Machine Problem 2: Video Streaming

CS414 Spring 2011: Multimedia Systems
Instructor: Klara Nahrstedt

Posted: Mar 7, 2011
Due: 11:59pm Mar 30, 2011

Introduction

Twenty years ago, the only method you could watch a movie was to either go to a theatre or buy/rent a video tape. However, things have changed a lot since then. People now prefer watching everything "online". YouTube becomes the place where we watch most video. Even the DVD movie rental companies, like NetFlix, are investing most of their money in expanding the online streaming business. What is the magic that makes all this revolution happen? Network Streaming!

In this first MP, you have already learned how to capture the raw images and raw audio data from external devices, and got some experience in video/audio playback. Therefore, we now focus on how to stream the captured video/audio contents from one machine to another machine through connecting network. In this MP, you will achieve a deep understanding of the concepts like bandwidth, QoS, and why they matter so much. Moreover, you will have all important components ready for your next MP: building a videoconferencing system.

You are required to work on Linux for this machine problem. You can use the Linux workstations in the EWS lab room SC216 and SC220. A group directory and an SVN account have been set up for everyone. You can use your own machine as well. The recommended Linux version to install is Ubuntu 10.04 or newer. C/C++ is the recommended language for your program. You still need the Logitech Webcam you have borrowed and use your own microphone/headphone for the audio recording/playback.

Problem Description

Implement two programs named as *recorder* and *player*. *recorder* should perform exactly the same as the requirements in MP1 with one exception: the captured video/audio data should not be saved to the file but sent to *player* through network. *player* should run on the other machine simultaneously with *recorder* and be able to play the video/audio data received from *recorder*. Here are the detailed feature requirements for the two programs:

Required Features (80 points)

1. *Video Streaming (10 points)*: *recorder* can display the captured video frames in the video window and at the same time stream the frames to *player*. The video resolution should be no smaller than 320 $\times$240 pixels and the frame rate should be no less than 15 frames per second. The video displayed by *player* should have similar quality (frame rate, image quality, etc) to the video displayed by *recorder*. Print the fps when the program quits.

2. *Audio Streaming (10 points)*: *recorder* streams the captured audio data to *player* and *player* can play the received audio data.

3. *A/V Sync (10 points)*: *player* should synchronize the playback of video and audio streams all the time.

4. *TCP/UDP (10 points)*: implement both TCP and UDP version of multimedia streaming.

5. *Bandwidth*: calculate the outgoing network bandwidth for *recorder* and the incoming network bandwidth for *player*. Print the stats every second.

6. *Latency (10 points)*: the latency of the audio/video playback by *player* should be minimized. Calculate the latency of your streaming system. Either *recorder* or *player* should print the latency every second. Any latency larger than 1 second gets zero point.

7. *Pause (10 points)*: *player* can pause and resume the audio/video playback. Pause stops the playback of audio immediately and freezes the video window on the last video frame. Resume starts the audio/video playback again. Please note that the audio/video data received by *player* during pause should be discarded so that the latency does not increase after resuming (The "pause/resume" here is different from the terms used for VCR or YouTube).

8. *Robustness (10 points)*: your programs are expected to run without crash for up to one hour.

Optional Features (50 points)

9. *GUI (5 points)*: design any graphic user interface can get 5 points.

10. *Name Server(10 points)*: set up a separate server for the translation from user names to IP addresses. *recorder* should register to the name server when starting. Then *player* does not need to know the IP address of *recorder* for connection. Instead, *player* will query the user name of *recorder* from the name server first for the IP address and then set up the connection.

11. *Remote Control (10 points)*: *player* can remotely pan/tilt the webcam controlled by *recorder*. Calculate the *interaction latency* (the delay time from when *player* receives the pan/tilt commands till when the first panned/tilted image frame displayed by *player* on screen). Print the stats for every action.

12. *Bandwidth Control (20 points)*: users can configure the actual network bandwidth usage of both *recorder* and *player*. The program should automatically decide how to translate the network bandwidth limitation to audio/video QoS parameters, transmit lower-quality video or audio to meet the bandwidth requirements, and control network traffic.

13. *RTP (5 points)*: use RTP for media streaming. You can build your own RTP implementation or re-use any open source RTP distribution.

Examples

If you design GUI for your programs, the user interface should be easy to understand. Add illustrations in your documents if anything is confusing. If you just design a command line program, here is an example for you to follow. Make sure to write usage explanation in your documents if your implementation is different from the example below.

For the required features, you can design *recorder* by taking one parameter to specify the port number which *recorder* will listen to.

recorder PORT

recorder should start before *player*. The video window is shown immediately after *recorder* starts although there is no *player* online. *player* should take at least three parameters to specify the IP address, port number, and transport protocol.

player IP_ADDRESS PORT TCP|UDP

where IP_ADDRESS is the IP address of the machine where *recorder* is running and the PORT is the port number which *recorder* is listening to. The third parameter specify whether TCP or UDP is used for streaming. After both *recorder* and *player* are connected, *recorder* should keep printing the information of outgoing bandwidth:

```
[1s] Outgoing Bandwidth: *** bps
[2s] Outgoing Bandwidth: *** bps
[3s] Outgoing Bandwidth: *** bps
......
```

player should keep printing the information of incoming bandwidth and playback latency.

```
[1s] Incoming Bandwidth: *** bps
[1s] Display Latency: *** ms
[2s] Incoming Bandwidth: *** bps
[2s] Display Latency: *** ms
[3s] Incoming Bandwidth: *** bps
[3s] Display Latency: *** ms
......
Playback Frame Rate: 15.4 fps
```

player can be paused and resumed by pressing the space bar. During the pause, the video window should freeze on the last video frame and no audio is played. *player* should keep printing the incoming bandwidth stats but not the latency time.

For the optional features, you are recommended to use the configuration file to manage all parameters. For example, *recorder* by default reads the configuration file `recorder.config` in the same directory for all input parameters. The configuration file `recorder.config` may look like the following:

```
LISTEN_PORT=9999
USER_NAME=simple_recorder
NAME_SERVER_IP=192.168.0.1
NAME_SERVER_PORT=8888
VIDEO_WIDTH=640
VIDEO_HEIGHT=480
VIDEO_FPS=30
VIDEO_DEVICE=/dev/video0
AUDIO_RATE=8000
AUDIO_BIT=8
AUDIO_CHANNEL=1
AUDIO_DEVICE=/dev/dsp
OUT_BW_LIMIT=500Kbps
```

player can use a similar configuration file `player.config` as follows:

```
NAME_SERVER_IP=192.168.0.1
NAME_SERVER_PORT=8888
RECORDER_NAME=simple_recorder
VIDEO_WIDTH=640
VIDEO_HEIGHT=480
VIDEO_FPS=30
VIDEO_DEVICE=/dev/video0
AUDIO_RATE=8000
AUDIO_BIT=8
AUDIO_CHANNEL=1
AUDIO_DEVICE=/dev/dsp
IN_BW_LIMIT=1Mbps
```

If you want to implement feature 10, you need to write a third program *name_server*, which should start before *recorder* and *player*. For feature 11, your *player* should print the information of interaction latency every time when pan/tilt is triggered. For feature 12, the bandwidth limitation can be any integer number from 50Kbps to 10Mbps. Your program is expected to (a) make sure the actual bandwidth usage (printed out every second) does not exceed the limitation; and (b) maintain the consistent and smooth audio/video streaming. Please note that the bandwidth limit applied to *recorder* and *player* can be different.

Of course, you are free to choose any other methods and formats for your input and output.

Submission

Reuse your MP1 project for this MP. Pack all source codes and documents into a zip file or tar ball and submit through Compass. **Make sure to name your zip or tar ball file in the format of "MP2_Groupxx.xxx". Late submissions will not be accepted this time!**

Source Code

You should put all your source codes (.h, .c or .cpp files), Makefile, and any open source libraries you use in the directory "src". **DO NOT** include any pre-compiled obj files (.o), binary execution, debug file, or pre-recorded media files in your "src" directory. If you have some other files to submit, create a directory "misc" for those files.

Documentation

You need to put two documents: user manual and development manual in the directory "doc". The user manual should include all instructions on how to compile and install your source code, and how to run your program and test all features. If you have designed any GUI, it is better to attach some screen shots to explain. The development manual should have the implementation details of all features. The implementation details include program flow, key data structures, media file formats, important algorithms, and so on.

Evaluation

Your solution is evaluated based on how many features you have implemented. In order to get full score (100 points) of this MP, you need to implement all 80-point required features and any

20-point optional features. If you get more than 100 points, you will get bonus points for your final score.

For each feature, you get 20% points if the feature is implemented and well documented; 20% if the source codes can compile and run; 20% if the feature is partially implemented; 20% if the feature is fully implemented; 20% if the program runs perfectly (no error, no crash, as good as other groups).

Special Rules

You will lose points if you do not follow these rules:

- You lose 10 points if you do not follow the instructions above to organize and name your submission.
- You lose 80% points if your program does not compile.
- You lose up to 20 points if you write only a few words in your documentation.
- You lose all points if you do not submit on time. There will be no exception this time!