National University of Singapore

School of Computing

CS2105 **Assignment 3** (5 marks) Semester 2 AY21/22

Submission Deadline

10th Apr 2022 (Sunday), 11:59 pm. 2 marks penalty will be imposed on late submission (late submission refers to submission or re-submission after the deadline). The submission folder will be closed on **13th Apr 2021 (Wednesday), 11:59 pm** and no late submission will be accepted afterwards.

Objectives

In this assignment, you will inspect three multimedia streaming applications that run on DASH, RTP, and WebRTC and discuss some important aspects that characterise these popular streaming standards. After completing this assignment, you should

- have a good understanding of how DASH, RTP and WebRTC work in real-life applications
- know how to use common network inspection tools for simple network debugging tasks
- know how to use FFmpeg for simple media processing.

This assignment is worth **5 marks** in total and they are split across **three exercises**. All the work in this assignment shall be completed **individually**.

Plagiarism Warning

You are free to discuss this assignment with your friends. **However, you should not share your answers, screenshots or network logs.** We highly recommend that you attempt this assignment on your own and figure things out along the way, as many resources are available online. The troubleshooting skills you learn will be very useful.

We employ a zero-tolerance policy against plagiarism. If a suspicious case is found, the student would be asked to explain their answers to the evaluator in person. Confirmed breach may result in zero marks for the assignment and further disciplinary action from the school.

Question & Answer

If you have any doubts about this assignment, please post your questions on Piazza forum before consulting the teaching team. **We will not debug programs for you.** However, we will help to clarify misconceptions or give necessary directions if required.

Setup

You should use your local machine for all tasks in this assignment. Note that you **should not** `ssh` into the `sunfire` server for this assignment. The tools you need (on your local machine) are -

1. Google Chrome or any other modern browser
2. Wireshark
3. FFmpeg

For Exercise 3, you are also required to activate an account (**24 hours** needed for activation) for YouTube Live.

Submission

1. For most of the questions, please enter your answers in **LumiNUS Quiz - Assignment 3**.
2. For the rest of the questions (e.g., Exercise 1 - Task 3), please prepare **a zip file** containing the submission files (details below) and submit it to the `Assignment_3_student_submission` folder. Name your zip file as **<Student_Number>.zip**, where **<Student_Number>** refers to your matric number that starts with the letter A. Note that the first and last letters of your student number should be capital letters. One example file name would be **A0112345X.zip**.

Your zip file should only contain the six submission files below -

<Student_Number>.zip contains:		
1	`e1_t3.txt`	Answers to Task 3 in Exercise 1.
2	`e1_t3.pcapng`	Wireshark log file for Task 3 in Exercise 1.
3	`e1_t4.png`	Screenshot for Task 4 in Exercise 1.
4	`e2_t2_output.avi`	The processed avi file for Task 2 in Exercise 2.
5	`e2_t2.txt`	Answers to Task 2 in Exercise 2.
6	`e3_t1.png`	Screenshot for Task 1 in Exercise 3.

Please ensure that your zip file has **no subfolders** (e.g., A0112345X, e1, e1_t3), otherwise you will be deducted by 2 points.

Exercise 1 - DASH (3 Marks)

In the lecture, we learnt about DASH (Dynamic Adaptive Streaming over HTTP), a pull-based streaming standard over HTTP. This exercise helps us understand how DASH works in real-life streaming applications. **There are four tasks in this exercise and we recommend completing the tasks in order.**

Recall the architecture of a streaming application built on DASH -

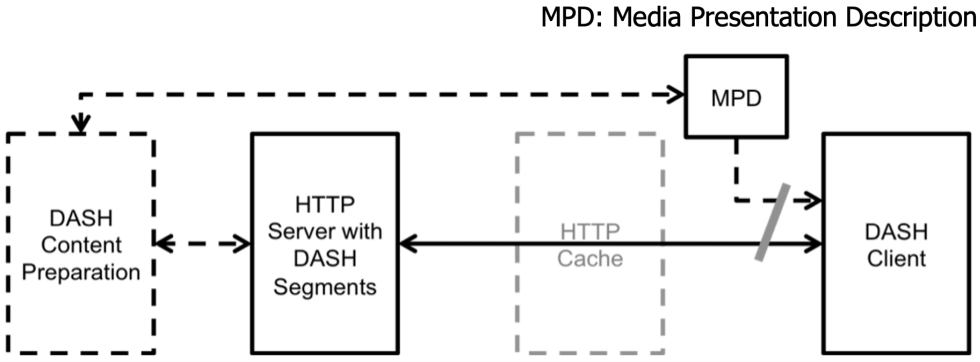

Figure 1: A streaming application built on DASH

The HTTP server provides the playlist and media segments (sometimes referred to as streamlets) for the DASH client to retrieve (i.e., "pull") on-demand.

Task 1: Inspecting the MPD Playlist (1 mark)

In this assignment, we will use our MPD playlist available at -

`http://monterosa.d2.comp.nus.edu.sg/~maylim/streaming/bbb.mpd`

Download the playlist file by pasting the MPD URL in your browser's address bar. View the playlist content using your favourite text editor and answer the following four discussion questions -

1. How many different video quality levels (resolution) are provided by this playlist?
2. What are the highest and lowest these quality levels (in resolution)?
3. Why do we need different quality levels? Please give an example where the player would retrieve the video at different quality levels.
4. Describe one advantage and one disadvantage of providing more quality levels for a video stream (e.g., up to 10 different quality levels).

[Submission]

Enter your answers clearly in LumiNUS Quiz - Assignment 3.

Task 2: Inspecting Video Segments (1 mark)

For our DASH client in this exercise, we will use an open source, browser-based video player provided by the DASH Industry Forum, available at[1]

`http://monterosa.d2.comp.nus.edu.sg/~maylim/dash.js-3.1.3/samples/dash-if-reference-player/`

Please access the player using any modern browser (we recommend using Google Chrome). The player also provides different playlist samples for users to try out (see the drop-down list under "Stream").

For this task, you should begin streaming the video provided by our playlist in Task 1 by replacing the MPD URL in the DASH player and clicking "Load". The link is also given below for convenience:

`http://monterosa.d2.comp.nus.edu.sg/~maylim/streaming/bbb.mpd`

Now let's inspect the video segments retrieved by the player. On your browser, open up the network inspection tool [2] to view the browser's network log.

While the player is playing (i.e., streaming) the video, you should see the network log being updated dynamically. This log contains the list of HTTP responses received by the client, including the video segments retrieved.

Pick one video segment from the list in the network log, download it and answer the following four discussion questions -

1. What is the URL of your selected video segment?
2. What is the file format of this video segment?
3. What is the duration of this video segment? (Note that all video segments from the same playlist should have the same duration.)
4. Different applications may use different video segment durations (typically between 2 and 10 seconds). From a networking perspective, describe one advantage and one disadvantage of using longer video segments (e.g., 10 seconds per segment).

[Submission]

Enter your answers clearly in LumiNUS Quiz - Assignment 3.

[1] If you get a URL "Not Found" error on your browser, please check that there is no additional space character in the link you pasted.

[2] For Google Chrome, right-click "Inspect" and click on the "Network" tab. For more details, refer to `https://developers.google.com/web/tools/chrome-devtools/network`.

Task 3: Inspecting HTTP-based Transfer (0.5 mark)

From Task 1 and 2, we can see that for DASH, the MPD playlist and video segments are retrieved over HTTP. Now let's inspect the HTTP request and response in greater detail using a software called **Wireshark**. A quick start can be found here[3].

Open Wireshark on the machine running the browser and begin capture. Following the same steps as Task 1, try downloading the MPD file again on your browser and observe the capture log being updated on Wireshark. You should notice that your capture log contains packets from different sources (thanks to background processes etc.), making network inspection difficult.

Now, let's restart the capture process, but this time using Wireshark's capture filters[4] **to capture only the relevant packets, e.g., from a certain IP address** involved in this particular request-response exchange for the MPD file download. Save this Wireshark capture log in the default `pcapng` file format.

Please also briefly describe the steps involved in the download procedure using such an **HTTP** based protocol.

[Submission]

- Save your log file as `e1_t3.pcapng`.
- Save your explanatory as `e1_t3.txt`.

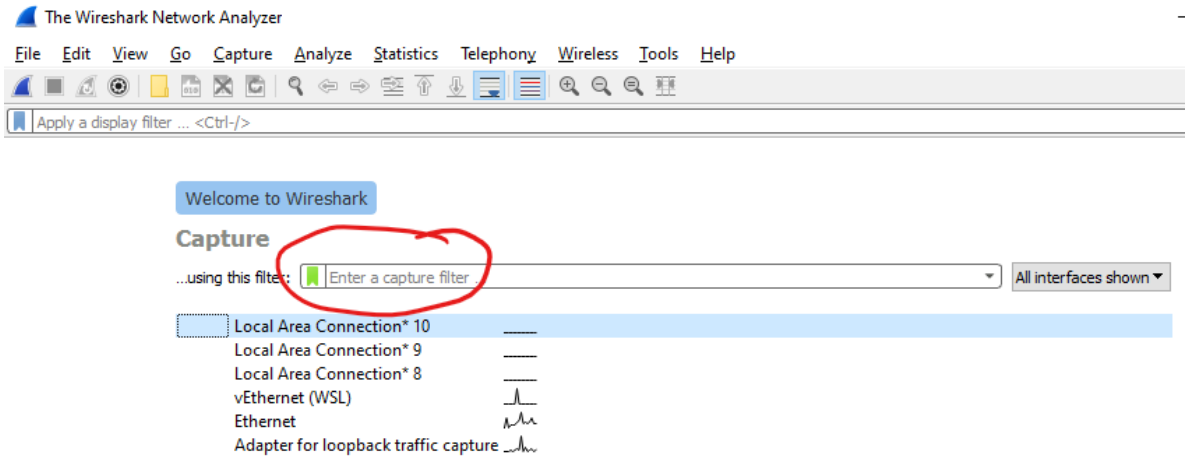

Figure 2: Add filters in Wireshark.

[3] `https://www.wireshark.org/download/docs/user-guide.pdf`
[4] See `https://wiki.wireshark.org/CaptureFilters`

Task 4: Simulating the Network for Adaptive Streaming (0.5 mark)

An important aspect of DASH is its adaptive streaming feature and now let's see it in action. For this task, refresh the web page of our DASH player and **"Load" the default MPD playlist instead** (as it contains more quality levels for better visualization), which should look like -

`https://dash.akamaized.net/akamai/bbb_30fps/bbb_30fps.mpd`

Next, we simulate a drop in network speed by using the browser's network throttling feature. Open the browser's network inspection tool again and select a slower network speed, e.g., "Fast 3G" instead of "Online" in Google Chrome[5]. The player should adapt and download future video segments of lower quality levels.

To observe this behaviour, we can use the analysis graph provided by the DASH player (at the bottom of the web page). It dynamically updates with the "Video Bitrate" of the segment being downloaded (as well as the current "Video Buffer Level"). Take a screenshot of the graph showing the adaptive streaming capability, i.e., the bitrate graph should drop/increase over time (along with the buffer level). A screenshot sample is given as below, which may not be the correct answer.

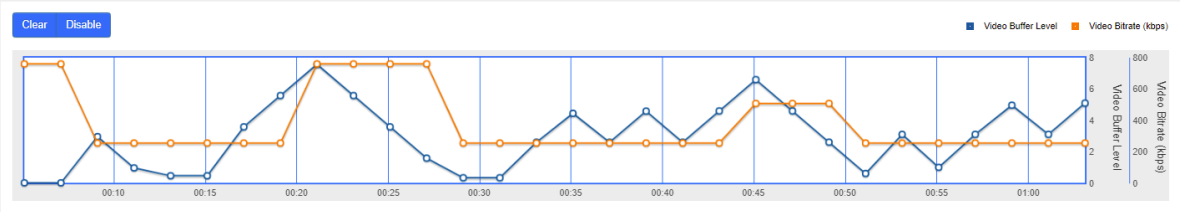

Figure 3: A sample of the graph for illustrating the adaptive streaming capability.

[Submission]

Save your screenshot as `e1_t4.png`.

[5]See instructions in `https://developers.google.com/web/tools/chrome-devtools/device-mode`

Exercise 2 - Basic Media Processing (1 Mark)

In this exercise, we will learn how to use **FFmpeg** for basic media processing. FFmpeg is a collection of libraries and tools to process multimedia content such as audio, video, subtitles and related metadata. You need to first download and install FFmpeg on your local machine. The following links may help:

`https://ffmpeg.org/download.html`

To validate that the package is installed correctly, use the `ffmpeg -version` command, which prints the FFmpeg version.

In this exercise, we will use the MP4 file titled **test.mp4** available at -

LumiNUS → Files → Assignment Questions → `A3-video.zip`

Please download test.mp4 to your local device and look into the two following tasks.

Task 1: Checking the metadata of MP4 file (0.5 mark)

Similar to photos, videos contain **metadata** information about the location where the video was shot. Likewise, container formats like AVI and MP4 contain meta information about codecs, video and audio streams and more. A metadata viewer reveals information of video files you may not be aware of.

To check the metadata of the given MP4 file, run **ffmpeg** command as follows:

```
ffmpeg -i test.mp4
```

View the output content and answer the following five questions -

1. What is the resolution of the video?
2. What is the frame rate of the video?
3. Which video compression standard is used in this video?
4. Does this file employ RGB for color encoding?
5. What is the sampling rate of the audio?

[Submission]

Enter your answers clearly in LumiNUS Quiz - Assignment 3.

Task 2: Processing MP4 file (0.5 mark)

FFmpeg enables us to process the video file with very simple commands. For example, we can type `ffmpeg -i test.mp4 test.flv` for converting video files to FLV format. To set the aspect ratio (e.g., 16:9) to video, enter the command as `ffmpeg -i test.mp4 -aspect 16:9 output.mp4`, where output.mp4 is the output file. For more operations such as video cropping, please refer to the official document[6]. In this task, you are asked to use ffmpeg to implement the following operations **within one command**:

1. Take test.mp4 as input.
2. Change the resolution to 640×480.
3. Extract the video from 00:00:17 to 00:00:27.
4. Remove the audio stream.
5. Convert the format of above result to .avi and save it as `e2_t2_output.avi`.

To evaluate the result, you can either open the AVI file with your favorite video player or use FFmpeg to print its metadata for comparison. For example, the metadata of the new file could be:

```
ffmpeg version 2.8.17-0ubuntu0.1 Copyright (c) 2000-2020 the FFmpeg developers
  built with gcc 5.4.0 (Ubuntu 5.4.0-6ubuntu1~16.04.12) 20160609
  configuration: --prefix=/usr --extra-version=0ubuntu0.1 --build-suffix=-ffmpeg --toolchain=hardened --libdir=/
  usr/lib/x86_64-linux-gnu --incdir=/usr/include/x86_64-linux-gnu --cc=cc --cxx=g++ --enable-gpl --enable-shared
  --disable-stripping --disable-decoder=libopenjpeg --disable-decoder=libschroedinger --enable-avresample
  --enable-avisynth --enable-gnutls --enable-ladspa --enable-libass --enable-libbluray --enable-libbs2b
  --enable-libcaca --enable-libcdio --enable-libflite --enable-libfontconfig --enable-libfreetype
  --enable-libfribidi --enable-libgme --enable-libgsm --enable-libmodplug --enable-libmp3lame
  --enable-libopenjpeg --enable-libopus --enable-libpulse --enable-librtmp --enable-libschroedinger
  --enable-libshine --enable-libsnappy --enable-libsoxr --enable-libspeex --enable-libssh --enable-libtheora
  --enable-libtwolame --enable-libvorbis --enable-libvpx --enable-libwavpack --enable-libwebp --enable-libx265
  --enable-libxvid --enable-libzvbi --enable-openal --enable-opengl --enable-x11grab --enable-libdc1394
  --enable-libiec61883 --enable-libzmq --enable-frei0r --enable-libx264 --enable-libopencv
  libavutil      54. 31.100 / 54. 31.100
  libavcodec     56. 60.100 / 56. 60.100
  libavformat    56. 40.101 / 56. 40.101
  libavdevice    56.  4.100 / 56.  4.100
  libavfilter     5. 40.101 /  5. 40.101
  libavresample   2.  1.  0 /  2.  1.  0
  libswscale      3.  1.101 /  3.  1.101
  libswresample   1.  2.101 /  1.  2.101
  libpostproc    53.  3.100 / 53.  3.100
Input #0, avi, from 'output.avi':
  Metadata:
    encoder         : Lavf56.40.101
  Duration: 00:00:10.01, start: 0.000000, bitrate: 524 kb/s
    Stream #0:0: Video: mpeg4 (Simple Profile) (FMP4 / 0x34504D46), yuv420p, 640x480 [SAR 1:1 DAR 4:3], 515 kb/s,
    29.97 fps, 29.97 tbr, 29.97 tbn, 30k tbc
```

Figure 4: The metadata of the processed file.

[Submission]

- Record your command in `e2_t2.txt`
- save the output file as `e2_t2_output.avi`.

[6] https://www.ffmpeg.org/ffmpeg.html

Note: You will need to complete both Exercise 1 and 2 before working on Exercise 3 as they are related.

Exercise 3 - Real-Time Conversational Applications (1 Mark)

In the lecture, we also learnt about RTP (Real-Time Protocol) and WebRTC (Web browsers with Real-Time Communications), two other popular multimedia streaming standards. There are two tasks in this exercise to help us better understand these protocols.

Task 1: Streaming MP4 to YouTube Live (0.5 mark)

Live streaming is becoming popular means of entertainment. In Exercise 2, we have shown the power of FFmpeg in media processing. This task will illustrate how one can **stream** a video file, screencast, or web camera output to a popular live streaming platform (i.e., YouTube Live) using FFmpeg.

First of all, you need to register a YouTube account and apply for a key for YouTube Live[7]. Please do take note that the Live account will be reviewed **in 24 hours**. We advise you to complete it ahead of time. After that, we can access the interface of the studio, as shown in the following figure. The following link illustrates how to stream IP Camera to YouTube Live:

`https://www.youtube.com/watch?v=MM2oTTb5zXg&t=353s&ab_channel=RickMakes`

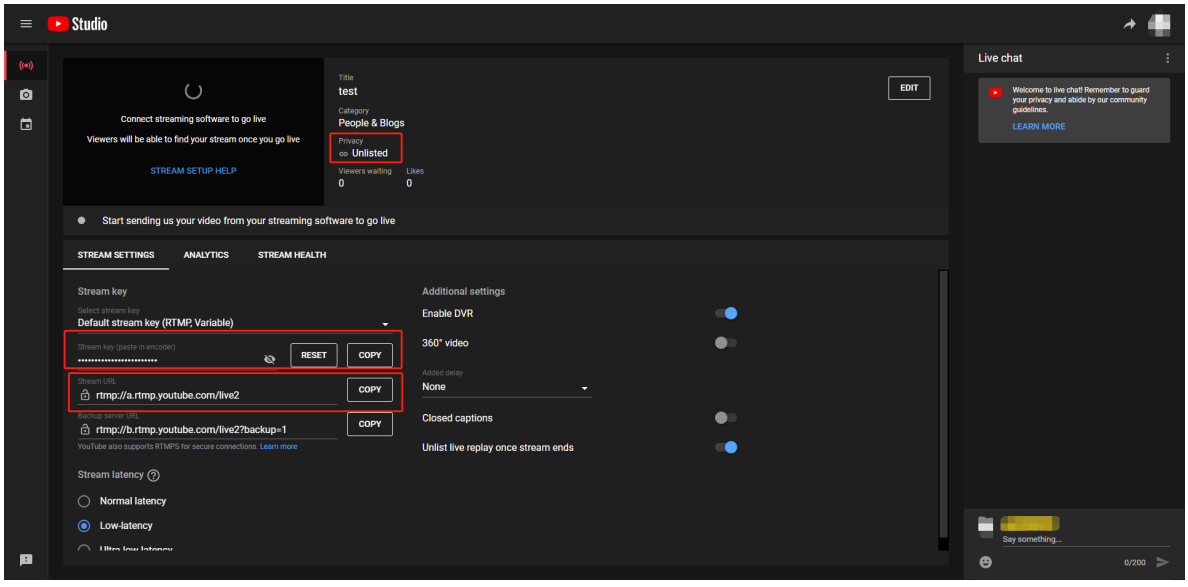

Figure 5: Interface of YouTube Live.

[7] `https://support.google.com/youtube/answer/2474026?hl=en&ref_topic=9257984`

In this task, you are required to follow the above instruction to stream your camera[8] and answer the following questions.

During the live streaming,

1. Which application layer protocol is used for upstreaming?
2. Which protocol is used in the transport layer? Why?
3. Record your command for pushing streams.
4. Capture a **screenshot** of your live room clearly showing:
    - the video content
    - sending a message with your matric number in the chat room.

    An example is given below.

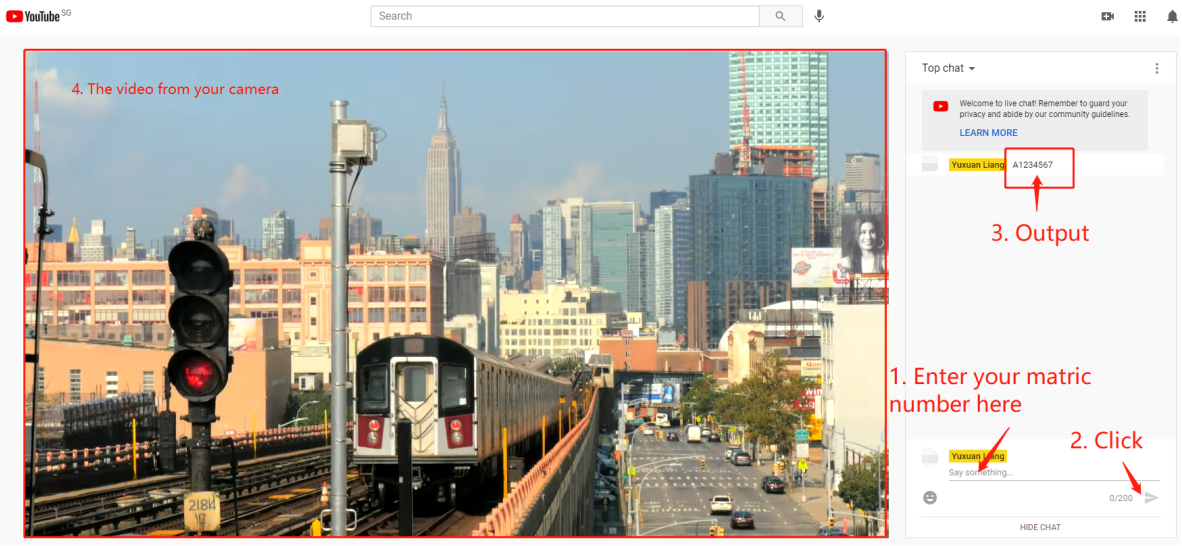

Figure 6: An example of submission.

[Submission]

- Enter your answers in LumiNUS Quiz - Assignment 3.
- Save the screenshot as `e3_t1.png`.

[8] If you are testing your own file, please take note that you have to follow the copyright policy.

Task 2: Inspecting a Simple WebRTC Application (0.5 mark)

To understand how WebRTC works in real-life applications, let's run a simple experiment (similar to Exercise 1). For our WebRTC client in this exercise, we will use a browser-based video chat application provided by the WebRTC project authors at -

`https://appr.tc/`

Please access the application using any modern browser (we recommend Google Chrome) and create a chat room for your use. As this is a two-way peer-to-peer communication application, we need another device to join the same chat room created. We recommend using your mobile phone or working with another student to get the video chat running.

Once the video chat is running, open and view the browser's network log and Wireshark's capture log. Note that we can only view very limited information about the WebRTC packets in Wireshark because they run on additional security protocols that encrypt most of the stream information. (We prefer to keep it secure in this exercise as it streams from your camera feed.) Answer the following two discussion questions -

1. You should notice that the browser's network log here behaves differently from DASH's case in Exercise 1. Specifically, the browser's network log for WebRTC does not update with the video data retrieved, as seen in DASH. Why do you think this is so?
2. Which transport protocol(s) do WebRTC and DASH generally use **for their video streams**? Give two reasons why they use the same/different transport protocol.

[Submission]

Record your answers in LumiNUS Quiz - Assignment 3.

ENJOY