DASH Streaming

Abstract

The objective of this project is to build a simple DASH video streaming system using open source tools. In particular, you will prepare DASH content at the server side using `ffmpeg` and `Bento4` tools. Then, you will develop buffer-based rate adaptation algorithm at the client side in `dash.js` library. To assess your algorithm, you are asked to evaluate your algorithm against the `dash.js` adaptation algorithm in different network profiles. You report three quality metrics: rendering bitrate, buffer level and buffering rate.

After completing this project, the student is expected to learn (1) how DASH video streaming systems work, (2) the need of rate adaptation algorithms in such systems, and (3) the important metrics that affect user engagement.

Environments and Tools

- A recent Linux distribution (Ubuntu >= 14.04)
- *Video encoding library:* `ffmpeg` release 3.1.2
- *Video segmentation and MPD generation:* `bento4`
- *DASH client:* `dash.js` https://github.com/Dash-Industry-Forum/dash.js/commit/e66681012ca7a561d94e274f9aeeea250393e558
- *HTTP Server:* `nginx` >= 1.4
- *Network emulator:* Linux `tc` and `NetEm`:
- *Suggested plotting libraries:*
  - `python` and `matplotlib`
  - `gnuplot`
  - `matlab`

Steps

1. **Setup your physical or virtual machine**
    a. A recent Linux distribution (Ubuntu >= 14.04), with `bash` shell.
    b. Install `git` client

```
$ apt-get install git
```

    c. Install `ffmpeg` (binaries in class directory)
    d. Install `Bento4` tool (binaries in class directory)
    e. Install `nginx` HTTP server

```
$ apt-get install nginx
```

2. **Prepare DASH content**
    a. Download the video file (~11 GB):
        i. URL: https://media.xiph.org/video/derf/y4m/big_buck_bunny_1080p24.y4m.xz
    b. Please read the video copyright: https://peach.blender.org/about/
    c. Convert the video from y4m to mp4

```
$ /path/to/ffmpeg -i input.y4m -c:v libx264 -preset slow
-crf 1 -pix_fmt yuv420p -movflags +faststart input.mp4

$ /path/to/ffmpeg -i input.mp4 -c:v libx264 -b:v 5000k
-minrate 5000k -maxrate 5000k -bufsize 1835k
input-cbr-07.mp4
```

    Check "`ffmpeg -h`" to show help message

d. Encode input-cbr-07.mp4 file to six bitrates

```
$ /path/to/ffmpeg -i input-cbr-07.mp4 -c:v libx264 -b:v
4000k -minrate 4000k -maxrate 4000k -bufsize 1835k
input-cbr-06.mp4

$ /path/to/ffmpeg -i input-cbr-07.mp4 -c:v libx264 -b:v
3000k -minrate 3000k -maxrate 3000k -bufsize 1835k
input-cbr-05.mp4

$ /path/to/ffmpeg -i input-cbr-07.mp4 -c:v libx264 -b:v
2000k -minrate 2000k -maxrate 2000k -bufsize 1835k
input-cbr-04.mp4

$ /path/to/ffmpeg -i input-cbr-07.mp4 -c:v libx264 -b:v
1500k -minrate 1500k -maxrate 1500k -bufsize 1835k
input-cbr-03.mp4

$ /path/to/ffmpeg -i input-cbr-07.mp4 -c:v libx264 -b:v
1000k -minrate 1000k -maxrate 1000k -bufsize 800k
input-cbr-02.mp4

$ /path/to/ffmpeg -i input-cbr-07.mp4 -c:v libx264 -b:v 500k
-minrate 500k -maxrate 500k -bufsize 300k input-cbr-01.mp4
```

e. Fragment each video input-cbr-X.mp4 to 2 seconds:

```
$ /path/to/bento4-sdk/bin/mp4fragment --fragment-duration
2000 input-cbr-X.mp4 input-cbr-X-frag.mp4
```

f. Create segments and MPD for the videos:

```
$ /path/to/bento4-sdk/bin/mp4dash -o video1 -f
--exec-dir=/path/to/bento4-sdk/bin --profiles=on-demand
input-cbr-01-frag.mp4 input-cbr-02-frag.mp4
input-cbr-03-frag.mp4 input-cbr-04-frag.mp4
input-cbr-05-frag.mp4 input-cbr-06-frag.mp4
```

Note that the directory that contains the segments and MPD is named video1

3. Setup HTTP server

a. Create a new directory at root directory and call it: streaming
b. Add a new directory under /streaming and call it: videos
c. Copy directory video1 to the directory /streaming/videos/
d. Edit /etc/nginx/nginx.conf as the following:
    i. Check nginx.conf in the class directory
e. Reload `nginx` configuration:

```
$ sudo nginx -s reload
```

f. Make sure you can access the MPD file from the web browser: http://machine-ip:8080/videos/video1/stream.mpd
g. Check "`nginx -h`" to show help message

4. Install dash.js library

a. Download the "dash.js-development.zip" file from class directory
b. Make sure to read the library license agreement
c. Since you will modify dash.js
    i. Install `node-v4.4.7` (in the class directory)
    ii. Install `grunt-js` (using npm command)

```
$ /path/to/node/bin/npm install -g grunt-cli
```

    iii. If you are using local NodeJS, make sure to change the path of your node installation in `/path/to/node/bin/grunt`
    iv. Move dash.js-development directory to /streaming/dash.js directory

v. Install dash.js:

```
$ cd /streaming/dash.js
$ /path/to/node/bin/npm install
```

vi. Run grunt default task:

```
$ /path/to/node/bin/grunt
```

5. Setup dash.js Client

a. Copy the prepared client files in the class directory to /streaming directory. This will create dash.js client at your machine to be accessible by web browsers
b. Make sure you can access the client here: http://machine-ip:8080/client/index.html

6. Implement buffer-based rate adaptation algorithm

a. Adaptation algorithms in `dash.js` are implemented as rules in the library. Each algorithm may consist of multiple rules, each runs at specific conditions. To facilitate the implementation, you are asked to implement only one rule (called `PureBufferOccupancyRule`). We modified the default `dash.js` client so that your rule/algorithm runs as we expect.
b. Create a new file called "`PureBufferOccupancyRule.js`" at src/streaming/rules/abr/
c. To create a new algorithm, you need to:
    i. Create a function called "`PureBufferOccupancyRule(config)`"
    ii. The function "`PureBufferOccupancyRule`" should return instance object with two functions: `setup` and `execute`
    iii. "`execute`" function has the following definition:
        1. "`execute (rulesContext, callback)`", which will be executed to run your algorithm
    iv. "`setup`" function should initialize your algorithm; its definition is:
        1. `setup()`
    v. Check the file "`src/streaming/rules/abr/ThroughputRule.js`" to learn how throughput-based algorithm is implemented
    vi. Open file "`src/streaming/rules/abr/ABRRulesCollection.js`":
        1. Remove lines 62-73
        2. Add code to use your buffer based algorithm
        3. Follow same convention as throughput-based algorithm

vii. Implement the buffer-based algorithm in `PureBufferOccupancyRule.js`:
    1. Set `MIN_BUFFER_LEN = 10` seconds
    2. Set `MAX_BUFFER_LEN = 50` seconds

d. Re-build your dash.js files
    i. Run grunt default task:
        1. `$ /path/to/node/bin/grunt`
    ii. Note that each time you modify the library you need to build it. But each time you modify the client, you do not have to build the library.

7. Dash.js APIs

a. You may use some library APIs to implement your algorithm, here we list some `dash.js` modules and classes. Note that this is not an exhaustive list, you may need to walk through `dash.js` code and documentation.
b. Recall that a `config` object is passed to your module at the beginning. This `config` object can be used to access `DASHMetrics`, `MetricsModel` and `Debug` objects. Check `StreamController.js` to learn how the `config` object is set and what it contains.
c. `RulesContext`: first input to "`execute`" function. `RulesContext` contains information about the current quality level and `MediaInfo` object
d. `MediaInfo`: contains information about representation count, video codec and mime type
e. `DASHMetrics`: you can access the current buffer level, scheduling information and representation switch
f. `SwitchRequest`: a `SwitchRequest` object communicates your algorithm decision to the main rate controller in `dash.js` (called `AbrController`). If you do not create a `SwitchRequest` object and pass it to `AbrController`, then your algorithm decisions will not have effect.
g. How should you pass the `SwitchRequest` to `AbrController`?
    i. Hint: "`execute`" function accepts `callback` as a second argument.
h. External API documentation:
    i. http://cdn.dashjs.org/latest/jsdoc/module-MediaPlayer.html
    ii. http://cdn.dashjs.org/latest/jsdoc/module-DashMetrics.html

8. Evaluation

a. You will compare your buffer-based algorithm against the `dash.js` throughput-based one
b. You will evaluate the two algorithms under seven network profiles and conditions.
c. To use the network traffic control, check the next section
d. For each network profile, report the following performance metrics of dash.js vs your algorithm:
    i. Figure: Rendering bitrate (x-axis: time, y-axis: bitrate)
    ii. Figure: Buffer level (x-axis: time, y-axis: level)
    iii. Buffering rate (one value)
e. Report the metrics of the first 3 minutes of the playback only

9. Setup network traffic control

a. Traffic control (`tc`) is a user-space binary supported by recent Linux kernels
b. It allows users control the scheduling algorithms of network interfaces as well as their throughput, delay, loss rate, ordering etc…
c. In your experiments, you will use tc to emulate network conditions of the local loopback interface (`lo`). This means you do not need more than one (virtual) machine to run these experiments.
d. We provide nine scripts: two general scripts to control the local interface, and one script for each network profile.
e. The first two scripts are: `simulate-network.sh` and `net-reset.sh`
    i. `simulate-network.sh`: accepts interface name, command (set|reset), bandwidth and delay
    ii. `net-reset.sh`: resets local interface to its original configuration
f. To setup your network for profile# X, run the script `net-scenario-X.sh`
    i. when the experiment is done, run `net-reset.sh`
g. Note that the local loopback interface provides a high speed network (few Gbps). This speed is not realistic in the real Internet networks. That is why you need to run the previous scripts before starting the experiment.
h. For network profiles #6 and #7, you need to start the streaming session as soon as you run the scripts. This is because these scripts change network condition over time.

Deliverables

1. **Final PDF Report contains the following:**
    a. Your name and SFU student ID
    b. Your buffer-based algorithm pseudo code plus default parameters values (if any)
    c. Experiments figures of your algorithm against `dash.js` algorithm:
        i. rendering bitrate figure per experiment
        ii. buffer level figure per experiment
    d. One table of the buffering rate:
        i. Two columns for buffer-based and `dash.js` algorithm
        ii. One row for each network profile
        iii. Example:

	Buffer-based	Dash.js
Profile 1		
...		
Profile N		

    e. Analysis of results
        i. Discuss the behaviour of the two algorithms
        ii. Provide systematic reasoning about their behaviours
        iii. Report further interesting findings. Examples:
            1. Changing algorithm parameters
            2. Adding more network profiles
            3. Results of multiple users sharing same network link

2. **In-class demo:**
    a. For the demo, please make sure to set `MIN_BUFFER_LEN = 10` seconds and `MAX_BUFFER_LEN = 50` seconds
    b. We will run two or three unseen network scenarios to assess your algorithm

3. **Source code:**
    - `PureBufferOccupancyRule.js`