CSE422 Computer Networks Spring 2017
Laboratory 3: Simple HTTP Live Streaming Client

Due: 23:59 Thursday, April 27, 2017

1 Goals

Gain experience and knowledge of streaming media using HTTP.

2 Overview

In this lab, you will implement a simple *streaming client* that downloads and plays videos using **HTTP Live Streaming (HLS)** protocol developed by Apple Inc. This lab will give you experience in streaming media and demonstrate to you another usage of HTTP.

This lab is worth for 8% if the final grade and is composed of 80 points. This lab is due no later than 23:59 (11:59 PM) on Thursday, April 27, 2017. No late submission will be accepted. No extension will be given for this lab. You will submit your lab using the *handin* utility (http://secure.cse.msu.edu/handin/).

3 HTTP Live Streaming (HLS)

The **HTTP Live Streaming (HLS)** protocol is a media streaming communication protocol based on HTTP, developed by Apple Inc. The basic idea behind HTTP Live Streaming is to break up a streaming file into a sequence of smaller *segments*. The segments are deployed on a regular HTTP server as web objects. Each of the segments has a *Uniform Resource Locator (URL)*. A *playlist* that indexes and locates the segments using their URLs is also deployed on the web server.

When a streaming client is instructed to play the file, the client first downloads the playlist. The client obtains the URLs of the media segments by examining the playlist. The client then downloads the media segments in the order provided by the playlist and plays the segments as if it were one single continuous media file. The communication between the client and the server is performed entirely using HTTP. Because HLS protocol is based on HTTP, HLS is able to benefit from services that are compatible with HTTP (such as traversing firewalls and proxy servers).

4 A Simple Video Player

In order to help you understand HTTP Live Streaming and focus on the implementation of the networking part, a simple video player class (`VideoPlayer`) and an example program (`simpleClient`) of this application are provided to you along with the skeleton code. [link]. Several supporting classes for HTTP are also provided along with this simple video player class.

The following description of the video player, combined with your studying the code (`simpleClient`, which is less than 70 lines of code) should be helpful in constructing your streaming client. The `simpleClient` can be used to open and play a local file.

```
Usage: ./simpleClient filename
```

Example invocation of the `simpleClient`:
`./simpleClient -f ~cse422b/web/fs14/lab3/whoa.mp4`

This invocation opens a local video file, reads the video file's content piece by piece and forwards/inputs the pieces to the `VideoPlayer` class. Similarly, your streaming client will be able to download a segment, open the segment and forward the segment to the `VideoPlayer` class for playback. Please refer to `simpleClient.cc` for more information.

This video player should be able to play most of video in MPEG format. A small sample video file is provided `~cse422b/web/fs14/lab3/whoa.mp4` and `~cse422b/web/fs14/lab3/small.mp4`.

5 Specification

To initiate a video streaming, the streaming client has to download the playlist, whose URL is given by the user in the command line. The streaming client only accept one argument, the URL of the playlist.

```
Usage: ./streamClient -p http://some_server/path/to/playlist.m3u8
```

Example invocation of the `streamClient`:
`./streamClient -p http://www.cse.msu.edu/~cse422b/fs14/lab3/whoa/whoa.m3u8`

The streaming client downloads the playlist specified by the user. A supporting library (`Playlist`) for handling the playlist is provided. The streaming client can parse the playlist using the `Playlist::parse` method and a list of segment URLs will be stored in `Playlist`. The streaming client then downloads the segments in the playlist one-by-one and forwards them to the `VideoPlayer`.

Both playlists and segments use `default transfer encoding`. You can easily create a function to download a web object in default transfer encoding using the code in `client.cc` from lab 2.

We summarize the operations of the streaming client as follows:

- Download the playlist from the URL provided by the user.
- Parse the playlist using the `Playlist` class's parse method.
- Create an instance of the `VideoPlayer` class.
- Download the segments in the parsed `Playlist` class. Forward the data downloaded to the `VideoPlayer` instance.

Your streaming client is expected to handle the following errors.

- The server does not exist (i.e., TCP connect fails, `gethostbyname` returns null ... etc)
- The playlist cannot be found on the server. (A 404 Not Found response is received.)
- The playlist URL is forbidden by the server. (A 403 Forbidden response is received.)
- The playlist cannot be parsed, such as parse failed or invalid playlist format.

The client is expected to output corresponding error message and terminate the program.

For other errors, the streaming handles the error by simply terminating the application.

5.1 Playlists

Two HLS videos are deployed in following URI.

- `http://www.cse.msu.edu/~cse422b/fs14/lab3/San_Diego_Clip/San_Diego_Clip.m3u8`
- `http://www.cse.msu.edu/~cse422b/fs14/lab3/centaur/centaur.m3u8`
- `http://www.cse.msu.edu/~cse422b/fs14/lab3/whoa/whoa.m3u8`

6 Guidelines

Working alone or in pairs. You may work on this assignment individually or in pairs (not in groups of 3 or more, however). If you prefer to work in a pair, **both** students must submit a copy of the solution and identify their MSU NetIDs in a README file. If you prefer to work individually, please clearly state that you are working individually and include your MSU NetID in the README file.

Programming Language. You must implement this project using `C++`. The skeleton code is written in `C++`. Please clearly state the command to compile your project submission in your README file.

Testing your code. Each Linux distribution might be slightly different. It is the students' responsibility to make sure that the lab submissions compile on *at least one* of the following machines in 3353 EB: `carl`, `ned`, `marge` or `skinner`. A statement must be provided in the README file's header. You will **not** be awarded any credit if your lab submission does not compile on any of those machines.

7 Deliverables

You must submit your lab using the *handin* utility (http://secure.cse.msu.edu/handin/). Please submit all files in your project directory. If you start your lab with the skeleton code, submit all files, even for those files that are not modified.

This lab is due no later than 23:59 (11:59 PM) on Thursday, April 27, 2017. No late submission will be accepted. No extension will be given.

The compilation must be done using a makefile. The code should compile and link on CSE machines in 3353 EB. You will not be awarded any points if your submission does not compile using the makefile. Please test your streaming client thoroughly before submitting the code.

A README file is required. A sample README file is also included in the skeleton code. You are also encouraged to include any relevant comments in the README file.

8 Grading

You will not be awarded any points if your submission does not compile.

```
General requirements: 5 points
_____  2 pts Coding standard, comments ... etc
_____  1 pts README file
```

```
_____  2 pts Descriptive messages/Reasonable output.

Streaming client: 75 points
_____  5 pts 404 Not Found handing
_____  5 pts 403 Forbidden handing
_____ 10 pts Download the playlist
_____  5 pts Parse the playlist correctly
_____ 30 pts Download the segments and play the video
_____ 10 pts User closes the video player before streaming
             is done
_____ 10 pts Clean up and error checking (-2 for each item.)
```

9 Notes

- Please develop your program on machines in CSE labs, instead of the servers. If possible, make sure your program compiles on following machines in 3353 EB: `carl`, `ned`, `marge`, `mcclure`, `apu`, `krusty`, `rod` or `skinner` before submitting.
- Please spend some time tracing the code in the provided classes. One should be able to build the entire streaming client using those classes. Tracing the `simpleClient` code would be a good start.

Please feel free to mail TA, Chin-Jung Liu, liuchinj AT cse.msu.edu for questions or clarifications.

10 Examples

10.1 403 Forbidden

```
>./streamClient -p http://www.cse.msu.edu/~cse422b/fs14/lab3/football/football.m3u8
Attempting to stream video from http://www.cse.msu.edu/~cse422b/fs14/lab3/football/footb
Request failed:
403 Forbidden
Aborting download.
Unable to download/parse playlist at http://www.cse.msu.edu/~cse422b/fs14/lab3/football/
Usage:  ./streamClient http://server.com/stream.m3u8
```

10.2 404 Not Found

```
>./streamClient -p http://www.cse.msu.edu/~cse422b/fs14/lab3/Not_Exist
Attempting to stream video from http://www.cse.msu.edu/~cse422b/fs14/lab3/Not_Exist
Request failed:
404 Not Found
Aborting download.
Unable to download/parse playlist at http://www.cse.msu.edu/~cse422b/fs14/lab3/Not_Exist
Usage:  ./streamClient http://server.com/stream.m3u8
```

10.3 Successful Playback

Note that the full path to the m3u8 is:
http://www.cse.msu.edu/~cse422b/fs14/lab3/San_Diego_Clip/San_Diego_Clip.m3u8

```
>./streamClient -p http://www.cse.msu.edu/~cse422b/fs14/lab3/San_Diego_Clip/San_Diego_Cl
Attempting to stream video from http://www.cse.msu.edu/~cse422b/fs14/lab3/San_Diego_Clip
Playlist has 99 segments.
Fetching segment 1
Fetching segment 2
Fetching segment 3
Fetching segment 4
Fetching segment 5
Fetching segment 6
Fetching segment 7
Fetching segment 8
......
Fetching segment 94
Fetching segment 95
Fetching segment 96
Fetching segment 97
Fetching segment 98
Fetching segment 99
Waiting for playback to finish.
```

The video playback window is shown in Figure 1.

10.4 Successful Playback

```
>./streamClient -p http://www.cse.msu.edu/~cse422b/fs14/lab3/centaur/centaur.m3u8
```

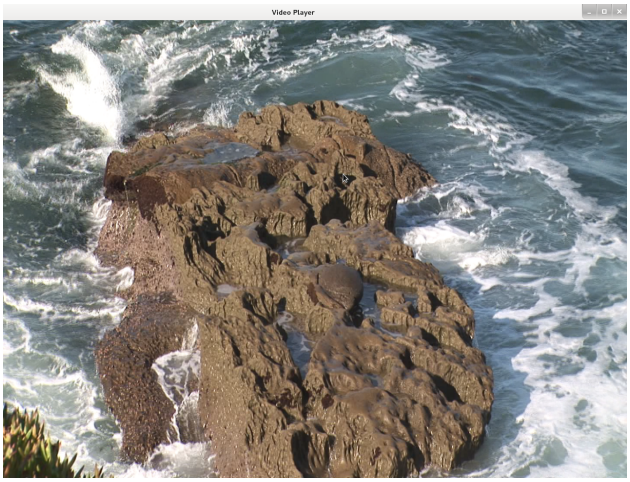

Figure 1: Video Playback

```
Attempting to stream video from http://www.cse.msu.edu/~cse422b/fs14/lab3/centaur/centau
Playlist has 5 segments.
Fetching segment 1
Fetching segment 2
Fetching segment 3
Fetching segment 4
Fetching segment 5
Waiting for playback to finish.
```

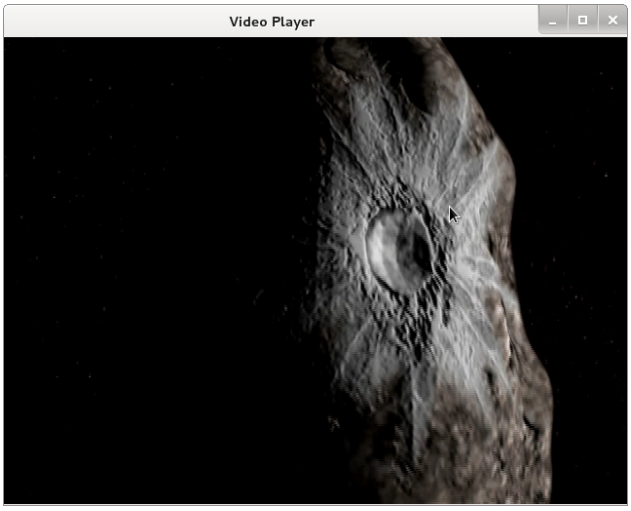

Figure 2: Video Playback

The video playback window is shown in Figure 2.