An analysis of adaptive video streaming with DASH

Max Crone
max.crone@aalto.fi

Jack Henschel
jack.henschel@aalto.fi

***Abstract*—In this report we investigate the behaviour of Adaptive Bitrate Streaming with the DASH protocol under different network conditions with regard to video quality and fairness between clients. We found that the adaption to static and changing network conditions works quite well to always provide clients with the highest possible video quality while minimizing buffering delays.**

***Index Terms*—video, adaptive, streaming, dash**

I. Introduction

In recent years, the MPEG-DASH standard has unified the previously diverse and proprietary landscape of Adaptive Bitrate Streaming formats. Adaptive Bitrate Streaming greatly enhances the Quality of Experience (QoE) for the user because the video quality dynamically adjusts to network conditions, thus it reduces buffering delays and at the same time maximizes quality for the user. With MPEG-DASH, video streaming providers need to re-encode videos into multiple qualities and create a so-called "manifest file" that contains metadata about the video, of which the URLs of the different quality segments are the most important. The DASH manifest is commonly stored in a `.mpd` file (Media Presentation Description) and is formatted as XML. The client (e.g. web browser or mobile device) will then use the information from the manifest to download the required audio and video segments. In the first part of this report we set up a streaming server, streaming client, and preprocess a video file so it can be used for Adaptive Bitrate Streaming. Then, in the second part, we evaluate how the adaptation behaves in a variety of network conditions.

II. Experimentation setup

Firstly, we obtained a high-quality video from the Blender Foundation with which we are going to conduct our experiments: "Big Buck Bunny Trailer" [1]. To have a video with sufficient length, we looped this video multiple times which resulted in a total length of 5 minutes and 30 seconds.

A. Transcoding

Next, we looked for tools to transcode our original input video with different quality settings. FFmpeg is the Swiss army knife of video encoding and packaging. It supports all commonly used formats, containers and codecs and is available under a Free Software license [2]. Thus it was a natural choice for the transcoding process.

The most important aspect for encoding is to align the keyframes between all the qualities. Keyframes contain the entire picture of a video without reference to any other frames of the video. As they incorporate all of the information about the pixels in each image, they take up a lot more space and are much less frequent than the other types of encoding frames. After encoding, the video needs to be divided into short segments where each segment has to start with a keyframe. If the keyframes across different qualities of the same video are not aligned, the lengths and positions of the segments will not match, making a fluent transition from one quality to another one impossible. Thus, all video qualities have to contain regular keyframes.

Fig. 1. Transcoding command for FFmpeg

```
ffmpeg -y -i "$VIDEO" \
 -c:v libx264 \
 -x264opts 'keyint=24:min-keyint=24:no-scenecut' \
 -b:v 700k -maxrate 700k -bufsize 500k \
 -vf "scale=-1:480" \
 low.mp4
```

Figure 1 shows an example command of how to encode the original video with FFmpeg. We specify the input file with `-t`, set the encoder to "x264", the encoding options ensure there is a keyframe every 24 frames (one per second), the desired bitrate (in this case 700 kb/s), the resolution of the output video (here the height is given with 480 pixels and the width will be automatically determined) and finally the output file.

Using this command, we converted the original source into three different videos according to the settings in Table I.

TABLE I
Quality settings for each video

Quality	**Bitrate** (kb/s)	**Resolution** (pixels)	**Filesize** (MB)
Original	7000	1920x1080	295
Low	700	960x540	20
Medium	1500	1280x720	41
High	2500	1920x1080	69

B. DASH Manifest

The next step is to package the video file for the clients and generate the DASH manifest. For this we used the MP4Box tool by GPAC which can be used to (among other features) prepare HTTP Adaptive Streaming content [3]. Given the duration for each DASH chunk and the (already encoded) input videos, MP4Box will repackage the videos into a different container and generate the DASH manifest.

C. *Web server*

The final step is uploading the generated output files to a web server. First we experimented with Google Cloud Storage which provides a simple object storage interface and makes the content available on the internet. However, we found that the aggressive caching of this service made our measurements unreliable. Therefore we set up our own Nginx web server on a Google Cloud Compute instance. This gives us total control over the server settings as well as letting us control the network environment.

It is important to set the correct Cross-Origin Resource Sharing (CORS) headers for the web server, otherwise modern web browser will refuse to load the video segments. For DASH, the “Content-Type”, “Origin” and “Range” headers need to be allowed for HTTP GET, HEAD and OPTIONS methods.

D. *DASH client*

To ensure all clients (including mobile browsers and older devices) can access the DASH content, we used version 3.0.0 of the `dash.js` library. It is a reference implementation of an MPEG-DASH client in JavaScript developed by the Dash Industry Forum [4]. The JavaScript library simply needs to be given an HTML `<video>` element along with the URL of the DASH manifest and will then start fetching the appropriate video segments specified in the manifest file. Optionally, more configuration options can be supplied to the library via JavaScript.

Putting all these components together, we end up with the files listed in Table II on our web server.

TABLE II
WEB SERVER FILES

Filename	**Description**
`index.html`	Website
`dash.all.min.js`	DASH JavaScript Library (minified version)
`high_dash.mp4`	High quality video stream
`low_dash.mp4`	Low quality video stream
`medium_dash.mp4`	Medium quality video stream
`output_init.mp4`	Video initialization file (contains metadata)
`output.mpd`	DASH Manifest (Media Presentation Description)

III. ANALYSIS UNDER DIFFERENT NETWORK CONDITIONS

In this section we will conduct analyses by streaming the video under different network conditions and observing the resultant performance of the DASH client. We will also conduct a comparison between multiple clients streaming simultaneously in order to analyze fairness.

A. *Data collection*

In order to collect metrics on the playback we use methods provided by the reference DASH client implementation. These provide us with the current video quality, a video timestamp and the buffer level in seconds.

The quality of a segment is expressed in a number. These values over time are plotted together with the buffer level in the same graph such that observing relations between these different metrics becomes more intuitive. Table III gives the interpretations for the three levels of video quality.

TABLE III
LEVELS OF VIDEO QUALITY

Level	**Quality**
0	Low
1	Medium
2	High

In order to be able to throttle the client’s network connection, we made use of the Firefox Developer Tools. Out of the box, it provides profiles to emulate various network conditions. Table IV lists the names of the profiles we used as well as their performance characteristics. We did not list the upload speed since this metric is mostly irrelevant for video streaming from the perspective of the client.

TABLE IV
NETWORK CONDITIONS [5]

Profile	**Download Speed** (Kbps)	**Minimum latency** (ms)
2G	450	150
3G	1500	40
4G	4000	20
WiFi	30000	2

The rest of this section will go over the multiple scenario’s we set up for testing. The behavior of the clients will be analyzed and explained.

B. *Analysis*

1) Ideal case: The first test we conducted consisted of a single client fetching the video from the server with no bandwidth throttling. This serves as a baseline for all other experiments.

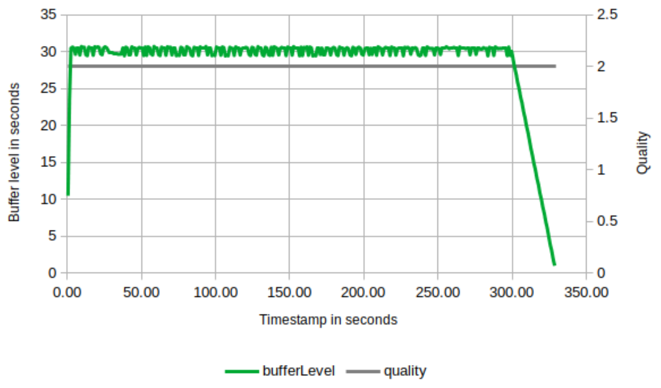

Fig. 2. Buffer level for the ideal case: a single client without a throttled connection.

From Figure 2 we can see that the client immediately fills up the playback buffer. The video is played back entirely in high quality and throughout the entire video playback there are no performance drops.

2) Stable 3G: In this first test making use of the throttling settings, we applied a single network condition (3G) to see how the client behaves with restricted bandwidth. Figure 3 illustrates that it takes the client around fifty seconds before it mostly consistently displays the high quality video segments. Even then, there are various drops to the medium quality video stream and even two to the lowest quality.

In general the adaption seems to strike a new balance between the size of the buffer and the amount of high quality segments it can display. Where the unthrottled case had its buffer size going up to thirty seconds, under the 3G network conditions the DASH client keeps the buffer size hovering around ten seconds, such that it can still manage to display the video for more than half of its duration in high quality.

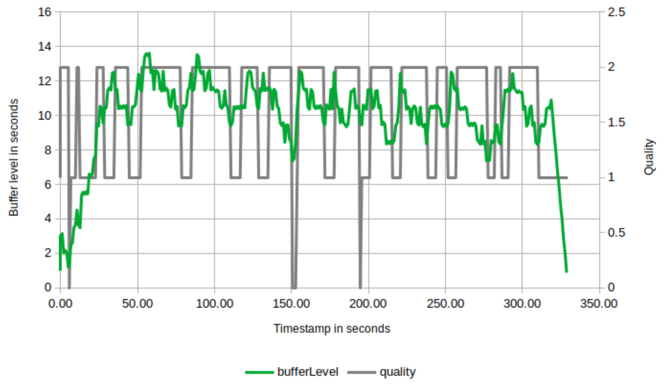

Fig. 3. Buffer level and quality over time for a good 3G connection.

3) Throttling levels: In the next experiment we started with a very low available bandwidth for a single client and then gradually increased it until the client had more bandwidth than necessary. After reaching the high point, we decreased the available bandwidth again in the same manner. See Table V.

TABLE V
CLIENT THROTTLING INTERVALS

Interval (s)	**Network**
0 - 30	2G
30 - 60	3G
60 - 90	4G
90 - 120	WiFi
120 - 150	4G
150 - 180	3G
180 - 329	2G

The results are included in Figure 4. It becomes clear that the client increases playback quality as the available bandwidth increases. Just like in the previous scenario, we see that under the 3G interval the client bounces back and forth between high and medium quality, while keeping its buffer size around ten seconds.

As soon as the bandwidth is upgraded to 4G, at around 60 seconds in, the quality stays high consistently and the buffer is being filled until it is thirty seconds in size. During the WiFi interval this trend continues, which is in line with what we expected based on the first, ideal scenario.

During the second 3G interval the client is presented nearly exclusively with high quality segments, while during the first 3G interval and also in the stable 3G scenario we saw that quality tended to interchange between medium and high. In this specific case however, the client could sustain on the buffered high quality segments produced during earlier, higher bandwidth intervals. That is why the high quality stays longer than one might expect based on the performance of the network condition profile in isolation.

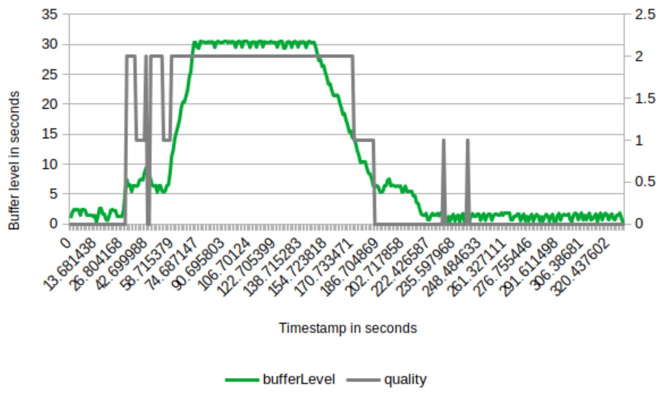

Fig. 4. Buffer level and quality over time under changing bandwidth conditions.

4) Short-term versus long-term throttling: In this next experiment we investigate how fast the DASH protocol reacts to changes in the network condition. To this end we conduct two tests. First we stream the video without any throttling, except for every minute when we throttle the connection to 2G conditions for a duration of 3 seconds. This is the short-term throttling scenario. The seconds tests proceeds in much the same way, except that we throttle the connection to 2G for a duration of 20 seconds every minute. This we call the long-term throttling scenario. The results can be found in Figure 5 and Figure 6, for the short-term and long-term scenarios respectively.

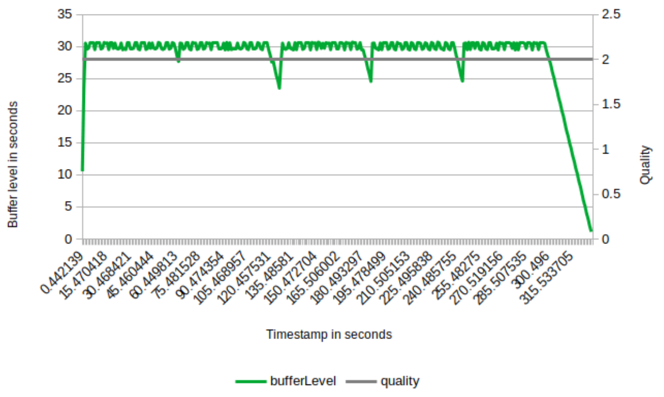

Fig. 5. Buffer level and quality over time for the short-term throttling scenario.

Clearly the short-term throttling has no effect on the quality of the segments that is presented to the user; they are always

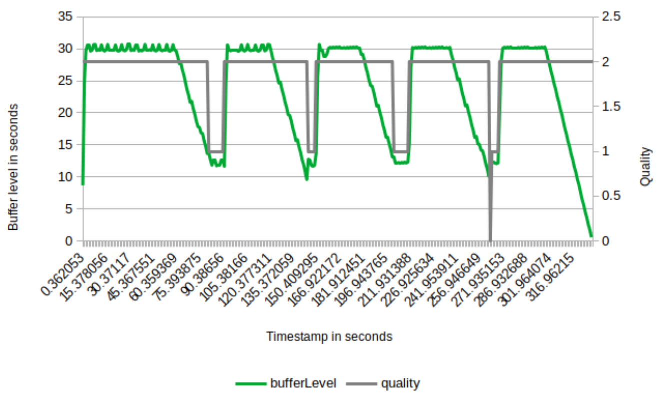

Fig. 6. Buffer level and quality over time for the long-term throttling scenario.

of high quality. We see momentary drops in the buffer level, but not much more than five seconds at a time.

In contrast, with the long-term throttling scenario there do occur drops in quality. Every minute we observe the drop in quality taking place around 21 seconds after initial throttling; just after the buffer was roughly halved in size. This seems to be an indicator that the DASH protocol reacts to. Coincidentally, by construction of our tests, that is also the moment that we stop the throttling on the network for the client. This has as effect that the buffer level stops dropping and stays constant for about 8 seconds. Then both the quality and the buffer level increase again. This means that the client was able to replace buffered segments of lower quality and build up a buffer with sufficiently many high quality segments in those 8 seconds.

We observe that even though the client has a 30 second buffer of high quality segments, it still drops the quality of playback to medium after 21 seconds. We expected it to at least present high quality segments to the user for 30 seconds. Apparently dropping the quality is part of the DASH protocol's strategy for managing its buffer while maximizing the playback experience for the user.

5) Fairness with two clients: In the next experiment we used two clients, both without throttling, but instead we limited the bandwidth on the server side to evaluate how fairly the available bandwidth is distributed between the clients. For this task we used the tool *wondershaper*. [6] The steps of available bandwidth for the server are illustrated in Table VI. We initialized the server with a total bandwidth of 8192 Kbps. After 90 seconds we halved the available bandwidth to 4096 Kbps, which was again halved to 2048 Kbps after another 90 seconds. Ultimately the available bandwidth to the server was reduced to 1024 Kbps at which it stayed until both clients finished streaming.

The results are illustrated in Figure 7 for the first client A and in Figure 8 for the second client B.

Comparing Figure 7 and 8, it can be seen that in the first part of the video the playback buffer develops similarly for both clients, but especially towards the end the available bandwidth is not fairly distributed between the clients. Nevertheless, both

TABLE VI
SERVER BANDWIDTH THROTTLING

Interval (s)	**Bandwidth** (Kbps)
0 - 90	8192
90 - 180	4096
180 - 270	2048
270 - 329	1024

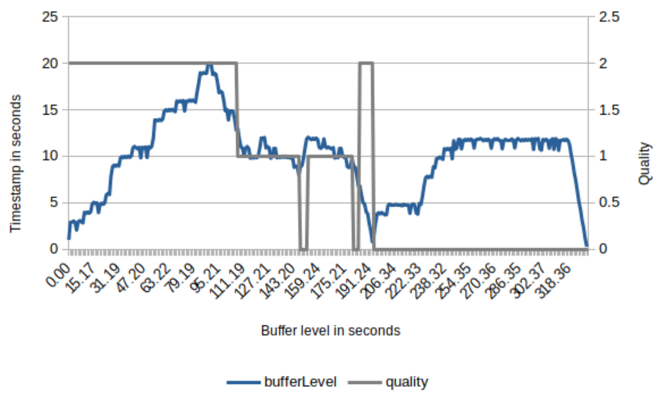

Fig. 7. Buffer level and quality over time for client A in the case of server throttling.

clients experience the same video quality, which is the most important metric for the end user from the perspective of Quality of Experience. In general, the video quality level of client B appears more volatile compared to client A's video quality. A possible explanation for this might be the particular network conditions in our test environment (i.e. WiFi).

IV. Conclusion

In this report we investigated the behaviour of Adaptive Bitrate Streaming with the DASH protocol under different network conditions with regard to video quality and fairness between clients. We found that the adaption to static and changing network conditions works quite well to always provide the user with the highest possible video quality while minimizing buffering delays.

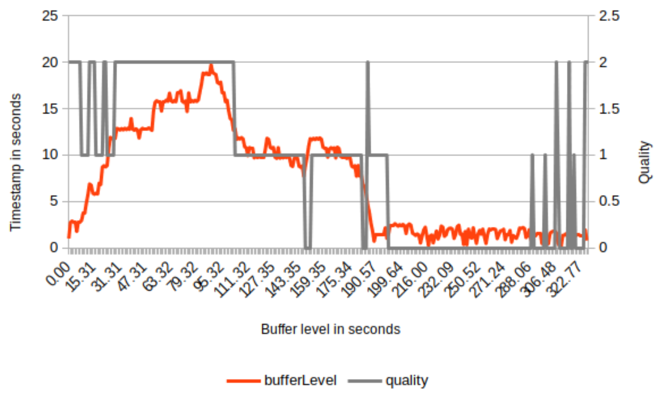

Fig. 8. Buffer level and quality over time for client B in the case of server throttling.

One limitation of our experiments was the low number of clients. Video streaming servers usually deal with thousands of clients at the same time which can lead to complex and unpredictable behaviours. In our tests we were not able to simulate these conditions.

References

[1] Blender Foundation, "Big Buck Bunny Trailer", https://peach.blender.org/trailer-page/, 2008.

[2] FFmpeg team, "FFmpeg Website", https://ffmpeg.org/, 2019.

[3] GPAC, "MP4Box Website", https://gpac.wp.imt.fr/mp4box/, 2019.

[4] Dash Industry Forum, "dash.js", https://github.com/Dash-Industry-Forum/dash.js, 2019.

[5] Mozilla, "Firefox Developer Tools Throttling", https://developer.mozilla.org/en-US/docs/Tools/Network_Monitor/Throttling, 2019.

[6] Bert Hubert, Jacco Geul & Simon Séhier. "magnific0/wondershaper: Command-line utility for limiting an adapter's bandwidth". https://github.com/magnific0/wondershaper. 2019.