National University of Singapore

School of Computing

CS2105 **Assignment 3** Semester 1 AY20/21

Submission Deadline

11th Nov 2020 (Wednesday) 11:59pm sharp. 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 **18th Nov 2020 (Wednesday) 11:59pm** sharp and no late submission will be accepted afterwards.

Objectives

In this assignment, you will inspect two multimedia streaming applications that run on DASH and WebRTC and discuss some important aspects that characterise these two popular streaming standards. After completing this assignment, you should (1) have good understanding of how DASH and WebRTC work in real-life applications, and (2) know how to use common network inspection tools for simple network debugging tasks.

This assignment is worth **5 marks** in total and they are split across two 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 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 and the troubleshooting skills you learn will be very useful.

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

Question & Answer

If you have any doubts on this assignment, please post your questions on LumiNUS forum or consult the teaching team. However, the teaching team will NOT debug issues for students. The intention of Q&A is to help clarify misconceptions or give you necessary directions.

Setup

You should use your local machine for all tasks in this assignment. Note that you should **not** need to `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

Submission

Please prepare a zip file that contains the submission files (details below) and submit it to the `Assignment_3_student_submission` folder of LumiNUS Files. Please name your zip file as **<Student_Number>.zip**, where **<Student_Number>** refers to your matric number that starts with 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 five submission files below -

<Student_Number>.zip contains:		
1	`discussion-<Student_Num>.txt`	A single plain text file to record your answers to the discussion questions in Exercise 1 (Task 1 and 2) and Exercise 2 (Task 2).
2	`dash-wireshark-<Student_Num>.pcapng`	See Exercise 1 (Task 3).
3	`dash-browser-<Student_Num>.png|jpg`	See Exercise 1 (Task 4).
4	`webrtc-browser-<Student_Num>.png|jpg`	See Exercise 2 (Task 1).
5	`webrtc-wireshark-<Student_Num>.png|jpg`	See Exercise 2 (Task 1).

Exercise 1 - DASH

In lecture, we learnt about DASH (Dynamic Adaptive Streaming over HTTP), which is 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 -

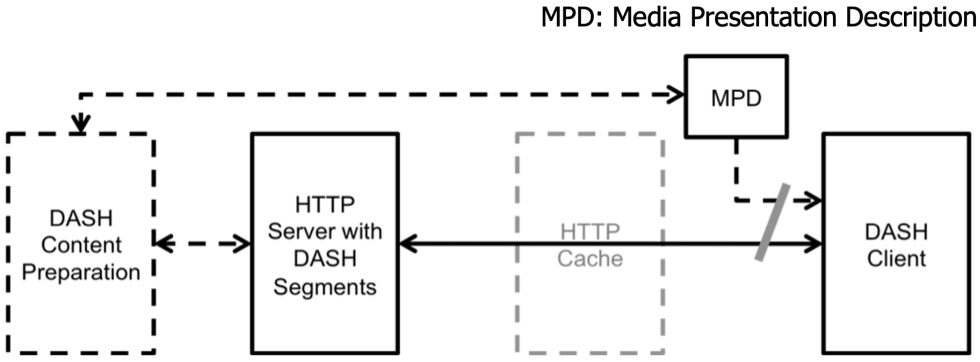

Figure 1: A streaming application built on DASH

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 are provided by this playlist?
2. What are the highest and lowest quality levels?
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]

Record your answers clearly in `discussion-<Student_Number>.txt`.

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 -

`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, begin streaming the video provided by our playlist in Task 1 by replacing the MPD URL in the DASH player and click "Load". Now let's inspect the video segments retrieved by the player. On your browser, open up the network inspection tool [1] 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]

Record your answers clearly in `discussion-<Student_Number>.txt`.

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 Wireshark.

[1] 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.

Open Wireshark on the machine running the browser and begin capture. Following the same steps as in 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.), which may make network inspection difficult.

Now, let's restart the capture process, but this time using Wireshark's capture filters[2] **to capture only the relevant packets** involved in this particular request-response exchange for the MPD file download. Save this Wireshark capture log in the default `pcapng` file format.

[Submission]

Save your file as `dash-wireshark-<Student_Number>.pcapng`.

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

An important aspect in 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. 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).

[Submission]

Save your screenshot as `dash-browser-<Student_Number>.png|jpg`.

[2] See https://wiki.wireshark.org/CaptureFilters

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

Exercise 2 - WebRTC

In lecture, we also learnt about WebRTC (Web browsers with Real-Time Communications), another popular multimedia streaming standard. There are two tasks in this exercise.

Task 1: 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 own 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 work 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 (similar to Exercise 1). 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.)

Take two screenshots - (i) the browser's WebRTC client and network log, and (ii) Wireshark's capture log to show completion of the task. The screenshots do not have to contain any particular information as long as we can reasonably deduce they belong to the WebRTC stream.

[Submission]

Save the two screenshots as (i) `webrtc-browser-<Student_Number>.png|jpg` and (ii) `webrtc-wireshark-<Student_Number>.png|jpg`.

Task 2: Discussion on WebRTC and DASH (1.5 marks)

After completing Task 1, answer the following three 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.
3. Discuss two (other) differences between WebRTC and DASH (e.g., from architecture perspective).

[Submission]

Record your answers clearly in `discussion-<Student_Number>.txt`.

THE END