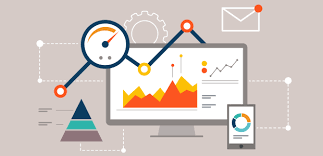
**CS 4713**

**INTRODUCTION TO THE INTERNET: ARCHITECTURE AND PROTOCOLS**

**ASSIGNMENT 2b:**

**ANALYZING VIDEO STREAMING OVER DIFFERENT NETWORKS**

**Deadline: Friday, 16th November 2018 at 11:50pm**

**Course Policy about Plagiarism**

* This assignment is suppose to be done individually.
* Students must not share any scripts or their analysis report with other students.
* Students must be prepared to explain any program code or report they submit.
* Students must indicate with their submission any assistance received.
* All submissions are subject to plagiarism detection.
* Students are strongly advised that any act of plagiarism will be reported to the Disciplinary Committee

**INTRODUCTION**

Many recent studies have highlighted the critical role that user-perceived quality-of-experience (QoE) plays in Internet video applications, as it ultimately affects revenue streams for content providers. Specifically, metrics such as the duration of rebuffering (i.e., the player’s playout buffer does not have content to render), startup delay (i.e., the lag between the user clicking vs. the time to begin rendering), the average playback bitrate, and the variability of the bitrate delivered have emerged as key factors. Consequently, a robust algorithm in client-side players which can adapt video bit rate is critical to ensure good user experience.

**MAHIMAHI**

Mahimahi is a framework to record and replay traffic from HTTP-based applications over emulated network conditions. It is structured as a set of four UNIX shells, namely, the record, replay, delay and link shells. The framework isolates its traffic from the rest of the host system, allowing multiple instances of its shells to run in parallel without interference.

The four shells are as follows:

1. Record Shell: Records HTTP traffic and stores it for subsequent replays
2. Replay Shell: Replays previously recorded HTTP content (possibly over different emulated network conditions)
3. Delay Shell: Imposes a user specified delay on all packets that originate from it.
4. Link Shell: Emulates a network link by delivering packets according to a particular trace.

You will use mahimahi to emulate different network conditions and observe changes in QoE metrics. For the purpose of this assignment, you will require only the last two shells but feel free to play around with them all. For more information, you can take a look at [this](http://mahimahi.mit.edu/mahimahi_atc.pdf) paper.

**SETUP**

**Installing Dependencies:**

* You will require Ubuntu version 14.04 or later.
* Run the following commands

sudo apt-get install python-setuptools

sudo apt-get install xvfb

* Run the setup file provided to you:

python setup.py

**Setting up Mahimahi for runs:**

* You are provided with network traces over which you will conduct your runs. Change the directory to the one containing the traces
* For the link shell:

mm-link [uplink trace file name] [downlink trace file name] (use the flag --meter-all flag to show graphs of throughput)

* For the delay shell:

mm-delay [delay value]

* You will now be inside a link/delay shell. To launch the browser from within it, do

google-chrome

* For examples of shell setups, visit [this](http://mahimahi.mit.edu/) link.

**GATHERING DATA**

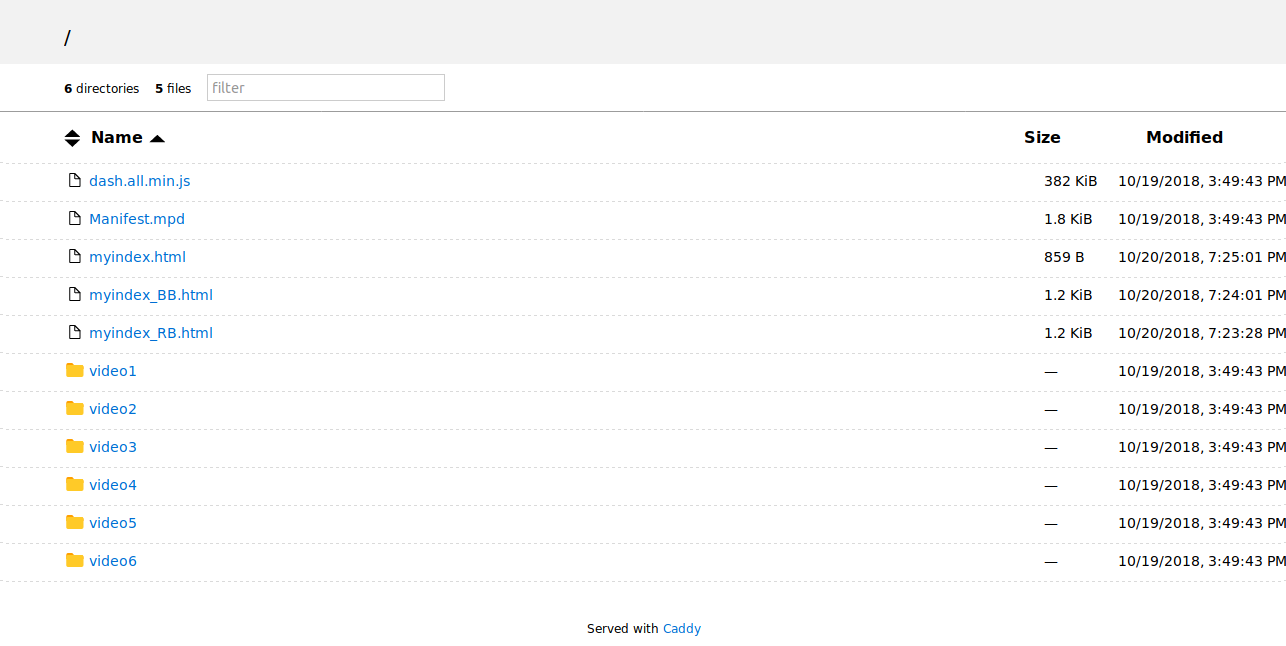
You will be analyzing the results generated in the log files by running two different client side video streaming algorithms.

**To run the server hosting your video files:**

* Open caddy folder, and change the IP in the Caddyfile to your machine’s ip (do not write localhost).
* Type the following command in the terminal (not in the mahimahi shell)

./caddy

* Open the link in the browser to check if the server is running. The page should be similar to the one in Figure 1



**Figure 1**

**To run the video client:**

* Run the mahimahi shell with the traces
* Open the real\_exp folder. Change the url to your server’s ip in run\_video.py file. Run the file run\_video.py in that shell. Sample run command is provided in the comments.
* The video will automatically open in the browser make sure you press the play button immediately.
* The log files containing the data will be saved in the results folder.
* The format of the log files is [timestamp] [bitrate] [buffer size] [rebuffering time] [size] [fetch time]

**Analyzing Video performance:**

* To evaluate QoE metrics you can use evaluageLog.py file in the results folder. Sample run command is provided in the comments.
* You will have to write your own scripts for any other analysis or plotting graphs.
* You can also use Chrome DevTools and server’s access log that is generated for analysis.

**ANALYSIS TASKS**

Design and implement experimental runs that will allow you to comment on the following aspects of video streaming process. Please note that grading in this assignment is dependent on how thoroughly you conduct runs and, in light of their results, how confidently you can draw conclusions.

1. **Rebuffering Time**

Compare the rebuffering time of the two algorithms on different traces?

1. **Bitrate Variation**

Compare the amount of bitrate variation of the two algorithms on different traces?

1. **Quality of Experience**

Compare the QoE metric of the two algorithms on different traces?

1. **QoE and Network Conditions**

Which trace file gives the best performance for each algorithm. Comment on why is this the case?

1. **Video Request**

Is the video downloaded as a single file or is it divided. Comment on why is this the case?

1. **Role of Buffer Size**

How does the amount of data in buffer affect the subsequent bitrates. Which algorithm takes buffer size into account when requesting the video?

1. **Possible Improvements in QoE function**

Which factors should be added or removed from the QoE function? How should the weights be changed? How are these changes going to lead to higher user engagement?

1. **Reverse Engineering Algorithms**

Which factor is taken into account by each algorithm before requesting the video in each iteration. Why is this factor important when requesting a video? How does the algorithm select a particular bitrate? When does the rebuffering take place?

1. **Possible Improvements in Algorithms**

Which factors should the algorithms incorporate in making bitrate decisions for better performance? Apart from network bottlenecks, how can the algorithms incorporate bottlenecks in low end client devices? Is it possible to incorporate machine learning. If yes, propose a machine learning model which will enhance performance?

The assignment is pretty open-ended and grading will depend upon the thoroughness of design and analysis of runs.

**SUBMISSION**

You are required to submit

1. All the scripts that you used to analyze the log files.
2. All log files
3. A document explaining the design, analysis and all relevant graph plots.