**Computer Networks, Fall 2016**

Instructor: Shashi Prabh

**Lab 4: Multicasting multimedia over IP**

**Due:** Nov. 23 (Firm deadline.)

# 1 Objective

In this lab you will gain experience with IP multicast and gain insights into the nature of multimedia traffic, especially, video traffic. Recall that the video traffic is projected to be the main driver of mobile data traffic growth. You will be developing an Internet TV/Radio application that uses multicast. This lab combines the experience you have gained in all three of the previous labs: (a) sending data over a TCP connection (b) sending multimedia data over UDP, and (c) sending data in structured manner so that applications implemented differently running on different platforms remain inter-operable. You'll be working with Any Source Multicast (ASM) model. In this model, multicast messages are identified by the multicast group address alone, and more than one sender can exist in a group. Since in the ASM model, any message sent to a multicast address is received by all nodes that have joined the group, it is important that different teams use disjoint addresses to avoid getting streams mixed-up. Therefore, we'll be assigning multicast address blocks to individual teams. Further, you can assume that the scope of multicast is limited to a LAN.

To get you started, I have put `sender.c` and `receiver.c` at the course website. Compile these by issuing commands `gcc sender.c -o sender` and `gcc receiver.c -o receiver`. Run the sender by issuing command `./sender 239.192.1.10`. In a separate terminal, run `sudo ./receiver 239.192.1.10 [interface_name]` to display all the strings sent to the multicast group `230.192.1.10` on the screen (the optional argument is the name of network interface, which defaults to `wlan0` if omitted). Before proceeding any further, understand and experiment with the code. Extending it to send and receive multimedia files, an exercise that you did in Lab 2, is rather straightforward.

This lab is divided into two main parts. The emphasis of Part-I is on socket programming, multicast and interoperability. Part-II is concerned with user interface and multimedia content management.

Have fun!

# 2 Part-I

## 2.1 Client

The client establishes a control channel using TCP socket and receives multimedia streams over a UDP socket. The client is used *interactively*. Upon receiving the station list from the sever, the client displays it on the screen. The user specifies the station that (s)he wants to receive by specifying the corresponding channel number. The client then sets-up a multicast receiver accordingly. The client also provides pause, restart, station change and terminate (the application) features by displaying the respective commands at start time (you can just use letters `P, R, C, X`, for example). The `pause` command closes the multicast reception and the following `restart` command resumes it, keeping the station

the same. `terminate` stops any ongoing multicast reception and exits the client. The `station_change` command results in a fresh station list query form the server followed by displaying of the list on the screen. Keeping the client interactive means that the user must be able to perform any of the above mentioned functionalities at any time of his/her choosing. For this part, the user provides inputs through `stdin`.
The server also periodically multicasts song/video related information over the stream information port. Your client should display and update this information as well. The station list sent by the server will be an ordered list of entries of the form
`(station_num, name, description, mcast_address, port, info_port, bit-rate)`.
The client will set-up (UDP) multicast socket to receive the data and set-up a buffer of adequate size for playing it. In particular, the size of the buffer should be large enough to hold $t$ seconds worth of data where $t$ is a user specified input. You will need to take both $t$ and the bit-rate into account while setting-up the buffer. Note that if a user changes station, you may need to recompute the buffer size. You may assume that the bit-rate remains fixed for a given station.
It may be helpful to store the data in a file (like you did in Lab 2) and set a player like `VLC or ffplay` to play that file before moving on to implement the buffer setting-up. Please make sure to use only the block of multicast addresses assigned to your team, which is `239.192.x.0 - 230.192.x.255` where `x` is your team number.

### 2.2 Server

The server listens over a TCP socket on port `5432` for station information request messages. It sends the list of all stations currently available in the format specified in the next section. In addition to multicasting multimedia data, it also multicasts information about current stream as well as the next stream. The latter needs to be sent about once every second and at the time when a new stream starts.
Note that multicast will be sent over UDP. Controlling data rate is extremely important. Since we are not implementing any feedback to the server from the client, buffer overflow is possible. The outgoing data rate should be nearly the same as the stream bit rate.

### 2.3 Content

Audio/video must first be converted to a format which can be played from any point. For this lab, you are free to choose existing audio/video converters to prepare the content to be streamed and to render the content on the client side. You can NOT, however, use any existing application for transmitting and receiving content over the network.
You may find the tools `ffmpeg` for file conversion and `ffprobe` for getting file information useful. Do `man ffmpeg` and visit `www.ffmpeg.org` for documentation of these tools. For a start, you can covert files to MPEG transport stream (MPEG-TS) container format, which is suitable for live streaming, using `ffmpeg` form command-line as follows:
`ffmpeg -i inputfile.mp4 -f mpegts streamable_output.mp4`
The output can be conformed to match certain dimensions or bit-rate by using various switches and filters.

### 2.4 Message format

A client developed by one team should work with the server developed by any other team. Therefore, it is necessary to stick to the message format described below. You are free

to define additional message formats, possibly to provide other features. If you define additional message formats and features, make sure to document them in your lab report.

### 2.4.1 Client to server messages

```
station_info_request:
uint8_t type = 1;
```

### 2.4.2 Server to client messages

```
site_info:
uint8_t  type = 10;
uint8_t  site_name_size;
char     site_name[ site_name_size ];
uint8_t  site_desc_size;
char     site_desc[ site_desc_size ];
uint8_t  station_count;
station_info station_list [station_count];

station_info:
uint8_t  station_number;
uint8_t  station_name_size;
char     station_name[ station_name_size ];
uint32_t multicast_address;
uint16_t data_port;
uint16_t info_port;
uint32_t bit_rate;
```

### 2.4.3 Stream information messages (multicast)

```
song_info:
uint8_t  type = 12;
uint8_t  song_name_size;
char     song_name[ song_name_size ];
uint16_t remaining_time_in_sec;
uint8_t  next_song_name_size;
char     next_song_name[ next_song_name_size ];
```

# 3 Part II

This part is mainly concerned with user experience and content management. You can choose to do either one. Doing both will earn extra credit.

## 3.1 Option 1: Client Interface

Develop a web interface or a GUI for interacting with the multimedia server. The content delivery must still be done by multicast. You can however choose to use existing applications/tools on the client side. The visual appearance of the client interface and quality of playback will be the main evaluation criteria for this part.

### 3.2 Option 2: Content management

The goal of this option to develop a campus radio/TV station. You can choose to do either one or both of the following: (a) Mimic a live radio/TV broadcast system where students can present shows using their personal computers and a webcam alone (b) Automatically pull radio/TV stream from the web and feed it to your server so that a subset of the stations remain live all the time. Understanding processing and storage demand is crucial here.

### 3.3 Grading

Your code must be well commented. Submit code on blackboard. Submit a pdf of the report on Blackboard as well as a hard copy before the deadline. In the report describe the design decisions taken by your team, details of the method used in your system to determine data rate, playback buffer size on the client side, include and screen-shots of the client interface. You should highlight all the features (specified here or added by your team) at one place. List the breakdown of contributions of team members in the report. A part of the grade will be based on individual efforts.

| Part | Component | Weight (%) | Notes |
|---|---|---|---|
| I | Client | 25 | Demo Part-I by Nov 16 |
| I | Server | 25 | ,, |
| II | GUI (*) | 20 | Demo on deadline date |
| II | Content (*) | 20 | ,, |
| Both | Overall design | 15 | |
| Both | Report | 15 | Submit hard copy as well as pdf on Blackboard |

*: One of the two options is mandatory. Doing both earns 20% extra credit