < this.mincosts[destid])
            {
                //update min cost table
                this.mincosts[destid] = this.costs[destid][srcid];
                mincosts12[destid] = this.costs[destid][srcid];

                //send new minum value to all its neighbors

                for(int k=0;k<this.dest.length;k++)
```



```

sim.toLayer2(creatertpkt(srcid,this.dest[k],mincosts[2]));
    }
    }
}
printTable();
}

void printTable()
{
    System.out.println("          via  \n");
    System.out.println(" D0 | 1  2  3 \n");
    System.out.println(" ----|-----\n");
    System.out.println(" 1| "+costs[1][1]+" "+costs[1][2]+" "+costs[1][3]);
    System.out.println(" 2| "+costs[2][1]+" "+costs[2][2]+" "+costs[2][3]);
    System.out.println(" 3| "+costs[3][1]+" "+costs[3][2]+" "+costs[3][3]);
}
}
}

```

Node.java

```

public class RoutingPacket {

    int sourceid; /* id of sending router sending this pkt */
    int destid; /* id of router to which pkt being sent
                (must be an immediate neighbor) */
    int mincost[]=new int [4]; /* min cost to node 0 ... 3 */
    public int getDestid() {
        return destid;
    }
    public void setDestid(int destid) {
        this.destid = destid;
    }
    public int[] getMincost() {
        return mincost;
    }
    public void setMincost(int[] mincost) {
        this.mincost = mincost;
    }
    public int getSourceid() {
        return sourceid;
    }
    public void setSourceid(int sourceid) {
        this.sourceid = sourceid;
    }
}
}

```

RoutingPacket.java

```

import java.util.ArrayList;
import java.util.Date;
import java.util.Iterator;
import java.util.Random;

public class Simulator {
    int LINKCHANGES =0;
    int TRACE = 1; /* for my debugging */
    int YES = 1;
    int NO = 0;
    int FROM_LAYER2 = 2;
    int LINK_CHANGE = 10;
}

```

```

double clocktime = 0.000;
private ArrayList eventList = null;
private Event event = null;
Node node0= new Node();
Node node1= new Node();
Node node2= new Node();
Node node3= new Node();
Simulator()
{eventList=new ArrayList();
}

/**
 * @param args
 */
public static void main(String[] args) {
    // TODO Auto-generated method stub
    Simulator sim= new Simulator();
    sim.run();
}
void init()          /* initialize the simulator */
{

    node0.rtinit(0,initDest0(),initCosts0(),initMinCosts0(),this);

    node1.rtinit(1,initDest1(),initCosts1(),initMinCosts1(),this);
    node2.rtinit(2,initDest2(),initCosts2(),initMinCosts2(),this);
    node3.rtinit(3,initDest3(),initCosts3(),initMinCosts3(),this);

}
int[] initDest0()
{
    int dest[]={1,3};
    return dest;
}
int[] initDest1()
{
    int dest[]={2,0};
    return dest;
}
int[] initDest2()
{
    int dest[]={3,1};
    return dest;
}
int[] initDest3()
{
    int dest[]={0,2};
    return dest;
}
int[][] initCosts0()
{
    int costs[][]= {{0,2,6,14},{1,1,999,999},
                    {3,999,3,999},{7,999,999,7}};
    return costs;
}
int[] initMinCosts0()
{
    int mincosts[]={0,1,3,7};
    return mincosts;
}

```

```

int[] initMinCosts1()
{
    int mincosts[]={1,0,1,999};
    return mincosts;
}
int[] initMinCosts2()
{
    int mincosts[]={3,1,0,2};
    return mincosts;
}
int[] initMinCosts3()
{
    int mincosts[]={7,999,2,0};
    return mincosts;
}
int[][] initCosts1()
{
    int costs[][]={{0,2,6,14},{1,1,999,999},
                  {3,999,3,999},{7,999,999,7} };
    return costs;
}
int[][] initCosts2()
{
    int costs[][]={{3,999,3,999},{999,1,1,999},
                  {6,999,0,4},{999,999,2,2} };
    return costs;
}
int[][] initCosts3()
{
    int costs[][]={{7,999,999,7},{999,999,999,999},
                  {999,999,2,2},{14,999,4,0} };
    return costs;
}

public int getLINKCHANGES() {
    return LINKCHANGES;
}
public void setLINKCHANGES(int linkchanges) {
    LINKCHANGES = linkchanges;
}
public int getTRACE() {
    return TRACE;
}
public void setTRACE(int trace) {
    TRACE = trace;
}
private double getJimRand(){
    Random r = new Random();
    return r.nextDouble();
}
public void toLayer2(RoutingPacket oRoutingPacket)
{
    System.out.println("toLayer2");
    Event newEvent=new Event();
    newEvent.setEventType(FROM_LAYER2);
    newEvent.setEventity(oRoutingPacket.destid);

    newEvent.setPacket(oRoutingPacket);
    long x;
    x = (new Double(this.getJimRand())).longValue();
    newEvent.setEventTime(new Date());
    addEvent(newEvent);
}

```

```

    }
    private void addEvent(Event event){
        eventList.add(event);
        System.out.println("addEvent ");
    }
    public Event getNextEvent(){
        System.out.println("getNextEvent ");
        Event tmpEvent=null;
        Iterator iterator = eventList.iterator();
        if (iterator.hasNext()){
            tmpEvent =(Event) iterator.next();
            eventList.remove(tmpEvent);
            return tmpEvent;
        }
        return null;
    }
    public void run(){
        Date time = null;
        int i,j;

        init();
        i=0;
        while (((event = getNextEvent()) != null) )
        {
            // runs in a infinite loop
            System.out.println("*****Loop :"+ i++);

            time = event.getEventTime();

            if (event.getEventType() == FROM_LAYER2 )
            {
                if (event.getEntity() == 0)
                    node0.rupdate(event.getPacket());

                else if (event.getEntity() == 1)
                    node1.rupdate(event.getPacket());
                else if (event.getEntity() == 2)
                    node2.rupdate(event.getPacket());
                else if (event.getEntity() == 3)
                    node3.rupdate(event.getPacket());

                else {
                    System.out.println(" unknown event entity\n");
                }
            }

        }

        System.out.println("Simulator Completed !");
    }
}

```

Simulator.java

Results:

Entering rtinit
 toLayer2
 addEvent

```
toLayer2
addEvent
via
D0 | 1 2 3
-----|-----
1| 1 999 999
2| 999 3 999
3| 999 999 7
Entering rtinit
toLayer2
addEvent
toLayer2
addEvent
via
D0 | 1 2 3
-----|-----
1| 1 999 999
2| 999 3 999
3| 999 999 7
Entering rtinit
toLayer2
addEvent
toLayer2
addEvent
via
D0 | 1 2 3
-----|-----
1| 1 1 999
2| 999 0 4
3| 999 2 2
Entering rtinit
toLayer2
addEvent
toLayer2
addEvent
via
D0 | 1 2 3
-----|-----
1| 999 999 999
2| 999 2 2
3| 999 4 0
getNextEvent
*****Loop :0
Entering rtupdate
toLayer2
addEvent
toLayer2
addEvent
toLayer2
addEvent
toLayer2
addEvent
via
D0 | 1 2 3
-----|-----
1| 1 999 999
```

2| 999 3 999
3| 999 999 7
getNextEvent
*****Loop :1

Entering rtupdate
toLayer2
addEvent
toLayer2
addEvent
via
D0 | 1 2 3

----|-----

1| 999 999 999
2| 999 2 2
3| 999 4 0
getNextEvent

*****Loop :2

Entering rtupdate
toLayer2
addEvent
toLayer2
addEvent
via
D0 | 1 2 3

----|-----

1| 1 1 999
2| 2 0 4
3| 8 2 2
getNextEvent

*****Loop :3

Entering rtupdate
toLayer2
addEvent
toLayer2
addEvent
via
D0 | 1 2 3

----|-----

1| 1 999 999
2| 2 3 999
3| 8 999 7
getNextEvent

*****Loop :4

Entering rtupdate
toLayer2
addEvent
toLayer2
addEvent
toLayer2
addEvent
toLayer2
addEvent

----|-----

via
D0 | 1 2 3

----|-----

1| 999 3 999

```
2| 999 2 2
3| 999 4 0
getNextEvent
*****Loop :5
Entering rtupdate
toLayer2
addEvent
toLayer2
addEvent
via
D0 | 1 2 3
----|-----
1| 1 4 999
2| 999 3 999
3| 999 5 7
getNextEvent
*****Loop :6
Entering rtupdate
via
D0 | 1 2 3
----|-----
1| 1 999 10
2| 2 3 9
3| 8 999 7
getNextEvent
*****Loop :7
Entering rtupdate
via
D0 | 1 2 3
----|-----
1| 1 1 5
2| 2 0 4
3| 8 2 2
getNextEvent
*****Loop :8
Entering rtupdate
via
D0 | 1 2 3
----|-----
1| 1 1 5
2| 2 0 4
3| 8 2 2
getNextEvent
*****Loop :9
Entering rtupdate
via
D0 | 1 2 3
----|-----
1| 1 999 10
2| 2 3 9
3| 8 999 7
getNextEvent
*****Loop :10
Entering rtupdate
via
D0 | 1 2 3
```

```
---|-----  
1| 1 1 5  
2| 2 0 4  
3| 8 2 2  
getNextEvent  
*****Loop :11  
Entering rupdate  
via  
D0 | 1 2 3  
---|-----  
1| 1 999 10  
2| 2 3 9  
3| 8 999 7  
getNextEvent  
*****Loop :12  
Entering rupdate  
via  
D0 | 1 2 3  
---|-----  
1| 1 999 10  
2| 2 3 9  
3| 8 999 7  
getNextEvent  
*****Loop :13  
Entering rupdate  
via  
D0 | 1 2 3  
---|-----  
1| 1 1 5  
2| 2 0 4  
3| 8 2 2  
getNextEvent  
*****Loop :14  
Entering rupdate  
via  
D0 | 1 2 3  
---|-----  
1| 999 3 999  
2| 999 2 2  
3| 999 4 0  
getNextEvent  
*****Loop :15  
Entering rupdate  
via  
D0 | 1 2 3  
---|-----  
1| 1 4 999  
2| 999 3 999  
3| 999 5 7  
getNextEvent  
*****Loop :16  
Entering rupdate  
via  
D0 | 1 2 3  
---|-----  
1| 1 4 999
```



```
2| 3 3 999
3| 999 5 7
getNextEvent
*****Loop :17
Entering rtupdate
via
D0 | 1 2 3
----|-----
1| 999 3 999
2| 1001 2 2
3| 999 4 0
getNextEvent
*****Loop :18
Entering rtupdate
via
D0 | 1 2 3
----|-----
1| 1 999 10
2| 2 3 9
3| 8 999 7
getNextEvent
*****Loop :19
Entering rtupdate
via
D0 | 1 2 3
----|-----
1| 1 1 5
2| 2 0 4
3| 8 2 2
getNextEvent
*****Loop :20
Entering rtupdate
via
D0 | 1 2 3
----|-----
1| 1 6 10
2| 2 3 9
3| 8 999 7
getNextEvent
*****Loop :21
Entering rtupdate
via
D0 | 1 2 3
----|-----
1| 1 3 5
2| 2 0 4
3| 8 2 2
getNextEvent
*****Loop :22
Entering rtupdate
via
D0 | 1 2 3
----|-----
1| 1 3 5
2| 2 0 4
3| 8 5 2
```

```
getNextEvent
*****Loop :23
Entering rtupdate
via
D0 | 1 2 3
----|-----
1| 1 6 10
2| 2 3 9
3| 8 8 7
getNextEvent
Simulator Completed !
```

Assignment 5 Streaming Video with RTSP and RTP

The Code

In this lab you 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). Your task is to implement the RTSP protocol in the client and implement the RTP packetization in the server.

We will provide you code that implements the RTSP protocol in the server, the RTP de-packetization in the client, and takes care of displaying the transmitted video. You do not need to touch this code.

Classes

There are 4 classes in the assignment.

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. Below is what the interface looks like. *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 do not need to modify this class.

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 and you do not need to modify it (but see [Optional Exercises](#)). *You will need to complete the first constructor of this class to implement RTP-packetization of the video data.* The second constructor is used by the client to de-packetize the data. You do not need to modify that.

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

```
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 (we have provided one example file [movie.Mjpeg](#)). The file format is described in the [Appendix](#).

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

1. Client

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 lab 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 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 lab 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 state diagram.

2. Server

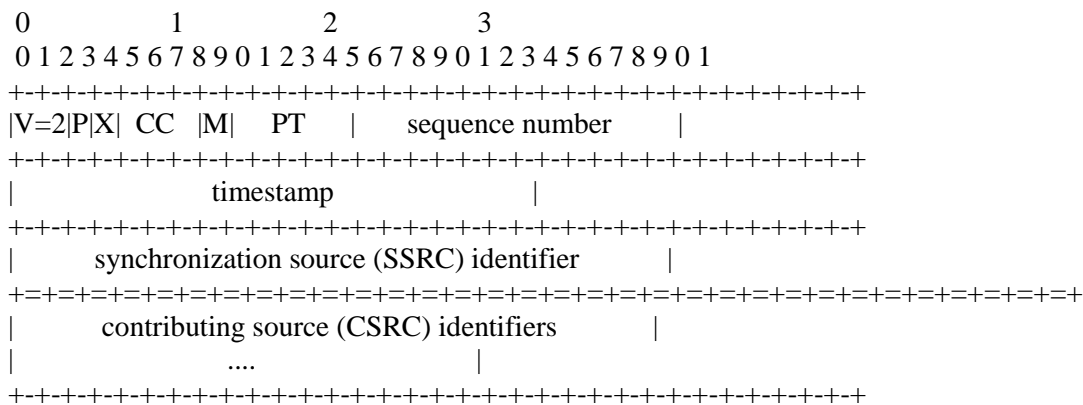
On the server you will need to implement the packetization of the video data into RTP packets. For this you will need to create the packet, set the fields in the packet header, and copy the payload (i.e., one video frame) into the packet.

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.



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.

If you're not comfortable setting bits, you can find more information in [the Java Tutorial](#).

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
```

```
27 . . . . . 20
```

Code : The Code will be provided in next semester

```
/* -----  
Server  
usage: java Server [RTSP listening port]  
----- */  
  
import java.io.*;  
import java.net.*;  
import java.awt.*;  
import java.util.*;  
import java.awt.event.*;  
import javax.swing.*;  
import javax.swing.Timer;
```



```

public class Server extends JFrame implements ActionListener {

    //RTP variables:
    //-----
    DatagramSocket RTPsocket; //socket to be used to send and receive UDP packets
    DatagramPacket senddp; //UDP packet containing the video frames

    InetAddress ClientIPAddr; //Client IP address
    int RTP_dest_port = 0; //destination port for RTP packets (given by the RTSP Client)

    //GUI:
    //-----
    JLabel label;

    //Video variables:
    //-----
    int imagenb = 0; //image nb of the image currently transmitted
    VideoStream video; //VideoStream object used to access video frames
    static int MJPEG_TYPE = 26; //RTP payload type for MJPEG video
    static int FRAME_PERIOD = 100; //Frame period of the video to stream, in ms
    static int VIDEO_LENGTH = 500; //length of the video in frames

    Timer timer; //timer used to send the images at the video frame rate
    byte[] buf; //buffer used to store the images to send to the client

    //RTSP variables
    //-----
    //rtsp states
    final static int INIT = 0;
    final static int READY = 1;
    final static int PLAYING = 2;
    //rtsp message types
    final static int SETUP = 3;
    final static int PLAY = 4;
    final static int PAUSE = 5;
    final static int TEARDOWN = 6;

    static int state; //RTSP Server state == INIT or READY or PLAY
    Socket RTSPsocket; //socket used to send/receive RTSP messages
    //input and output stream filters
    static BufferedReader RTSPBufferedReader;
    static BufferedWriter RTSPBufferedWriter;
    static String VideoFileName; //video file requested from the client
    static int RTSP_ID = 123456; //ID of the RTSP session
    int RTSPSeqNb = 0; //Sequence number of RTSP messages within the session

    final static String CRLF = "\r\n";

    //-----
    //Constructor
    //-----
    public Server(){

        //init Frame
        super("Server");

        //init Timer
        timer = new Timer(FRAME_PERIOD, this);
        timer.setInitialDelay(0);
        timer.setCoalesce(true);

        //allocate memory for the sending buffer

```

```

buf = new byte[15000];

//Handler to close the main window
addWindowListener(new WindowAdapter() {
    public void windowClosing(WindowEvent e) {
        //stop the timer and exit
        timer.stop();
        System.exit(0);
    }
});

//GUI:
label = new JLabel("Send frame #      ", JLabel.CENTER);
getContentPane().add(label, BorderLayout.CENTER);
}

//-----
//main
//-----
public static void main(String argv[]) throws Exception
{
    //create a Server object
    Server theServer = new Server();

    //show GUI:
    theServer.pack();
    theServer.setVisible(true);

    //get RTSP socket port from the command line
    int RTSPport = Integer.parseInt(argv[0]);

    //Initiate TCP connection with the client for the RTSP session
    ServerSocket listenSocket = new ServerSocket(RTSPport);
    theServer.RTSPsocket = listenSocket.accept();
    listenSocket.close();

    //Get Client IP address
    theServer.ClientIPAddr = theServer.RTSPsocket.getInetAddress();

    //Initiate RTSPstate
    state = INIT;

    //Set input and output stream filters:
    RTSPBufferedReader = new BufferedReader(new InputStreamReader(theServer.RTSPsocket.getInputStream()));
    RTSPBufferedWriter = new BufferedWriter(new OutputStreamWriter(theServer.RTSPsocket.getOutputStream()));

    //Wait for the SETUP message from the client
    int request_type;
    boolean done = false;
    while(!done)
    {
        request_type = theServer.parse_RTSP_request(); //blocking

        if (request_type == SETUP)
        {
            done = true;

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

            //Send response
            theServer.send_RTSP_response();
        }
    }
}

```

```

        //init the VideoStream object:
        theServer.video = new VideoStream(VideoFileName);

        //init RTP socket
        theServer.RTPsocket = new DatagramSocket();
    }

//loop to handle RTSP requests
while(true)
{
    //parse the request
    request_type = theServer.parse_RTSP_request(); //blocking

    if ((request_type == PLAY) && (state == READY))
    {
        //send back response
        theServer.send_RTSP_response();
        //start timer
        theServer.timer.start();
        //update state
        state = PLAYING;
        System.out.println("New RTSP state: PLAYING");
    }
    else if ((request_type == PAUSE) && (state == PLAYING))
    {
        //send back response
        theServer.send_RTSP_response();
        //stop timer
        theServer.timer.stop();
        //update state
        state = READY;
        System.out.println("New RTSP state: READY");
    }
    else if (request_type == TEARDOWN)
    {
        //send back response
        theServer.send_RTSP_response();
        //stop timer
        theServer.timer.stop();
        //close sockets
        theServer.RTSPsocket.close();
        theServer.RTPsocket.close();

        System.exit(0);
    }
}

//-----
//Handler for timer
//-----
public void actionPerformed(ActionEvent e) {

    //if the current image nb is less than the length of the video
    if (imagenb < VIDEO_LENGTH)
    {
        //update current imagenb
        imagenb++;
    }
}

```

```

    try {
        //get next frame to send from the video, as well as its size
        int image_length = video.getnextframe(buf);

        //Builds an RTPpacket object containing the frame
        RTPpacket rtp_packet = new RTPpacket(MJPEG_TYPE, imagenb, imagenb*FRAME_PERIOD, buf,
image_length);

        //get to total length of the full rtp packet to send
        int packet_length = rtp_packet.getlength();

        //retrieve the packet bitstream and store it in an array of bytes
        byte[] packet_bits = new byte[packet_length];
        rtp_packet.getpacket(packet_bits);

        //send the packet as a DatagramPacket over the UDP socket
        senddp = new DatagramPacket(packet_bits, packet_length, ClientIPAddr, RTP_dest_port);
        RTPsocket.send(senddp);

        //System.out.println("Send frame #" + imagenb);
        //print the header bitstream
        rtp_packet.printhead();

        //update GUI
        label.setText("Send frame #" + imagenb);
    }
    catch(Exception ex)
    {
        System.out.println("Exception caught: "+ex);
        System.exit(0);
    }
}
else
{
    //if we have reached the end of the video file, stop the timer
    timer.stop();
}
}

//-----
//Parse RTSP Request
//-----
private int parse_RTSP_request()
{
    int request_type = -1;
    try{
        //parse request line and extract the request_type:
        String RequestLine = RTSPBufferedReader.readLine();
        //System.out.println("RTSP Server - Received from Client:");
        System.out.println(RequestLine);

        StringTokenizer tokens = new StringTokenizer(RequestLine);
        String request_type_string = tokens.nextToken();

        //convert to request_type structure:
        if ((new String(request_type_string)).compareTo("SETUP") == 0)
            request_type = SETUP;
        else if ((new String(request_type_string)).compareTo("PLAY") == 0)
            request_type = PLAY;
        else if ((new String(request_type_string)).compareTo("PAUSE") == 0)
            request_type = PAUSE;
        else if ((new String(request_type_string)).compareTo("TEARDOWN") == 0)

```

```

        request_type = TEARDOWN;

    if (request_type == SETUP)
    {
        //extract VideoFileName from RequestLine
        VideoFileName = tokens.nextToken();
    }

    //parse the SeqNumLine and extract CSeq field
    String SeqNumLine = RTSPBufferedReader.readLine();
    System.out.println(SeqNumLine);
    tokens = new StringTokenizer(SeqNumLine);
    tokens.nextToken();
    RTSPSeqNb = Integer.parseInt(tokens.nextToken());

    //get LastLine
    String LastLine = RTSPBufferedReader.readLine();
    System.out.println(LastLine);

    if (request_type == SETUP)
    {
        //extract RTP_dest_port from LastLine
        tokens = new StringTokenizer(LastLine);
        for (int i=0; i<3; i++)
            tokens.nextToken(); //skip unused stuff
        RTP_dest_port = Integer.parseInt(tokens.nextToken());
    }
    //else LastLine will be the SessionId line ... do not check for now.
}
catch(Exception ex)
{
    System.out.println("Exception caught: "+ex);
    System.exit(0);
}
return(request_type);
}

//-----
//Send RTSP Response
//-----
private void send_RTSP_response()
{
    try{
        RTSPBufferedWriter.write("RTSP/1.0 200 OK"+CRLF);
        RTSPBufferedWriter.write("CSeq: "+RTSPSeqNb+CRLF);
        RTSPBufferedWriter.write("Session: "+RTSP_ID+CRLF);
        RTSPBufferedWriter.flush();
        //System.out.println("RTSP Server - Sent response to Client.");
    }
    catch(Exception ex)
    {
        System.out.println("Exception caught: "+ex);
        System.exit(0);
    }
}
}

```

Server.java

```

//VideoStream

import java.io.*;

```

```

public class VideoStream {

    FileInputStream fis; //video file
    int frame_nb; //current frame nb

    //-----
    //constructor
    //-----
    public VideoStream(String filename) throws Exception{

        //init variables
        fis = new FileInputStream(filename);
        frame_nb = 0;
    }

    //-----
    // getnextframe
    //returns the next frame as an array of byte and the size of the frame
    //-----
    public int getnextframe(byte[] frame) throws Exception
    {
        int length = 0;
        String length_string;
        byte[] frame_length = new byte[5];

        //read current frame length
        fis.read(frame_length,0,5);

        //transform frame_length to integer
        length_string = new String(frame_length);
        length = Integer.parseInt(length_string);

        return(fis.read(frame,0,length));
    }
}

```

VideoStream.java

```

//class RTPpacket

public class RTPpacket{

    //size of the RTP header:
    static int HEADER_SIZE = 12;

    //Fields that compose the RTP header
    public int Version;
    public int Padding;
    public int Extension;
    public int CC;
    public int Marker;
    public int PayloadType;
    public int SequenceNumber;
    public int TimeStamp;
    public int Ssrc;

    //Bitstream of the RTP header
    public byte[] header;

    //size of the RTP payload
    public int payload_size;
}

```

```

//Bitstream of the RTP payload
public byte[] payload;

//-----
//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] = ...
    // .....

    //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)
    //.....

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

}

//-----
//Constructor of an RTPpacket object from the packet bistream
//-----
public RTPpacket(byte[] packet, int packet_size)
{
    //fill default fields:
    Version = 2;
    Padding = 0;
    Extension = 0;
    CC = 0;
    Marker = 0;
    Ssrc = 0;

    //check if total packet size is lower than the header size
    if (packet_size >= HEADER_SIZE)

```

```

{
    //get the header bitstream:
    header = new byte[HEADER_SIZE];
    for (int i=0; i < HEADER_SIZE; i++)
        header[i] = packet[i];

    //get the payload bitstream:
    payload_size = packet_size - HEADER_SIZE;
    payload = new byte[payload_size];
    for (int i=HEADER_SIZE; i < packet_size; i++)
        payload[i-HEADER_SIZE] = packet[i];

    //interpret the changing fields of the header:
    PayloadType = header[1] & 127;
    SequenceNumber = unsigned_int(header[3]) + 256*unsigned_int(header[2]);
    TimeStamp = unsigned_int(header[7]) + 256*unsigned_int(header[6]) + 65536*unsigned_int(header[5]) +
16777216*unsigned_int(header[4]);
}
}

//-----
//getpayload: return the payload bistream of the RTPpacket and its size
//-----
public int getpayload(byte[] data) {

    for (int i=0; i < payload_size; i++)
        data[i] = payload[i];

    return(payload_size);
}

//-----
//getpayload_length: return the length of the payload
//-----
public int getpayload_length() {
    return(payload_size);
}

//-----
//getlength: return the total length of the RTP packet
//-----
public int getlength() {
    return(payload_size + HEADER_SIZE);
}

//-----
//getpacket: returns the packet bitstream and its length
//-----
public int getpacket(byte[] packet)
{
    //construct the packet = header + payload
    for (int i=0; i < HEADER_SIZE; i++)
        packet[i] = header[i];
    for (int i=0; i < payload_size; i++)
        packet[i+HEADER_SIZE] = payload[i];

    //return total size of the packet
    return(payload_size + HEADER_SIZE);
}

//-----
//gettimestamp

```



```

//-----
public int gettimestamp() {
    return(TimeStamp);
}

//-----
//getsequencenumber
//-----
public int getsequencenumber() {
    return(SequenceNumber);
}

//-----
//getpayloadtype
//-----
public int getpayloadtype() {
    return(PayloadType);
}

//-----
//print headers without the SSRC
//-----
public void printhead()
{
    //TO DO: uncomment
    /*
    for (int i=0; i < (HEADER_SIZE-4); i++)
    {
        for (int j = 7; j>=0 ; j--)
            if (((1<<j) & header[i] ) != 0)
                System.out.print("1");
            else
                System.out.print("0");
            System.out.print(" ");
        }

    System.out.println();
    */
}

//return the unsigned value of 8-bit integer nb
static int unsigned_int(int nb) {
    if (nb >= 0)
        return(nb);
    else
        return(256+nb);
}
}

```

RTPpacket.java

```

/* -----
Client
usage: java Client [Server hostname] [Server RTSP listening port] [Video file requested]
----- */

import java.io.*;

```

```

import java.net.*;
import java.util.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.Timer;

public class Client{

    //GUI
    //----
    JFrame f = new JFrame("Client");
    JButton setupButton = new JButton("Setup");
    JButton playButton = new JButton("Play");
    JButton pauseButton = new JButton("Pause");
    JButton tearButton = new JButton("Teardown");
    JPanel mainPanel = new JPanel();
    JPanel buttonPanel = new JPanel();
    JLabel iconLabel = new JLabel();
    ImageIcon icon;

    //RTP variables:
    //-----
    DatagramPacket rcvdp; //UDP packet received from the server
    DatagramSocket RTPsocket; //socket to be used to send and receive UDP packets
    static int RTP_RCV_PORT = 25000; //port where the client will receive the RTP packets

    Timer timer; //timer used to receive data from the UDP socket
    byte[] buf; //buffer used to store data received from the server

    //RTSP variables
    //-----
    //rtsp states
    final static int INIT = 0;
    final static int READY = 1;
    final static int PLAYING = 2;
    static int state; //RTSP state == INIT or READY or PLAYING
    Socket RTSPsocket; //socket used to send/receive RTSP messages
    //input and output stream filters
    static BufferedReader RTSPBufferedReader;
    static BufferedWriter RTSPBufferedWriter;
    static String VideoFileName; //video file to request to the server
    int RTSPSeqNb = 0; //Sequence number of RTSP messages within the session
    int RTSPid = 0; //ID of the RTSP session (given by the RTSP Server)

    final static String CRLF = "\r\n";

    //Video constants:
    //-----
    static int MJPEG_TYPE = 26; //RTP payload type for MJPEG video

```

```

//-----
//Constructor
//-----
public Client() {

    //build GUI
    //-----

    //Frame
    f.addWindowListener(new WindowAdapter() {
        public void windowClosing(WindowEvent e) {
            System.exit(0);
        }
    });

    //Buttons
    buttonPanel.setLayout(new GridLayout(1,0));
    buttonPanel.add(setupButton);
    buttonPanel.add(playButton);
    buttonPanel.add(pauseButton);
    buttonPanel.add(tearDownButton);
    setupButton.addActionListener(new setupButtonListener());
    playButton.addActionListener(new playButtonListener());
    pauseButton.addActionListener(new pauseButtonListener());
    tearDownButton.addActionListener(new tearDownButtonListener());

    //Image display label
    iconLabel.setIcon(null);

    //frame layout
    mainPanel.setLayout(null);
    mainPanel.add(iconLabel);
    mainPanel.add(buttonPanel);
    iconLabel.setBounds(0,0,380,280);
    buttonPanel.setBounds(0,280,380,50);

    f.getContentPane().add(mainPanel, BorderLayout.CENTER);
    f.setSize(new Dimension(390,370));
    f.setVisible(true);

    //init timer
    //-----
    timer = new Timer(20, new timerListener());
    timer.setInitialDelay(0);
    timer.setCoalesce(true);

    //allocate enough memory for the buffer used to receive data from the server
    buf = new byte[15000];
}

```

```

//-----
//main
//-----
public static void main(String argv[]) throws Exception
{
    //Create a Client object
    Client theClient = new Client();

    //get server RTSP port and IP address from the command line
    //-----
    int RTSP_server_port = Integer.parseInt(argv[1]);
    String ServerHost = argv[0];
    InetAddress ServerIPAddr = InetAddress.getByName(ServerHost);

    //get video filename to request:
    VideoFileName = argv[2];

    //Establish a TCP connection with the server to exchange RTSP messages
    //-----
    theClient.RTSPsocket = new Socket(ServerIPAddr, RTSP_server_port);

    //Set input and output stream filters:
    RTSPBufferedReader = new BufferedReader(new
InputStreamReader(theClient.RTSPsocket.getInputStream() ));
    RTSPBufferedWriter = new BufferedWriter(new
OutputStreamWriter(theClient.RTSPsocket.getOutputStream() ));

    //init RTSP state:
    state = INIT;
}

//-----
//Handler for buttons
//-----

//.....
//TO COMPLETE
//.....

//Handler for Setup button
//-----
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);

RTPsocket.setSoTimeout( 5);
//set TimeOut value of the socket to 5msec.
//....

}
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 = ....
    state = READY;
    //System.out.println("New RTSP state: ....");
}
} //else if state != INIT then do nothing
}
}

//Handler for Play button
//-----
class playButtonListener implements ActionListener {
    public void actionPerformed(ActionEvent e){

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

        if (state == READY)
        {
            //increase RTSP sequence number
            //.....
RTSPSeqNb += 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
    {
        state = PLAYING;
        //change RTSP state and print out new state
        //.....
        System.out.println("New RTSP state: ...");

        //start the timer
        timer.start();
    }
} //else if state != READY then do nothing
}

//Handler for Pause button
//-----
class pauseButtonListener implements ActionListener {
    public void actionPerformed(ActionEvent e){

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

        if (state == PLAYING)
            {
                RTSPSeqNb += RTSPSeqNb;
                //increase RTSP sequence number
                //.....

                //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
                    {
                        state = READY;
                        //change RTSP state and print out new state
                        //.....
                        System.out.println("New RTSP state: ...");

                        //stop the timer
                        timer.stop();
                    }
            }
        //else if state != PLAYING then do nothing
    }
}

```

```

}

//Handler for Teardown button
//-----
class tearButtonListener implements ActionListener {
    public void actionPerformed(ActionEvent e){
        RTSPSeqNb += RTSPSeqNb;
        //System.out.println("Teardown Button pressed !");

        //increase RTSP sequence number
        // .....

        //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
        {
            state = INIT;
            //change RTSP state and print out new state
            //.....
            System.out.println("New RTSP state: ...");

            //stop the timer
            timer.stop();

            //exit
            System.exit(0);
        }
    }
}

//-----
//Handler for timer
//-----

class timerListener implements ActionListener {
    public void actionPerformed(ActionEvent e) {

        //Construct a DatagramPacket to receive data from the UDP socket
        rcvdp = new DatagramPacket(buf, buf.length);

        try{
            //receive the DP from the socket:
            RTPsocket.receive(rcvdp);

            //create an RTPpacket object from the DP

```

```

RTPpacket rtp_packet = new RTPpacket(rcvdp.getData(), rcvdp.getLength());

//print important header fields of the RTP packet received:
System.out.println("Got      RTP      packet      with      SeqNum      #
"+rtp_packet.getsequencenumber()+" TimeStamp "+rtp_packet.gettimestamp()+" ms, of type
"+rtp_packet.getpayloadtype());

//print header bitstream:
rtp_packet.printhead();

//get the payload bitstream from the RTPpacket object
int payload_length = rtp_packet.getpayload_length();
byte [] payload = new byte[payload_length];
rtp_packet.getpayload(payload);

//get an Image object from the payload bitstream
Toolkit toolkit = Toolkit.getDefaultToolkit();
Image image = toolkit.createImage(payload, 0, payload_length);

//display the image as an ImageIcon object
icon = new ImageIcon(image);
iconLabel.setIcon(icon);
}
catch (InterruptedException iioe){
    //System.out.println("Nothing to read");
}
catch (IOException ioe) {
    System.out.println("Exception caught: "+ioe);
}
}
}

//-----
//Parse Server Response
//-----
private int parse_server_response()
{
    int reply_code = 0;

    try{
        //parse status line and extract the reply_code:
        String StatusLine = RTSPBufferedReader.readLine();
        //System.out.println("RTSP Client - Received from Server:");
        System.out.println(StatusLine);

        StringTokenizer tokens = new StringTokenizer(StatusLine);
        tokens.nextToken(); //skip over the RTSP version
        reply_code = Integer.parseInt(tokens.nextToken());

        //if reply code is OK get and print the 2 other lines
        if (reply_code == 200)

```



```

    {
        String SeqNumLine = RTSPBufferedReader.readLine();
        System.out.println(SeqNumLine);

        String SessionLine = RTSPBufferedReader.readLine();
        System.out.println(SessionLine);

        //if state == INIT gets the Session Id from the SessionLine
        tokens = new StringTokenizer(SessionLine);
        tokens.nextToken(); //skip over the Session:
        RTSPid = Integer.parseInt(tokens.nextToken());
    }
}
catch(Exception ex)
{
    System.out.println("Exception caught: "+ex);
    System.exit(0);
}

return(reply_code);
}

//-----
//Send RTSP Request
//-----

//.....
//TO COMPLETE
//.....

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

        //write the CSeq line:
        //.....

        //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 ( request_type == "SETUP"){
            RTSPBufferedWriter.write("Transport: RTP/UDP; client_port= " + RTP_RCV_PORT +
            CRLF);

```

```

    }
    else
        RTSPBufferedWriter.write("Session: " + RTSPid + CRLF);

    RTSPBufferedWriter.flush();
    }
    catch(Exception ex)
    {
        System.out.println("Exception caught: "+ex);
        System.exit(0);
    }
}

} //end of Class Client

```

Client.java

Results:

```

RTSP/1.0 200 OK
CSeq: 1
Session: 123456
RTSP/1.0 200 OK
CSeq: 2
Session: 123456
New RTSP state: ...
Got RTP packet with SeqNum # 0 TimeStamp 0 ms, of type 0
Got RTP packet with SeqNum # 0 TimeStamp 0 ms, of type 0
Got RTP packet with SeqNum # 0 TimeStamp 0 ms, of type 0
Got RTP packet with SeqNum # 0 TimeStamp 0 ms, of type 0
Got RTP packet with SeqNum # 0 TimeStamp 0 ms, of type 0
RTSP/1.0 200 OK
CSeq: 4
Session: 123456
New RTSP state: ...
RTSP/1.0 200 OK
CSeq: 8
Session: 123456
New RTSP state: ...
Got RTP packet with SeqNum # 0 TimeStamp 0 ms, of type 0
Got RTP packet with SeqNum # 0 TimeStamp 0 ms, of type 0
Got RTP packet with SeqNum # 0 TimeStamp 0 ms, of type 0
Got RTP packet with SeqNum # 0 TimeStamp 0 ms, of type 0
Got RTP packet with SeqNum # 0 TimeStamp 0 ms, of type 0
RTSP/1.0 200 OK
CSeq: 16

```

Session: 123456
New RTSP state: ...