CS 5248: Systems Support for Continuous Media
Fall 2019

Project Assignment

Due: Report Draft – 11 Nov 2019 (11:59:59 pm)
Due: Code & Report – 14 Nov 2019 (11:59:59 pm)
Due: Presentation & Demo – 15 Nov 2019 (in class, 6:30 to 8:30 pm)

THIS IS A TEAM-OF-3-or-4 PROJECT WORK

REMEMBER TO CHECK FOR ANNOUNCEMENTS ON THE CLASS WEB SITE:
http://www.comp.nus.edu.sg/~cs5248/

TA: Abdelhak Bentaleb bentaleb@comp.nus.edu.sg
Lecturer: Roger Zimmermann rogerz@comp.nus.edu.sg

Project Assignment Description

In this project, your task is to build a DASH-compliant (Dynamic Adaptive Streaming over HTTP – that is, compatible with MPEG DASH standard) client-server-based live video streaming system on top of a LAMP stack (Linux, Apache, MySQL, PHP) or your own server framework.

The DASH approach of streaming is becoming very popular. It is basically a replacement for the RTSP/RTP/RTCP based approach to streaming, especially with Video on Demand (VoD). With DASH, the server is a simple HTTP web server (e.g., Apache). The media (video) is divided into small individually playable video segments which are, for example, 10 seconds long. These segments are also called *streamlets*. The client media player retrieves the streamlets from the web server, one at a time, and plays them without interruption. For the client to know the streamlet files that belong to a complete video, the server provides a *playlist* file (also called a Media Presentation Description (MPD)). The playlist file has a special format; it is basically an ***XML*** file for the MPEG DASH standard. To start streaming, the client player loads the playlist file and then starts to download the streamlets that are listed in this file. Your task is to generate such a playlist file onto the server from a media file and support VoD playback of those streamlets onto another client device (e.g., your laptop).

You will be given a number of utilities that will help you to get the project done. Additional information has been given during the Lecture 4 on **6 September 2019** and can be found in the slides for that lecture.

The project work is divided into three tasks: (1) create a mobile application that captures a live video and upload its streamlets to a server as they become available, (2) create server-side utilities that prepare incoming media files into streamlets serviceable via DASH playlist, and (3) create a media player application based on **dash.js** that plays a stored (VoD) DASH playlist, while adaptively switching and playing video streamlets.

Task 1 (Mobile Video Capture and Uploader): The mobile application, running on a Samsung tablet computer, is required to provide the following sub-tasks:

1) Capture a live video feed of 720p resolution at 30 fps from the device camera. You can use MediaCodec API from Android to encode frames with h.264 encoding with 5 Mbps bitrate.

2) Upload the captured frames to a web server reliably on-the-fly.

    a. Segment the original live feed into a number of self-contained 3-second-long MP4 segments before uploading on-the-fly. Segmentation can either be done first, before the upload, or in parallel, together with the upload.

    b. Use the HTTP POST method to deliver the segmented MP4 video feed to the server.

For you to understand the internal structure of the MP4 format, we intentionally ask you to segment the video at the mobile client (i.e., at the tablet computer), not at the server. To segment the video, you may use third-party libraries such as MP4Parser.

To upload the segmented video chunks reliably, you are required to design a simple protocol on top of HTTP, such as checking the current upload status or providing segments with a sequence number.

The following are additional functionalities that the mobile app can provide, which will receive extra credits:

3) Provide a resumed upload when the network connection is interrupted.

4) Retrieve the list of the uploaded videos available from the web server.

5) Videos should be playable live onto the client device during a session as well as stored on the server for on-demand playback.

Task 2 (Server-side DASH Preparation): The server-side utilities may be written in PHP (default setup) or any other web-framework of your choice (Django, Rails, Node.js, etc.)[1]. There is no restriction on the use of any other third-party software packages as long as such usage doesn't make your effort trivial. The minimum requirements for the server functionality are:

1) Receive segmented MP4 videos from a mobile client via HTTP POST and store them in a video repository location (i.e., directory) which you define. To keep track of the uploading status, you may use a MySQL database.

    a. You can use any segmentation tool such as ffmpeg, MP4Box, etc.

[1] For custom server-side setup, please contact the TA

b. Generate a playlist (i.e., generate an MPD file) and deliver it to a client (dash.js reference player) during a live DASH streaming session.

2) Transcode every received MP4 video segment into the following four encoded media versions:

   a. 720p: 1280x720 4000 kbps H.264/AVC and 128 kbps AAC audio (Very High)

   b. 480p: 854x480 2000 kbps H.264/AVC and 128 kbps AAC audio (High)

   c. 360p: 640x360 1000 kbps H.264/AVC and 128 kbps AAC audio (Medium)

   d. 240p: 426x240 700 kbps H.264/AVC and 64 kbps AAC audio (Low)

3) Convert every version into an MPEG-2 Transport Stream.

4) Provide MPD XML of DASH playlists. The MPD XML playlist should be playable from *your* dash.js MPEG-DASH player (see Task 3 below).

Task 3 (MPEG-DASH Player): The web-based media player application, based on dash.js (**v.2.9.3:** https://github.com/Dash-Industry-Forum/dash.js/archive/v2.9.3.zip), should retrieve the list of the uploaded videos available on your web server. Your player should allow on-demand playback of previously uploaded videos. It should support adaptive stream switching (termed Adaptive BitRate logic, i.e., ABR) ─ that means it adaptively chooses the most suitable quality for the next video streamlet to be downloaded based on your bandwidth estimation algorithm, while playing the current streamlet. The minimum subtasks for this task are:

1) Retrieve a list of videos available on your server and download the MPEG-DASH XML-formatted playlist file for the selected video (on-demand).

2) Read the playlist file and schedule retrieval of individual streamlets on-the-fly.

3) Implement and add your ABR algorithm (BBA, ELASTIC, FESTIVE, QDASH) to the dash.js.

   a. BBA algorithm:

      i. http://yuba.stanford.edu/~nickm/papers/sigcomm2014-video.pdf

   b. ELASTIC algorithm:

      i. https://c3lab.poliba.it/images/a/a1/Elastic-pv2013.pdf

   c. FESTIVE algorithm:

      i. https://conferences.sigcomm.org/co-next/2012/eproceedings/conext/p97.pdf

d. QDASH algorithm:

   i. http://www4.comp.polyu.edu.hk/~csrchang/mms12-qdash.pdf

e. To add your algorithm, you have to follow the same structure of how BOLA is added.

4) Switch the rendering of one video streamlet to another based on your ABR algorithm.

Note that this player does not have to support HLS. The suggested format of video streamlets is MP4 or m4s (very common).

The following additions are given extra credits.

1) Display the player status: show the bitrate selected and current network bandwidth estimation results visually.

2) Compare your ABR to the conventional, already built-in ABRs of dash.js (BOLA, throughput-based, and Dynamic).

All the above functionalities are required to run automatically, meaning that there is no manual work other than a user's upload request of a recorded video via the mobile app.

Reference and Software Information

All the students taking the CS5248 module will receive a user account on our server `monterosa.d2.comp.nus.edu.sg`. There would be one user account per team. The server is running the Ubuntu 18.04.2 LTS distribution of Linux. You will need to use `ssh` to connect to the machine. If you are using the default LAMP setup provided by us, your PHP scripts would be executed by the webserver from *your_home_dir*/`public_html` directory. For example, if `team0` wants to serve the file `info.php` it will be placed at `/home/team0/public_html/info.php` and would be available over the web at `http://monterosa.d2.comp.nus.edu.sg/~team0/info.php`. The server also has SSL enabled. So, teams can optionally use `https` in their GET and POST request URLs. It is however recommended to avoid SSL for requests involving file transfers such as the *streamlets*.

When segmenting the video with the mobile application, you may use the `MP4Parser` package. When developing server utilities, you are allowed to use any useful MPEG-4 parsing and transcoding utilities such as `MP4Box` or `ffmpeg`. For your convenience, you can use `ffmpeg`, `MP4Box`, and `mp4info`, etc., commands from /usr/local/bin on the monterosa server. If you need any further enhancements/additions to the provided default setup or you prefer to use a custom server-side setup other than the default LAMP stack, please contact the TA.

Additional information and links:

- Apple Live Streaming Internet Draft (http://tools.ietf.org/html/draft-pantos-http-live-streaming-23)
- Microsoft Smooth Streaming (http://www.iis.net/downloads/microsoft/smooth-streaming)
- Sample instruction how to build iPhone HTTP Streaming (http://www.ioncannon.net/programming/452/iphone-http-streaming-with-ffmpeg-and-an-open-source-segmenter/)
- MP4Parser https://github.com/sannies/mp4parser
- FFmpeg http://ffmpeg.org
- Bento4 package http://www.bento4.com/
- MP4Box http://www.videohelp.com/tools/mp4box
- Python Django https://www.djangoproject.com/
- dash.js https://github.com/Dash-Industry-Forum/dash.js

If you have any questions, email either the TA or the professor!

Submission Guidelines

1. Materials which you need to submit

(1) Your Android app source code and installable package (.apk file); your compiled/built dash.js file (dash.all.min.js file); your ABR algorithm file (ABR.js); your modified dash.js files; your PHP, Python or other framework's server-side code. Create a tarball (i.e., a tar file) or ZIP file of all your **sources** (please exclude sample videos, object files, etc.).

(2) A detailed README.txt file, that includes

a) Your name(s) and your username(s) on the host monterosa.
b) A brief description of each source file, how it works and is related to your project.

Please create a tarball (i.e., an archive created with the `tar` utility or the `zip` utility) that collects all your files into a package with your matriculation number. Then compress it (with `gzip`, in case you are using `tar`) before you submit the file. **Make sure that your code compiles without ANY errors!**

(3) Submit your report, presentation, and compressed source code package into the module's **LumiNUS Workbin** folder called 'Student Submissions' by 23:59 on 14 Nov 2019.

Note that the project demonstrations will be held on 15 Nov 2019 during the last CS5248 class (6:30 pm to 8:30 pm) and cannot be done later.

Grading Policy (40 full marks)

1. **Project Demonstration (25 marks)**. You will present your project in class on the last module day (15 November 2019). You will have approximately 25 minutes to demo your project. The demo should include (1) segmenting the source video **(5 marks)**, (2) playing the uploaded video from desktop browser **(5 marks)**, and (3) displaying the playlist and playing a video from it on your dash.js player while dynamically adapting to network conditions **(10 marks)**. Note that we will have a setup whereby we will synthetically vary the network bandwidth available to your DASH client. (4) demonstrating a VoD streaming session over DASH **(5 marks)**.

2. **Project Source Code (5 marks)**. We will examine your code. Criteria are, for example, how well it is documented, how well it is understandable, how robust it is (i.e., can it process all relevant input files), etc. We will also check whether your MPD can pass the MPEG-DASH MPD validator (see https://conformance.dashif.org/).

3. **Project Report (10 marks)**. Your write-up should be similar to a standard research survey paper. Please use a research paper template such as the ones from ACM or IEEE. For equations, please use math notations, not source code variables. Please include (at least) the following sections:
    a) Abstract (an overview of what the project includes)
    b) Introduction and motivation for your project work (your specific design and implementation choices)
    c) Describe some of the work that your implementation is related to. This does not have to be a complete survey of related work, just the most relevant techniques to your project. Include some references. Avoid using web links as references.
    d) Description of your implementation, i.e., details on what you did, any difficulties encountered, any special features or functionalities that you implemented, etc.
    e) Results, comparison and discussion, i.e., what should we learn from this and what could possibly be further improved. Do you have some experimental results? i.e., did you measure the latency or bandwidth adaptation?
    f) Other research directions regarding lowering end-to-end delays for MPEG-DASH streaming
    g) Conclusions
    h) References

    Most of the marks will be given for sections b), d), and e).

Additional Notes

No plagiarism is tolerated. You are encouraged to use the LumiNUS Forum and/or other channels made available by the teaching staff. While you can talk to other teams, you cannot copy source code and/or text in your report. If you are not sure whether something is permissible, either talk to the instructor or consult the university policy at http://www.usp.nus.edu.sg/curriculum/plagiarism.