Adaptive Bitrate Streaming

SIIN903 - Multimedia Networking
UBINET Master 2024-25
Université de Côte d'Azur
Instructor : Ramón APARICIO (ramon.aparicio-pardo@univ-cotedazur.fr)

Based on the lab `https://witestlab.poly.edu/blog/adaptive-video-reproducing/`

10 January 2025

1 Objective

This group project aims to understand how adaptive bitrate algorithms trades-off between video quality and rebuffering time To do that, a DASH server and a DASH client are provided.

2 Rules

You can work in groups of 1 or 2 students.

You have to prepare a zip folder containing : (1) a 5-page report analysing the behaviour of the provided adaptive bitrate algorithms for the two provided scenarios, and ; (2) if you do the extra work, the Python source code of your proposal with the analysis of its behaviour for the same three benchmarks.. Then, you will have to upload it to the drop box "Report and Source Code - Lab 2 : DASH" in the `lms` course site before February 3th at 23 :45.

3 Before the lab

In this lab, we will work with `Docker`, an open-source platform that automates the deployment, scaling, and management of applications using containerization technology. You can install it in any Operating System with *Docker Desktop* from `https://docs.docker.com/get-started/get-docker/`. If it's the first time that you work with `Docker`, have a look to this tutorial : `https://www.docker.com/101-tutorial/`.

We will use `Docker` to deploy a simple network composed by a DASH server and a DASH client placed in the same IP subnetwork. To do that we will use `docker-compose` : a tool that allows you to define and manage multi-container Docker applications using a single YAML file [1]. You have some tutorials here : `https://docs.docker.com/guides/docker-compose/` and `https://docs.docker.com/get-started/docker-concepts/the-basics/what-is-docker-compose/`.

Finally, you have a good compilation of the basics of `Docker` and `docker-compose` here : `https://github.com/seed-labs/seed-labs/blob/master/manuals/docker/SEEDManual-Container.md`

1. `https://docs.docker.com/compose/`

In the `lms` course site, you have a folder called `src` with all the files required to create the simple network with the server and the client.

4 Setting up the network

In the `src` folder, you have four elements :

1. the `server` folder, containing the `dockerfile` to build the dash server from the official Apache2 webserver container, and the video frames in the compressed file `media.tgz`. You need to download the video frames directly from the url below and later put them inside this folder :

   `https://nyu.box.com/shared/static/d6btpwf5lqmkqh53b52ynhmfthh2qtby.tgz`

2. the `client` folder, containing the `dockerfile` to build the dash client from the official linux container, and a `python` based emulated DASH video player in the compressed file `AS-stream.zip`. The video player corresponds to the github project `https://github.com/teaching-on-testbeds/AStream/tree/master`.
3. the `docker-compose.yml` file, defining the network to set up.
4. the `volumes` folder, that will be mounted in both containers (client and server) to become a shared folder between the containers and your operating system hosting them.

To deploy the network, we must simply run in your terminal the next two commands.

```
$ docker-compose build  # Build the container images
$ docker-compose up     # Start the containers and the network
```

When we finish our work, we must turn down properly the containers with the command below :

```
$ docker-compose down # Shut down the containers
```

In the future, when we will come back to the laboratory activity, we will need only to start the containers with the command `docker-compose up` since their images are already built.

Once the network deployed, open a second terminal. We are going to examine the deployed network. Type the command `docker network ls`. Among the available networks, you will see a `bridge` network called `net-10.9.0.0`. Now, type the command `docker network inspect net-10.9.0.0`. This is our network : a network with address `10.9.0.0/24` composed by a virtual L2 switch connecting three network interfaces :

1. `10.9.0.1` : the client container,
2. `10.9.0.80` : the web server container, and
3. `10.9.0.254` : the default gateway to exit the network. (This is actually the host operating system, that act as NAT router).

Now, we are going to enter in the server container. To do that, type the next command in a terminal :

```
$ docker exec -it dash_server /bin/bash
```

Once inside check with `ip a` that the server uses the IP address `10.9.0.80`. You can also `ping 10.9.0.1` to verify that you can communicate with the client. To exit the container and come back to the host OS terminal, simply type `exit`.

Since the two containers are not connected by a real network but by a docker environment, we need to constraint the output bandwidth from the server. Otherwise, we will have non-realistic values for the downlink bandwidth. For this, a script called `constant_rate.sh` at the folder `/volumes` is available. This script runs the Linux tool `tc qdisc`. This tool manipulates the traffic control settings of a network interface [2]. In particular, this script will fix the downlink bandwidth from the server to 1 Mbps by running it (`./constant_rate.sh`). To check that output bandwidth has been modified, type `tc qdisc show dev eth0`. Eventually, similar scripts will allow us to simulate bandwidth variations to test our adaptive video algorithms.

5 Streaming an adaptive video

At this moment, we can play from the client, using the DASH protocol, the video hosted on the server. To play the video, we are going to use a client application written in `Python` called AStream (`https://github.com/teaching-on-testbeds/AStream/`). This `Python` program works (emulates) as a DASH player that generates the HTTP requests required to play the video and decides which quality file to download depending on an adaptation algorithm.

By-default, the client uses the adaptation algorithm called `basic`. This basic adaptation is a *rate-based policy* that chooses a video rate based on the observed download rate. It keeps track of the average download time for recent segments, and calculates a running average. If the download rate exceeds the current video rate by some threshold, it may increase the video rate. If the download rate is lower than the current video rate, it decreases the video rate to ensure smooth playback.You can see the source code for the basic policy here : `https://github.com/teaching-on-testbeds/AStream/blob/master/dist/client/adaptation/basic_dash2.py`

Now, you are going to make a first video play using this `basic` rate-based algorithm. As we did for the server in the previous section, you can enter in the client container by typing :

```
$ docker exec -it dash_client /bin/bash
```

You will land at the folder `/home`. From this folder, you type in the terminal :

```
 python3 AStream-master/dist/client/dash_client.py -m
 http://10.9.0.80/media/BigBuckBunny/4sec/BigBuckBunny_4s.mpd -p 'basic' -d
```

In the command prompt, you will see to appear the information about the playing. The temporal evolution of the playing will be saved as log files in the client directory `/home/ASTREAM_LOGS`. Play the video at least for several minutes. Once finished the play-out, you can copy this folder into the working directory of your host OS using the command :

```
docker cp <container_id>:/home/ASTREAM_LOGS/ .
```

To help with data visualization, you can use this Python notebook : `https://colab.research.google.com/drive/1pNPWfFJPajYfFHmodl0eT4gE81YEQplB`. You will need a Google account, since the Python notebook is hosted by Google Colab, and you should do a local copy inyo your Google drive. You can find also the Python notebook in the `lms`, But, you will need to install `jupyter notebook` from `https://jupyter.org/`. Follow the instructions to upload your log file, change the filename variable, and plot your results.

You can also *simulate* the video play by recreating the video from the individual segments downloaded by the DASH client. These video segments will have been downloaded into a directory with the prefix `TEMP` in the "client" home directory. You will use the `ffmpeg` tool to recreate the video. Enter into the directory with `cd` and type the next commands :

```
cat BigBuckBunny_4s_init.mp4 $(ls -vx BigBuckBunny_*.m4s) > BigBuckBunny_tmp.mp4

ffmpeg -i  BigBuckBunny_tmp.mp4 -c copy BigBuckBunny.mp4
```

2. see more information here : `https://linux.die.net/man/8/tc`

Once you have generated the file BigBuckBunny.mp4, you can copy it into the working directory of your host OS using the command :

```
docker cp <container_id>:/home/TEMP_ <12345>/BigBuckBunny.mp4 .
```

Then, just play it.

6 Your work

The objective of this project is that you study the fundamental tradeoff between video quality (bitrate) and rebuffering. To do that, you will use a rate-based and a buffer-based algorithm in two different scenarios.

6.1 Scenario 1 : Constante rate interrupted

In this scenario, you will go to the server container, and you will activate the shell script `constant_rate_interrupted.sh` to manipulate the server network interface. This scripts sets up again the downlink bandwidth from the server to 1 Mbps for 70 seconds, reduce dramatically the bandwidth for 60 seconds to 100 kbps, and restore the original bandwidth to 1 Mbps. Once the script activated, play the emulated video player using the rate-based algorithm `basic`. **Play the whole video, retrieve the logs, plot the video rate and buffer evolution, and discuss the behaviour of the algorithm.** You have access to its code in `https://github.com/teaching-on-testbeds/AStream/blob/master/dist/client/adaptation/basic_dash2.py`.

Now, repeat the procedure but using the buffer-based algorithm `netflix` whose code is here `https://github.com/teaching-on-testbeds/AStream/blob/master/dist/client/adaptation/netflix_dash.py`. This code implements the algorithm proposed in this paper :

Te-Yuan Huang, Ramesh Johari, and Nick McKeown. 2013. Downton abbey without the hiccups : buffer-based rate adaptation for HTTP video streaming. In Proceedings of the 2013 ACM SIGCOMM workshop on Future human-centric multimedia networking (FhMN '13). Association for Computing Machinery, New York, NY, USA, 9 ?14 `https://doi.org/10.1145/2491172.2491179`

6.2 Scenario 2 : Variable rate

Now, repeat the previous tests, but running the shell script `variable_rate.sh`. This script initially sets up the bandwidth to 1 Mbps for 110 seconds, changes the bandwidth to 350 kbps for 75 seconds, and, finally, increases the bandwidth to 2 Mbps for 125 seconds.

6.3 Extra work

Once you have done the previous part, you may implement an algorithm able to beat the previous ones. To beat means to obtain (1) higher *average video rate*, and (2) lower *total paused time* (startup time plus rebuffering time) for the video play with the bandwidth evolution instrumented by `variable_rate.sh`. The better your algorithm is with respect to the ones from your colleagues, the more marks you will obtain. You are not required to make up your own algorithm from the scratch. You can re-implement any algorithm that you find from the bibliography, but you need to understand it and explain in the report why it is better than the previous algorithms using the plots of video rate, and the buffer status.