SysProg Assignments 4 and 5: Audio Streaming

Arno Bakker Guillaume Pierre

Abstract

You are expected to build a streaming media environment, similar to RealPlayer or WindowsMedia. The application must be able to process the sound using filters. You are also asked to write a short report about your implementation.

Media Streaming Software

The media streaming application you will build consists of a network server that reads data from a media file and transmits it through the network, and a network client that receives the network stream and plays it to a sound card. The client and the server implement a communication protocol that you are supposed to define.

Both server and client must accept *filters* to modify the datastream. Filters are external libraries that can be loaded by your application at runtime ('plug-ins').

You are will write *at least 3 filters*, but are free to choose which actions they undertake. Naturally, complex or innovative filters will give you higher points. Be creative. Here are some examples that we accept:

1. Stereo to mono: transform a stereo file into mono to reduce the usage of network resources
2. Volume: adjust the volume of the audio file
3. Add noise to the audio file
4. Add echo to the audio file
5. Play the audio file faster or slower than normal
6. Buffering: introduce a buffer on the client-side to mask drops in throughput

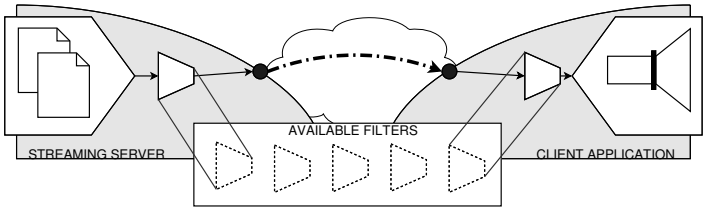

Figure 1: Design of the Streaming Media System

7. Compression: use a compression algorithm or tool to reduce bandwidth consumption (e.g., gzip, mp3). You are permitted to use external third-party libraries to perform the actual compression/decompression (but not for other functions!)

Most filters will require paired operation between server and client. If the server, for instance, compresses data using gzip, then the client must run the corresponding decompression filter on the other side. You will have to come up with an agreement ('protocol') through which the applications can communicate these demands. You can either create a separate control protocol, or add headers to the data-packets to achieve this. You must use UDP as the transport layer for your protocol.

We will supply code for reading in audio-files and sending sound to the speaker.

As this is quite a large exercise, we do not want you to write the complete system at once. Instead, we ask you to build your applications to respect the following deadlines.

Friday 21 October 2016 at 23:59: Basic Streaming Server

You are expected to write a simple network client/server pair.

- The audio server is called `audioserver`. It *must* not take any command-line parameter. This program expects to find WAV files in its current directory. It waits until it receives a request from a client containing the name of a file. When it has received a request, it opens the requested file and streams it to the client. When the streaming is finished, it waits for another client to connect.

- The audio client is called `audioclient`. It *must* have the following interface:
  `audioclient server_host_name file_name`
  The client sends a request to the server containing the name of the file to play. It then receives the audio stream from the server and plays it to the audio card.

All communication between the client and the server *must* be carried over UDP. The use of TCP is forbidden. You will be expected to define your own simple communication protocol. In particular, it is important that the server sends out the audio data at the right speed so that the client can play it. Which informations must be carried by each network packet? What should each program do when it receives a given message? Get inspiration from the Computer Networks class.

You are permitted to make the following assumptions:

1. The first UDP packet sent from the client to the server does not get lost in the network. Similarly, the first UDP packet sent from the server to the client does not get lost. On the other hand, you *cannot* assume that the subsequent packets will all arrive.

2. If a packet carrying some audio content or some acknowledgement gets lost, then it is acceptable that the client plays some silence instead of the expected sound. However, it *must* start playing again as soon as it starts receiving packets again. This means that your programs should not crash or get stuck if a packet gets lost.

You do *not* need to implement audio buffering at the client to prevent such silences from occurring when packets get lost. On the other hand, this may be an interesting filter to build for the next assignment.

3. Your server can be designed to serve only one client at a time. If the server is already busy with one client and another client tries to contact it, then the server can refuse to server the second client and return an error message to it.

   It is, however, *permitted* to design your server to handle multiple clients simultaneously. If it works, then this will gain you some extra points. Note that this makes the assignment much more difficult! I strongly recommend to start with a system that can serve a single client, then expand it to multiple clients if you feel like it.

4. Your programs do not need to interoperate with the programs returned by other students, or with any other existing piece of software. It is good enough if the client and server that you write work correctly with each other.

You are required to submit this assignment together with some documentation called `protocol.txt` or `protocol.pdf` which describes your communication protocol. You should describe the protocol in words, possibly augmented with ASCII art to describe packet layout, as in `https://tools.ietf.org/html/rfc7574#section-8.4`. In general, 2 pages should be sufficient for this. In any case *do not send documentation longer than 4 pages*. Also, please do not send Word files (you can use Word, but then please export the file to PDF). Also include the README file as usual, all together in an uncompressed `tar` file.

Note that the audio library needs some special support to play audio files. From the README:

> "Compile and link your program with `audio.c` Then run the program using the PulseAudio OSS Wrapper:
>
> ```
> padsp ./program
> ```
>
> This will ensure the `aud_writeinit()` function can find a `/dev/dsp` device to which it can write your samples, thereby playing them."

Grading: if your programs implement just the required features, are reasonably well programmed and work OK then you will get an 8. This means that in normal situations the music should be recognizable, and that a few packets can be dropped by the network without breaking the whole system. If your programs do not implement all this, or contain bugs, then your grade will be lower than 8. If your programs do more (such as streaming to multiple clients simultaneously) and they do not contain too many bugs then your grade will be greater than 8.

Sunday 30 October 2016 at 23:59: Advanced Streaming Server and Documentation

You are expected to extend your programs to allow the use of plug-in filters that can process the sound at the client and/or the server. You must implement 3 plugins. You are permitted to invent any plugin that you like.

The interface of your server should remain unchanged. The interface of your client *must* be the following:

```
audioclient server_host_name file_name filter_name [filter_parameter]
```

Your client must also work if no filter is specified. Please expand your documentation with the list of filters that you have implemented, the syntax of their parameters (if they have parameters), and the sound effect that they are supposed to have.

We will accept simple filters, such as streaming only one audio channel instead of stereo, adjusting the volume, etc. However, coming up with more sophisticated filters will give you extra points.

Hint: think *very carefully* about the way you will implement your filters and interface them with the rest of the system. This should be the major difficulty of this assignment.

Optional: program each filter as a separate dynamic library. Your programs must then load the appropriate library at runtime using `dlopen()` depending on the user's request. Doing so will give you extra points.

Optional: design your programs so that they can chain multiple filters (e.g., produce an echo and adjust volume at the same time). Doing so will give you extra points.

Grading: if your programs work correctly with 3 simple filters (at least one of them running at the server), then you will get an 8. If they do not work as expected, contain bugs or all filters run at the client side, your grade will be lower than 8. If you develop non-trivial filters, or implement one of the optional extensions, then your grade will be greater than 8.