CS 176c Homework 4:
Streaming Video with RTSP and RTP

Due: June 7

In this assignment you will develop a streaming video client that communicates using the Real-Time Streaming Protocol (RTSP) and sends data using the Real-time Transfer Protocol (RTP). In the following exercises, you will use this client, along with random UDP traffic, to analyze network performance. For these exercises, you should use your own laptop, and run tcpdump or wireshark to monitor the packets as they are transmitted. If you do not have a laptop, let your professor or TA know and one can be loaned to you for completing the experimentation portion of this assignment.

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. It is your responsibility to complete the RTP client code, write a UDP Traffic Simulator, host one (or more) server instances, and perform an analysis of two wireless networks using these programs.

Note 1: RTSP, the Real-Time Streaming Protocol, is a very simple HTTP-like protocol that is used to control streaming of video. It has play, pause, teardown, and other commands for streaming and controlling video. Google it for more information.

Note 2: Students using OSX Lion will have issues viewing the video. This will not affect your ability to complete this assignment.

Part 1: RTP Client

We will provide the python code described below, which partially implements a full RTSP/RTP video client. Your task is to implement the RTSP protocol in the client. The client is complete except for a few select methods: openRtpPort, sendRtspRequest, and parseRtspReply. Read and understand the program, study the RTSP protocol, then complete these methods.

Code

Client, ClientLauncher

The ClientLauncher starts the Client and the user interface that you use to send RTSP commands and to display the video. In the Client class, you will need to implement the actions that are taken when the buttons are pressed. You do not need to modify the ClientLauncher module.

ServerWorker, Server

These two modules implement the server that responds to the RTSP requests and streams back the video. The RTSP interaction is already implemented and the ServerWorker calls methods from the RtpPacket class to packetize the video data. You do not need to modify these modules.

RtpPacket

This class is used to handle the RTP packets. It has separate methods for handling the received packets at the client side and you do not need to modify them. The Client also de-packetizes (decodes) the data.

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:

```
python ClientLauncher.py server_host server_port  RTP_port  video_file
```

where server_host is the name of the machine where the server is running, server_port is the port that the server is listening on, RTP_port is the port where the RTP packets are received, and video_file is the name of the video file you want to request (we have provided one example file movie.Mjpeg).

The client opens a connection to the server and pops up a window like this:

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

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

The server always replies to all the messages that the client sends. The code 200 means that the request was successful while the codes 404 and 500 represent file_not_found error and connection error, respectively. In this lab, you do not need to implement any other reply codes. For more information about RTSP, please see RFC 2326.

The Client
Your first task is to implement the RTSP protocol on the client side. To do this, you need to complete the functions that are called when the user clicks on the buttons on the user interface. You will need to implement the actions for the following request types. When the client starts, it also opens the RTSP socket to the server. Use this socket for sending all RTSP requests.

SETUP

- Send SETUP request to the server. You will need to insert the Transport header in which you specify the port for the RTP data socket you just created.
- Read the server's response and parse the Session header (from the response) to get the RTSP session ID.
- Create a datagram socket for receiving RTP data and set the timeout on the socket to 0.5 seconds.

PLAY

- Send PLAY request. 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 the 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 the server's response.

TEARDOWN

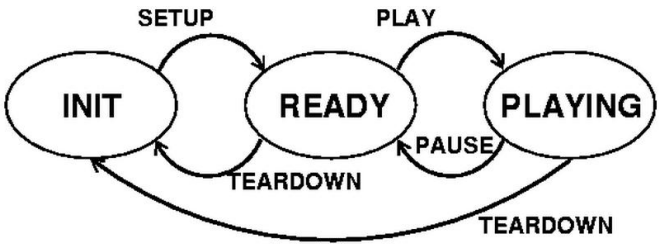
- Send TEARDOWN request. You must insert the Session header and use the session ID returned in the SETUP response. You must not put the Transport header in this request.
- Read the server's response.

Cseq Field

You must insert the CSeq header field in every request you send. The value of the CSeq header field is an integer that starts at 1 and is incremented by one for each request you send. The Cseq value is included with the reply by the server. See the example below.

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. The client changes state when it receives a reply from the server according to the following state diagram.



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 do not use sophisticated parsing methods, and they expect the header fields to be in the order you see below.

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

Part 2: UDP Traffic Simulator

Once you have your RTSP code working, your next step is to write a simple python program, called TrafficSimulator.py . You will use this program in Parts 2 and 3 to create additional traffic on the wireless network, allowing you to analyze the resulting changes in performance to the video you are streaming.

Your program should conform to the following usage statement:

```
python TrafficSimulator.py  server_host  server_port
packets_per_second   payload_size_in_bytes    seconds_to_run
```

where server_host is the name of the machine where the server is running, packets_per_second is the number of packets to send each second, payload_size_in_bytes is the payload size of each packet you will send, and seconds_to_run is the number of seconds the program will run for.

Your program should use these parameters to send UDP packets to the indicated host at the desired rate. Each packet's payload should contain your cs account name, followed by information about the TrafficSimulator instance. Pad the payload with the correct amount of random data to make len(name + info + random_string) == payload_size_in_bytes. You must put your cs account name in every packet generated by TrafficSimulator. You should also include the settings in your packet data. Your packets look as follows:

username:packet_number_in_sequence:packets_per_second:payload_size_in_bytes:running_time_in_seconds:random_data

You can write your program using a simple connectionless UDP socket in python. Please be conscious of the fact that you are creating a program capable of generating large amounts of traffic. Ensure that your program terminates cleanly and is not run with unwilling hosts as the target of your data flows. **You may want to experiment with creating interfering flows from the server to your client, from the client to the server, and from the client to a third machine.**

Testing your TrafficSimulator
When you are confident in the correctness of your TrafficSimulator, begin capturing packets in Wireshark and execute the following commands:

```
python TrafficSimulator.py  your-server  4567  10  50  10
```

This command should send 10 packets per second, each containing a 50 byte payload, for 10 seconds.

Verify that this is what occurs. To do so, answer (and turn in your answers to) the following questions:

Question 1: How many packets total were sent? What was the total size of each packet (don't forget about header data)? What is the average data rate of this flow, in bytes/second? What percentage of the flow is overhead?

Now execute: `python TrafficSimulator.py  your-server  4567  1000  50  5`

This command should send 1000 packets per second, each containing a 50 byte payload, for 5 seconds.

Question 2: Is your program capable of sending 1000 packets in one second? What is the recorded time of the first packet sent? The last packet? Describe how packet creation is distributed over the five seconds of program execution. What is the average data rate of this flow, in bytes/second?

Now execute: `python TrafficSimulator.py  your-server  4567  1  4000  1`

Question 3: What happens at the data link layer when you send a single, 4000 byte payload? Write a formula to calculate the percentage overhead required to transmit an X byte payload when packet fragmentation occurs as you observe it in this example. (Hint: You may find the modulus operator useful.) If you plug in 4000 for X, do your experimental results reflect the results of your formula?

Now execute: `python TrafficSimulator.py  your-server  4567  1  100000  1`

Question 4: What is the result when you attempt to send a 100,000 byte payload?

Part 3: Experimental Section

In CSIL, there are 2 wireless routers, 176C_A and 176C_B. One of these routers will provide some quality of service, the other will not. Your goal is to analyze how your RTP video client performs while connected to each router, in varying network traffic conditions. You are required to measure and **automatically** create graphs (using xgraph or gnuplot) using data on loss, jitter, bandwidth utilization, and the number of flows. This will allow you to compare the router that uses quality of service settings to the one that does not.

For this part of the lab, you should position yourself near the routers (in the front room of CSIL) and capture in Promiscuous or Monitor mode. This will allow you to capture all traffic on the network, from you and your classmates. Your analysis MUST consider all streams, not just your own, when calculating bandwidth utilization and number of flows.

To generate these graphs, you should write scripts that use a tool like dpkt, tshark, grep, perl's Net::Pcap::Easy or tcpdump's built in filters to parse your pcap capture files. You will need to calculate and extract the relevant statistics and data from the packets you are interested in. Your scripts should use as their input one or more *pcap* capture files, and produce as their output either gnuplot graphs as images (PNG, PDF, or JPEG), or produce a plain text file suitable for use as a gnuplot data source.

Note: Now would be a good time to refer to the last page for an example on how to use gnuplot

To perform these experiments, you will need to use the CS_176A and CS_176B routers in CSIL. To use CS_176A or CS_176B, you must assign yourself a static IP address with the following settings (the same for both routers):

IP_address:	Your static IP from the table below.
Subnet Mask:	255.255.255.0
Router/Gateway:	192.168.1.1

For your experiment, you should operate one server instance on a CSIL machine. You may use the python server code, or the java server code provided on Gauchospace, called ServerJava.tar. Each should be equivalent.

Here's the easiest way to launch a java server instance on the CSIL machines. First, download the ServerJava.tar and movie.Mjpeg files onto your machine, and in the same directory launch the following commands:

```
tar xvf ServerJava.tar
javac *.java
java Server [rtsp_port]
```
(i.e. java Server 4567)

You run your server on CSIL using:

```
java Server [rtsp_port]
```

Once your server is running on a csil machine (i.e. kenny) you may use the ClientLauncher on your laptop to communicate with your server. So for example, a student could start a server on kenny with the above code, then call (on his own laptop):

```
python ClientLauncher.py kenny.cs.ucsb.edu   4567   10020  movie.Mjpeg
```

Note: Check the list for your assigned IP address and port, which are sufficient to determine "ownership" for all RTP streams on the network.

Perform the following experiments in order to observe the impact of background traffic on the video flow. Answer the following questions to demonstrate your understanding of the performance you observe.

The Experiment:
→ Connect to the 176C_A router in CSIL.

Create a single RTP video stream, streaming the file 'movie.Mjpeg'. Play the Movie file, and graph the utilization of the wireless network from SETUP to TEARDOWN. Create a network baseline graph showing the bandwidth used by your RTP stream, by other students' RTP streams, and by any TrafficSimulator streams. Use one or more TrafficSimulator instances to gradually increase the amount of interfering traffic on the network. **You should monitor the network and consider other student's experiments and bandwidth utilization when you set your own, in order to get a diverse set of experimental data.** You should consider scripting the creation of these TrafficSimulator instances, so your experiments are easily repeatable (you will be repeating them on the other router!) Ensure you are capturing packets and logging them to one or multiple PCAP files. Use your observations from your captured traffic traces to perform the performance analysis.

Note: For those not using an OSX machine, we strongly encourage you to also observe the quality of the video to see first-hand the impact on the video.

Question 5: How does the number of traffic streams affect the RTP stream? At what traffic level do you notice performance degradation? Significant performance degradation? Can you cause the RTP flow to become unusable? To attempt this, you can vary not just the number of interfering traffic streams, but also the size of the packets they send and the frequency of packet transmissions. **Ensure you consider all traffic on the network, not just your own, in your calculations.** Provide a general discussion about the impact of varying packet sizes, packet transmission rates, and number of packet streams on the RTP video flow. What is the impact on the packet reception? **For those not on an OSX system**, at what points do you notice any visible impact on the video?

Your baseline bandwidth graph should have time on the X axis, and bandwidth utilization on the y axis. Use 2 lines on one plot: one to show utilization by your RTP/RTSP streams, and the other to show the total amount of traffic on the network over time. You should run for a sufficiently long time to experience a variety of network conditions. Use long running Traffic Simulator instances launched over several minutes so you can maintain and measure somewhat stable network conditions.

Now using those same packet captures to statistically validate your intuition in your discussion, provide 6 graphs. The graphs should have either Loss or Jitter on the Y axis, and the X axis should have either the total number of UDP traffic streams (for some constant packet size and interpacket spacing), total bandwidth utilization, or interpacket transmission rate (for a constant number of traffic streams and packet size). With the two y-axis options and three x-axis options, you should produce six graphs.

Question 6: How do loss rates and jitter change as UDP traffic increases? As the number of UDP flows increase? Use your graphs to validate your reasoning.

→ Connect to the 176C_B router in CSIL. Repeat the above experiment.

Question 7: How are the results different with 176C_B than 176C_A?

Question 8: Identify which router is actively using QoS settings to prioritize RTP traffic. Describe in two or three sentences what effect this has.

Question 9: On the router employing QoS, repeat the above experiment, but create multiple RTP flows in addition to UDP flows. How does this impact the loss rates and jitter for each RTP flow? Provide any relevant graphs and explain your results. In your explanation, be sure to explain your experiment – how many RTP flows did you have, what were the characteristics of the UDP flows, etc.

Submission

Ensure you have answered all questions and provided all requested data from this lab. Make sure your submission and source code is readable, well organized, and easy to follow!

You should turn in HW4 by putting the following files into a .tar.gz archive and submitting them to the correct GauchoSpace turn-in link:

Client.py TrafficSimulator.py LASTNAME_HW4_Writeup.pdf, all scripts you wrote for the experimental section, including scripts to get data from pcap files, and your xgraph or gnplot files for each graph.

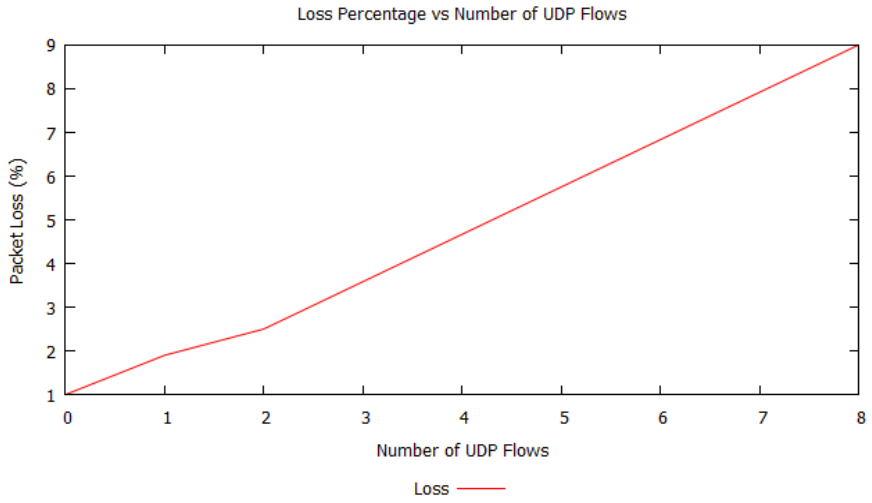
GNUPLOT: If you've never used gnuplot, the files it uses for input have a simple structure:

```
# Input file: test.dat
# An example gnuplot input file (format only, made up numbers)
# Number of UDP Flows      Loss Percentage       Jitter
0                     1               2
1                     1.9             3
2                     2.5             3
8                     9               10
```

The file should be represented as columns of numbers, separated by whitespace. Lines that begin with "#" are ignored. To create a graph, you associate columns with axes with gnuplot's 'plot' command. In the example below, we plot column 1 of test.dat (number of flows) as the X axis, and column 2 (loss percentage) as the Y axis. Note that we start counting at 1 in gnuplot, not 0.

If you create the above test.dat file, inputting the following lines into the gnuplot console (or adding and running the file directly) will produce the example output graph provided below.

```
#!/usr/bin/gnuplot -persist # This will let you run this as an
executable.
set xrange [0:8]
set title "Loss Percentage vs Number of UDP Flows"
set key below
set xtic auto
set ytic auto
set xlabel "Number of UDP Flows"
set ylabel "Packet Loss (%)"
plot 'test.dat' using 1:2 title "Loss" with lines
```



Note that this is simply an example to get you familiar with the functioning and basic behavior of gnuplot. More detailed documentation and examples can be found on gnuplot.info. When you create your graphs, ensure you use appropriate `xrange` and `yrange` values; that your plot includes a title, axis labels, and labels for each flow; and that appropriate units are used where required. You may plot multiple lines on the same graph using a single plot command. For example:

```
plot 'test.dat' using 1:2 title "Loss" with lines, \
               using  1:3 title "Jitter" with lines
```