A Video Streaming System With Dash.js base on BBA rule

Team ID

SWS3021T5

Team Member

- **Wang Rui**, from BUPT
- **Ying Hang**, from SU
- **Zhan Zhengdao**, from SUSTech

Greatly Thanks to

- Abdelhak Bentaleb, from NUS
- Qian Tangjiang, from UESTC
- Roger Zimmermann, from NUS

You can visit our ***Project Website***.

Table of Contents

tags: DASH.js BBARule Video Streaming

Brief Introduction

This project is a video streaming system using BBA rules based on DASH.js. Users can upload their own video files to the server where they will be chopped into segments of about 3 seconds each before generating the related `mpd` file automatically. Then the player will load and play the `mpd` file, which implements BBA rules to streaming adaptation according to ***A buffer-based approach to rate adaptation: evidence from a large video streaming service*** [1].

Video Segmentation & MPD Generation

Once the video finishes uploading process to the server, a script will be triggered for segmentation. It uses *ffmpeg* for both transcoding and segmentation. The code is followed.

```
ffmpeg -i final.mp4 -map 0:v:0  -map 0:v:0 -map 0:v:0 -map 0:v:0 -map 0:a:0 \
-b:v:0 700k -filter:v:0 "scale=426:240" -profile:v:0 high \
-b:v:1 1000k -filter:v:1 "scale=640:360" -profile:v:1 high \
-b:v:2 2000k -filter:v:2 "scale=854:480" -profile:v:2 high \
-b:v:3 4000k -filter:v:3 "scale=1280:720" -profile:v:3 high \
-f dash -min_seg_duration 3000 -use_template 1 -use_timeline 1 \
-init_seg_name 'original/$Bandwidth$/init-stream.mp4' \
-media_seg_name 'original/$Bandwidth$/$Number%05d$.m4s' \
-adaptation_sets "id=0,streams=v  id=1,streams=a" manifest.mpd
```

Here in our test code, *ffmpeg* filters the original video into four different bitrates, i.e. 700kbps, 1000kbps, 2000kbps and 4000kbps. As a result, four sets of video chunks and one set of audio chunks will be created for video quality switch. Additionally, an according mpd file will be created afterwards.

Detailed Code Illustration

- `-i` ---- input file
- `-map 0:v:0` ---- set output video tracks from input， which can be indicated from v:0 to v:9
- `-map 0:a:0` ---- audio tracks, the same as video tracks
- `-b:v:0` ----- set video track bitrate
- `-filter:v:0` ---- set video track scale
- `-profile:v:0` ---- set track profile (baseline, main, high, high10, high422, high444)
- `-f dash` ---- force output file format to DASH
- `-min_seg_duration` ---- set segment duration in milliseconds
- `-use_template 1` ---- use SegmentTemplate in MPD
- `-use_timeline 1` ---- use SegmentTimeline in MPD
- `-init_seg_name` ---- set the name of initiation files
- `-media_seg_name` ---- set the name of chunks
- `-adaptation_sets` ---- set adaptation sets, here one for video, another for audio
- `manifest.mpd` ---- the output name of MPD

More info please refer to ***FFmpeg Formats Documentation*** *or* ***FFmpeg Documentation****.*

Set Customized Algorithm in DASH.js

To add/change adaptation algorithms, it's necessary for users to edit 5 files in DASH.js folder. Here we use DASH.js's default player.

1. Constants.js

> Add ABR_STRATEGY_...

```
        this.ABR_STRATEGY_DYNAMIC = 'abrDynamic';
        this.ABR_STRATEGY_BOLA = 'abrBola';
        this.ABR_STRATEGY_THROUGHPUT = 'abrThroughput';
        this.ABR_STRATEGY_BBA0 = 'abrBBA0';
        this.MOVING_AVERAGE_SLIDING_WINDOW = 'slidingWind
        this.MOVING_AVERAGE_EWMA = 'ewma';
```

2. ABRRulesCollection.js

> Import BBARule class

```
import DroppedFramesRule from './DroppedFramesRule';
import SwitchHistoryRule from './SwitchHistoryRule';
import BolaRule from './BolaRule';
import FactoryMaker from '../../../core/FactoryMaker';
import SwitchRequest from '../SwitchRequest';
import BBA0Rule from "./BBA0Rule";
```

> Push BBARule class to *qualitySwitchRules* Remove other classes pushed by *qualitySwitchRules* if not needed

```
qualitySwitchRules.push(
    BolaRule(context).create({
        metricsModel: metricsModel,
        dashMetrics: dashMetrics,
        mediaPlayerModel: mediaPlayerModel
    })
);
qualitySwitchRules.push(
    ThroughputRule(context).create({
        metricsModel: metricsModel,
        dashMetrics: dashMetrics
    })
);
qualitySwitchRules.push(
```

```
qualitySwitchRules.push(
    BolaRule(context).create({
        metricsModel: metricsModel,
        dashMetrics: dashMetrics,
        mediaPlayerModel: mediaPlayerModel
    })
);
qualitySwitchRules.push(
    BBA0Rule(context).create({
        metricsModel: metricsModel,
        dashMetrics: dashMetrics,
        mediaPlayerModel: mediaPlayerModel
    })
);
```

> Remember to keep *abandonFragmentRules* for sudden fragment switch on the occasion that current segment cannot be downloaded properly due to bandwidth sudden change. Of course, *abandonFragmentRules* can be customized.

```
            // );
            abandonFragmentRules.push(
                AbandonRequestsRule(context).create({
                    metricsModel: metricsModel,
                    dashMetrics: dashMetrics,
                    mediaPlayerModel: mediaPlayerModel
                })
            );
        }
```

3. MediaPlayerModel.js

> Change BUFFER_TO_KEEP to set the required buffer length to run again after player stops Change DEFAULT_MIN_BUFFER_TIME to set max buffer length

```
const BUFFER_TO_KEEP = 3;
const BUFFER_AHEAD_TO_KEEP = 80;
const BUFFER_PRUNING_INTERVAL = 10;
const DEFAULT_MIN_BUFFER_TIME = 60;
const DEFAULT_MIN_BUFFER_TIME_FAST_SWITCH = 3;
const BUFFER_TIME_AT_TOP_QUALITY = 60;
```

> Change ABRStrategy to customize default strategy if needed

```
    function setup() {
        UTCTimingSources = [];
        useSuggestedPresentationDelay = false;
        useManifestDateHeaderTimeSource = true;
        scheduleWhilePaused = true;
        ABRStrategy = Constants.ABR_STRATEGY_BBA0;
        useDefaultABRRules = true;
        fastSwitchEnabled = false;
```

> Add options condition to value

```
HROUGHPUT || value === Constants.ABR_STRATEGY_BBA0) {
```

4.samples/dash-if-reference-player/app/main.js

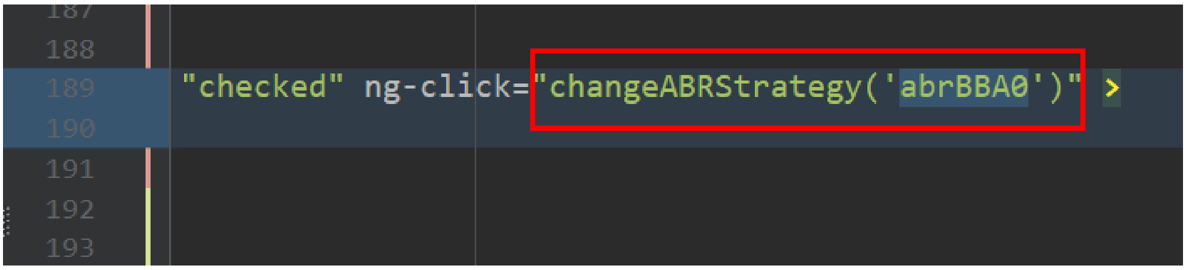
> Change ABRStrategy to customize default strategy if needed

```
    $scope.fastSwitchSelected = false;
    $scope.videoAutoSwitchSelected = true;
    $scope.videoQualities = [];
    $scope.ABRStrategy = 'abrBBA0';

    // Persistent license
```

5.samples/dash-if-reference-player/index.html

> Change *ng-click* of targerted button, which will trigger proper algorithm if selected May also need to change other texts of the button if needed



BBA Rule Implementation

The general idea of *Buffer-Based Approach to Rate Adaptation* is to select upcoming video chunks according to current buffer level. The paper[1] shows the base algorithm as well as its two-level optimizations, which have all been implemented in this project.

BBA0

#

BBA0 is a relatively conservative adaptation algorithm. In many cases, the video quality tends to increase step by step. The key of this algorithm is to define a rate map which indicates a discrete bijective relation from buffer level to video rate, namely quality to play. To ensure the playing fluency in low bandwidth condition, the designers added a reservoir to the algorithm. At first, the algorithm chooses the lowest video rate as default. Only when the buffer level outtakes the volume of reservoir will the player be allowed to switch to video chunks with higher bitrate. Then the algorithm of the cushion part takes control.

When the rate indicated by the bijective relation surpasses `Rate+` , the program will choose the maximum rate that is lower than the indicated one. In the same way, the minimum rate that is higher than the indicated one is planned to be selected if `Rate-` is larger. Obviously, this strategy always tries to stabilize the video rate since it keeps fluctuating in a smaller range while changing the rate according to buffer-rate relation.

The pseudocode[1] is followed.

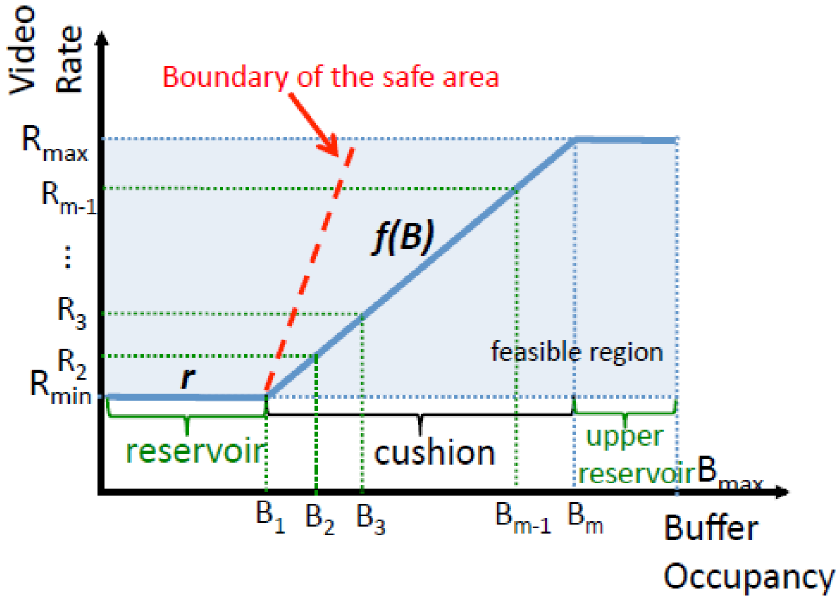
```
Input:  Rate_prev: The previously used video rate
        Buffer_now: The current buffer occupancy
        r: The size of reservoir
        cu: The size of cushion
Output: Rate_next: The next video rate

if Rate_prev = Rate_max then
    Rate+ = Rate_max
else
    Rate+ = min{Ri : Ri > Rate_prev}
if Rate_prev = Rate_min then
    Rate- = Rate_min
else
    Rate- = max{Ri : Ri < Rate_prev}

if Buffer_now <= r then
    Rate_next = Rate_min
else if Buffer_now >= (r + cu) then
    Rate_next = Rate_max
else if f(Buffer_now) >= Rate+ then
    Rate_next = max{Ri : Ri < f(Buffer_now)};
else if f(Buffer_now) <= Rate- then
```

```
    Rate_next = min{Ri : Ri > f(Buffer_now)};
else
    Rate_next = Rate_prev;

return Rate_next;
```

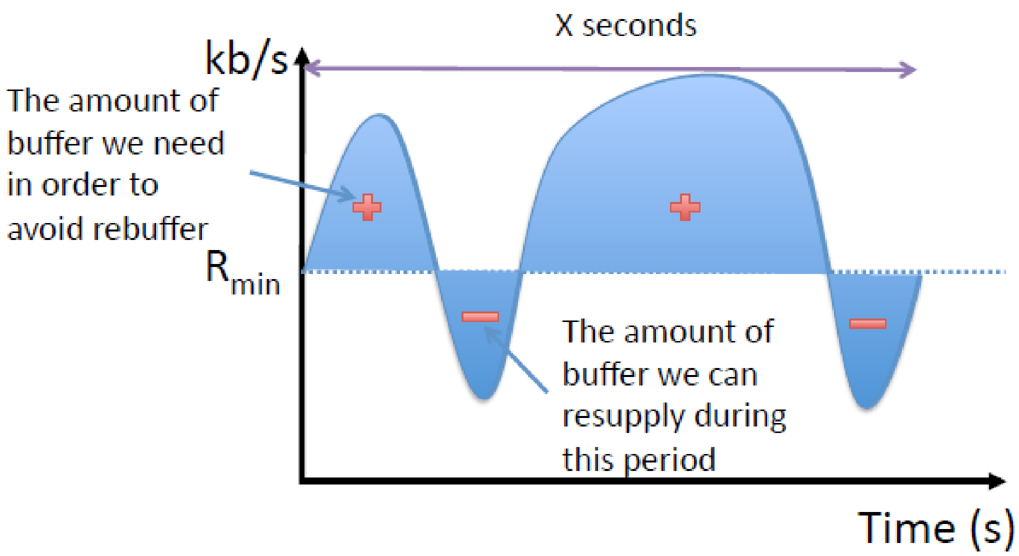


BBA1

Based on BBA0, BBA1 optimizes the former version by making reservoir dynamically change as well as transforming from buffer-rate map to buffer-chunk map, consequently taking the size of different chunks into consideration.

The prerequisite of this improvement is to retrieve next several chunks' sizes from the server while running the algorithm. Our solution is to get the chunks sizes, storing into an text file once the segmentation is accomplished. Then when the program starts to run, the size information will be got in the setup function, making them could be used after certain segment downloads.

According to the paper[1], it sets a duration, twice as the max buffer level, for reservoir adjustment test. ' *By summing up the amount of buffer the client will consume minus the amount it can resupply during the next X seconds, we can figure out the amount of reservoir we need.* ', it says.



As shown in the above graph, we need to calculate the sum of all blue area above the *Rmin* line minus that below the line. To simplify the process, the algorithm will calculate the difference between the area below the chunk variation line and that below the *Rmin* line.

Talking about the chunk map, BBA1 only changes the mapping relation compared to BBA0's buffer-rate mapping strategy. The main idea, as descripted in the article, is that *the algorithm stays at the current video rate as long as the chunk size suggested by the map does not pass the size of the next upcoming chunk at the next highest available video rate (Rate+) or the next lowest available video rate (Rate-)* . This means that the standard is changing since the next upcoming chunk size is not fixed.

The pseudocode is shown as followed.

BBA-1

2019年7月19日 星期五 16:39

quality ID　　segment ID

Input : $Rate_{prev}$, Buf_{now}, r, cu, chunksize[ ][ ]

Output : $Rate_{next}$

if $Rate_{prev} = R_{max}$ then

　　$Rate_{+} = R_{max}$

else

　　$Rate_{+} = \min\{R_i : R_i > Rate_{prev}\}$

if $Rate_{prev} = R_{min}$ then

　　$Rate_{-} = R_{min}$

else

　　$Rate_{-} = \max\{R_i : R_i < Rate_{prev}\}$

if $Buf_{now} \le r$ then

　　$Rate_{next} = R_{min}$

else if $Buf_{now} \ge r + cu$ then

　　$Rate_{next} = R_{max}$

else if $f(Buf_{now}) \ge$ Chunksize[$Rate_{+}$ID][upcoming seg ID]

　　$Rate_{next} = \max\{R_i : R_i < Rate(f(Buf_{now}))\}$

else if $f(Buf_{now}) \le$ Chunksize[$Rate_{-}$ID][upcoming seg ID]

　　$Rate_{next} = \min\{R_i : R_i > Rate(f(Buf_{now}))\}$

else

　　$Rate_{next} = Rate_{prev}$

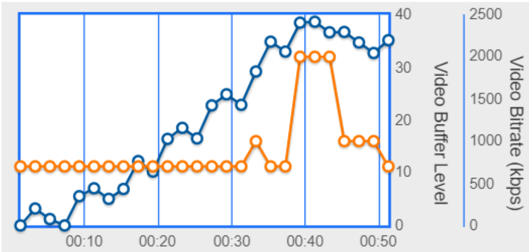
BBA2

Due to the effect of reservoir, both BBA-0 and BBA-1 suffer from slow startup since they will ignore the current bandwidth and use the minimum video rate as default if the current buffer level stays in the duration of reservoir. To solve this problem and avoid sacrificing overall bitrate, BBA-2 calculates the difference between input and output duration, namely the change of buffer level. According to the paper, $\Delta B, V - ChunkSize/c[k]$, should reach a level larger than 0.875 * input segment length before the quality is allowed to be switched to higher ones.

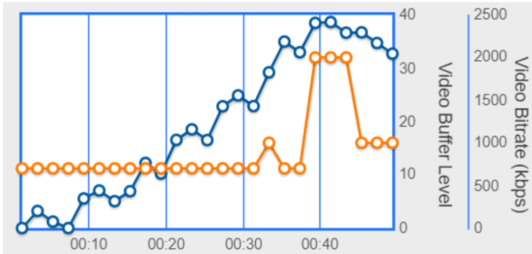
Performance Comparsion

Startup at 1000Kbps, shaped by Chrome

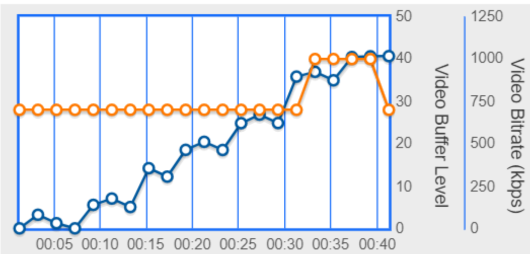
BBA0



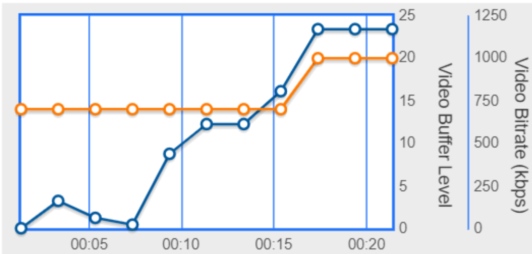
BBA1



BBA2

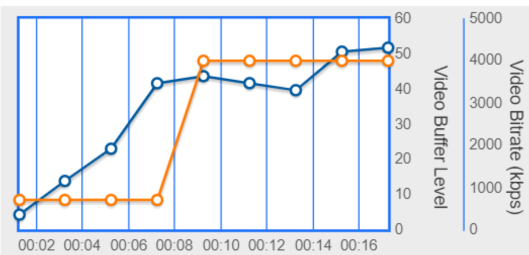


Bola

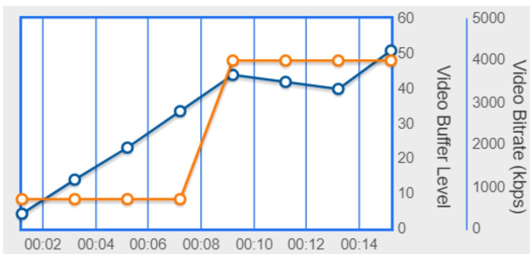


Startup at 5900Kbps, shaped by Chrome

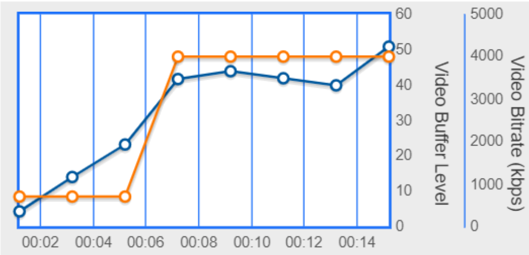
BBA0



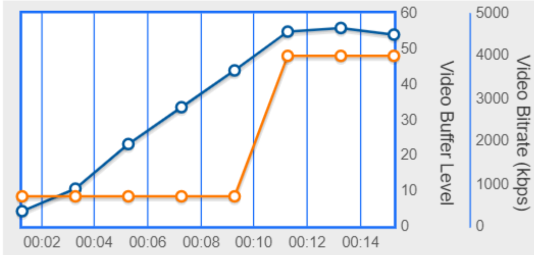
BBA1



BBA2

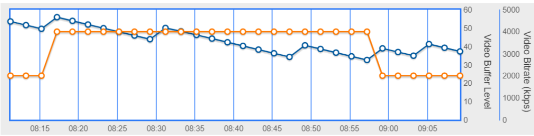


Bola

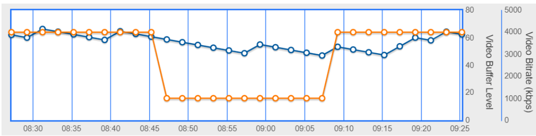


Script test, shaped by Abdelhak Bentaleb

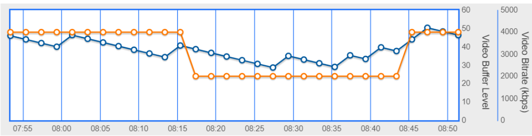
BBA0



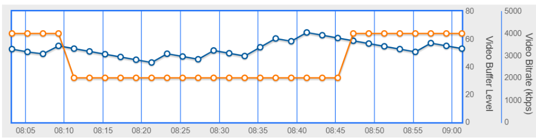
BBA1



BBA2



Bola



Web Work

At this point, it's not enough to just complete the algorithm, so we're going to build our own website to demonstrate our project. At the same time, in order to prepare show case, it is necessary to establish an interactive user interface.

Since we need that users can upload and transform videos all automatically, we need to call shell scripts at the back end.

Taking all these factors into account, we choose node.js as the back-end language and use the Express framework.

- In order to satisfy the function of uploading， we choose multer module of Express. We first add an input element in our html page, then it is followed by a js script to send a post request. The backend codes deploying the multer module are as follows.

```
const multer = require('multer');
const storage = multer.diskStorage({
        destination: (req, file, cb) => {
        const uploadFolder = './video';  // Directory of uploaded files
        try {
            fs.accessSync(uploadFolder);
        } catch (error) {
            fs.mkdirSync(uploadFolder);
        }
        cb(null, uploadFolder);
    },
    filename: (req, file, cb) => {
```

```
        cb(null, file.originalname);
    }
});
const upload = multer({             // Instantiate Multer object
    storage: storage
});
```

- After uploading, we need to transform the video and generate mpd file immediately. So we use **Child Process | Node.js** to run shell script to do that. The backend codes receiving and transforming videos are as follows.

```
app.post('/upload', upload.single('myFile'), (req, res) => {
    var child_process = require('child_process');
    child_process.execFile('./s0.sh', function(error, stdout, stderr){
        if(error){
            throw error;
        }
        res.send({
            message: 'Upload and transform successfully'
        });
        console.log(stdout);
        console.log("success");
    });
});
```

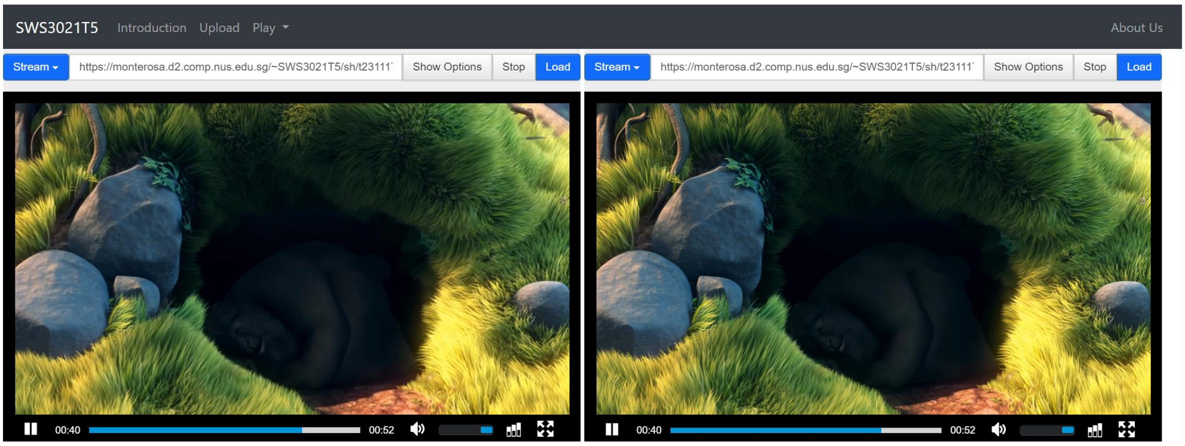
- According to our BBA algorithm, wo also need to generate a txt file which contains the standard out of another shell script. So we do like this `sh s1.sh > chunks.txt` to run and record script.

When implementing the front-end interface, we use **Bootstrap 4** framework, which is easy and aesthetic.

- In order to alleviate the pressure on the server, we use CDNs to load our style and scripts.

```
<link rel="stylesheet" href=
"https://maxcdn.bootstrapcdn.com/bootstrap/4.3.1/css/bootstrap.min.css">
<script src=
"https://ajax.googleapis.com/ajax/libs/jquery/3.4.1/jquery.min.js">
</script>
<script src=
"https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.14.7/umd/popper.min.js">
</script>
<script src=
"https://maxcdn.bootstrapcdn.com/bootstrap/4.3.1/js/bootstrap.min.js">
</script>
```

- We also embed an original dash player and our dash player using BBA algorithm in one page for us to compare. It lets us compare not only the data but also realistic perception.



- We also customize our Dash.js player, such as changing layout, delete useless element, and set default mpd link.

```
app.controller('DashController',function ($scope,sources,
contributors, dashifTestVectors) {
    $scope.selectedItem = {
        url:
'http://monterosa.d2.comp.nus.edu.sg/~SWS3021T5/sh/t221445/manifest.mpd'
    };
```

Last but not least, all needed resources can be accessed via internet. No outside files or directories are required. All you need is internet and a browser like Chrome.

Challenges & Solutions

- **Q:** When using MP4Box to generate MPD file , we found that the player would easily get stuck and keep asking for a certain chunk. **A:** We think that it is the MPD file that probably cause the problem, but after many tests, we still could not ensure what leads to that problem. As a result, finally, we decide to use ffmpeg instead, which avoids the problem.
- **Q:** Our team have no idea to identify which rule we are using. We found that after adding console output to different rules' *getMaxIndex* methods, all the output would be printed at the same time no matter which rule we were choosing at that time. **A:** According to Mr.Bentaleb's explanation, what results in that consequence is that we use *dash.all.debug.js* for grunting, which will run all the algorithms but actually use the selected one by default. It's suggested that we should `grunt dis` and implement DASH.js with *dash.all.main.js* .
- **Q:** At first, we have no idea about how to retrieve the size of current downloaded chunk. **A:** With Mr.Bentaleb's instructions, we realize the problem with the code followed.

  ```
  let dashMetrics = config.dashMetrics;
  let LasthttpRequest = dashMetrics.getCurrentHttpRequest(metrics);
  let SizeSegByte = LasthttpRequest.trace.reduce((a, b) => a + b.b[0], 0);
  ```

- **Q:** To implement BBA1 algorithm, the sizes of next upcoming chunks are required to estimate the reservoir adjustment and buffer-chunk mapping. But DASH.js doesn't provide the certain function to retrieve the information of the following segments. **A:** The only solution is to prepare the information of chunks of different qualities before running the algorithm. We have written a script, one that will be triggered once the segmentation finishes, to get the size of chunks and store it into .txt file on the server. Before running the algorithm, we just need to load the file into array to use it.

Reference

[1] Te-Yuan Huang, Ramesh Johari, Nick McKeown, Matthew Trunnell, and Mark Watson. 2014. A buffer-based approach to rate adaptation: evidence from a large video streaming service. In Proceedings of the 2014 ACM conference on SIGCOMM (SIGCOMM '14). ACM, New York, NY, USA, 187-198. DOI: **https://doi.org/10.1145/2619239.2626296**

tags: **DASH.js** **BBARule** **Video Streaming**