The Effect of Bandwidth on the Adaptive Behaviour of MPEG-DASH

Alazar Alemayehu Abebaw
Student No. ******
alazar.abebaw@aalto.fi

Axel Ilmari Neergaard
Student No. ******
axel.neergaard@aalto.fi

***Abstract*—This paper investigates the adaptive streaming behavior of MPEG-DASH over a controlled network with fluctuating bandwidth. Network bandwidths were controlled and observed in 4 different experiments: gradual decrease; gradual increase; gradual decrease then gradual increase; and gradual increase then gradual decrease. Bandwidth control was performed through deliberate throttling at levels that ensured the attainability of all stream qualities for our chosen multimedia player. The stream consisted of a video encoded into segments of 4 seconds, and was accessed through the DASH Industry Forum's reference player, dash.js. From our results we conclude that the dash.js multimedia player—with small buffer settings and throughput dependant history—optimizes for stream stability by keeping the buffer filled at all times. The throughput history of the stream heavily affects whether the player attempts to up-switch the video bitrate. Once the player is able to reach high buffer levels in high network bandwidths scenarios, it will more gradually down-switch as it is able to server higher quality video from its buffer.**

***Index Terms*—MPEG-DASH dash.js bandwidth adaptive**

I. Introduction

The boom in the video streaming industry has led to the need to deliver uninterrupted, high quality videos. However, the quality of experience between users—or even between uses for the same user—vary with network fluctuations. This requires streaming providers to support video and audio streams that adapt to fluctuating network conditions. One such solution is the widely adopted Dynamic Adaptive Streaming over HTTP (MPEG-DASH or DASH) standard, which supports the adaptive streaming of different video and audio qualities over pre-defined network bandwidths, increasing stream stability and quality of experience. This adaptive nature is achieved by encoding a video (with or without audio) into segments of pre-defined bitrate and length, which correspond to the qualities for network bandwidths. A multimedia player can choose, or be configured to, adapt the stream bitrate depending on different criteria, such as the current buffer level of the player or a history of previous segment downloads.

In this paper we will examine the behaviour of the DASH Industry Forums reference player, dash.js, configured to use a small buffer target size and the average throughput of previously video segment requests. The player requests the video segments from a controlled Hypertext Transfer Protocol (HTTP) server over a controlled local area network. The player is continually polled for its current buffer level and the current video bitrate that it is serving the stream with over pre-defined and fluctuating network bandwidths. Additionally we periodically measure the calculated download bitrate based on byte-count decoding to find any correlation to future chosen video bitrate requests.

With our experiments we try to form a picture of how the dash.js multimedia player adapts the requested video bitrate when streaming through fluctuating bandwidth periods. We hypothesize that the player will not change the video bitrate too eagerly, but will rather try to keep its buffer full for a stable streaming experience.

II. Part I

In this part of the paper, we start by discussing the tools used in the experiment. Then we continue exploring the experiment setup and the steps followed in the setting up the experiment. Then we conclude the section by explaining the data collection process.

A. Tools

1) Apache HTTP: DASH—as the name would suggest—runs over HTTP, which means it requires a DASH compatible HTTP server to stream videos. For this purpose we decided to serve videos with Apache HTTP[16] (also known as httpd), which supports DASH out-of-the-box. For our experiments the default settings for Apache were used.

2) Docker: To host the HTTP server we decided against using cloud based solutions, as random network fluctuations would be challenging to account for. Instead we decided to use the readily available httpd Docker image (specifically version 2.4) to host the server locally on our machines and on our isolated local area networks[1]. This solution provided us with full control over the network between the server and the video viewer (client).

3) FFmpeg: FFmpeg is a multi-platform multimedia framework used to decode, encode, trans-code, filter, and to generate multi-bitrate (different qualities) of a video. We chose FFmpeg because it is easy to setup, comes as a pre-installed tool for many Linux distributions, and has good resources on Mozilla Developers Network. It also allowed us to control the different parameters such as segment length of the video.

4) Chrome Browser: For viewing the videos, we used the Google Chrome browser (version 87.0.4280.67) as a client. The Chrome browser enabled us to finely restrict the perceived

client network bandwidth by throttling through its integrated developer tools. We also disabled the caching feature of the browser to minimize experiment intereference. Additionally, Chrome—being a modern web browser—enabled us to embed JavaScript code into our custom experiment web page for analytics, which could run uninterrupted in the background as Chrome fetched the video stream. We decided against testing other browsers (such as Mozilla Firefox) to better control experiment variables. Other types (i.e., non-browsers) of multimedia players (such as VLC) were not considered for in this experiment as they could be JavaScript incompatible, and to control experiment variables. Although the Chrome browser natively supports MPEG-DASH through HTML5 [2, 3] we decided to use a more customizable video player, please refer to the *dash.js* section for more information.

5) Dash.js: dash.js is a client side MPEG-DASH player written in JavaScript by the DASH Industry Forum [4]. We decided to use dash.js as our player due to its aim at being an essentially free production quality framework, and its extensive documentation and customizability. We used the latest available release of dash.js at the time of writing, version 3.1.3[5].

6) Custom Webpage & JavaScript: For gathering analytics on streams (more on that in later sections) we used a custom webpage with embedded, custom JavaScript. The application programming interface (API) of dash.js allowed us to extract fine-grained data points of each video stream, inspired heavily by the dash.js samples page [6]. The custom webpage enabled more control over the stream behaviour, such as on-demand load of videos streams instead of load on webpage load (further enabling us to set initial conditions).

B. Experiment Setup

In this section of the paper, we discuss the experiment environment and setup. We start by describing the fragment generation experiment setup, followed by a discussion on the bandwidth throttling setup, and finish off with explanations on related dash.js settings.

1) Fragment Generation: For this experiment a pre-installed FFmpeg (version 4.4.2) binary on a Ubuntu 20.04 machine was used to encode the video of high quality (1920 by 1080 pixels) into different quality formats. Then a Media Presentation Description (MPD) file, containing information about the bandwidth at which each bitrate video will be played, was generated. The FFmpeg encoding command [7] was run to generate videos in three quality formats. Each quality of the original video was segmented/fragmented into 4 seconds. A segment of a video is a fragment of video that a client can read from the server with a single request. In our case, every time a client made a request to the server asking for a video file, the server would return a 4 second video fragment. The audio of the video was encoded only into one quality, as the audio bitrate does not affect the overall stream quality in notable way [8].

2) Bandwidth Throttling: Five custom bandwidths were defined using the throttling feature provided by the Network section of the Google Chrome Developer tools. These bandwidths were:

- 1125 Kbps
- 1500 Kbps
- 2000 Kbps
- 3000 Kbps
- 5000 Kbps

These bandwidths were chosen based trail and error. We noticed that throttling around the bandwidths defined in the MPD yielded an almost unbearable streaming experience. We did not further pursue the root of the error.

3) Dash.js settings: The dash.js framework supports both general media player settings (such as buffer stream behaviour and size) and more detailed settings (such as bitrate adaptation algorithms). This section explains the behaviour of settings related to our experiment, starting with general settings and then settings related to Adaptive Bitrate (ABR). For a full list of settings please refer to [9].

The `stableBufferTime` setting defines the internal player buffer target post startup, in seconds. This means that the player will try to keep the defined amount of content in its buffer at all times. Important to note is that larger values will result in less frequent adaptive bitrate switches.

The `bufferTimeAtTopQuality` setting defines the internal player buffer target when playing at top quality. This means that once the highest quality video bitrate is playing the buffer target will be changed to this value to increase stability and to maintain media quality.

The `movingAverageMethod` ABR setting defines ”the moving average method used for smoothing throughput estimates.”[10] In other words, this is the method the player will use to heuristically calculate the network throughput, which is used to adapt the stream quality. Two methods are available in dash.js: `slidingWindow` and `ewma`. The `slidingWindow` method calculates the average throughput using the last four segments downloaded. The `ewma` (stands for exponentially weighted moving average) computes the average using exponential smoothing. Explaining the inner workings of the methods are outside the scope of this paper, curious users are referred to the dash.js documentation [10].

The `ABRStrategy` ABR setting defines the ABR strategy (i.e. which stream bitrate should be downloaded) for the player, which can be either built-in strategies provided by dash.js or custom strategies. There are many built-in rules, but we are only concerned with the primary built-in rules. The primary built-in strategies are `abrThroughput`, `abrBola`, and `abrDynamic`. `abrThroughput` chooses stream bitrate based on recent throughput history. `abrBola` chooses stream bitrate based on current buffer level (with higher bitrates for higher buffer levels). `abrDynamic` switches between the two aforementioned rules in real time. For interested readers the dash.js documentation includes more information about built-in rules, and also includes some diagrams on the logic flow for strategies in general [11].

4) Analytics: The dash.js framework includes a DASH metrics module [12] through which a user can get metrics on,

e.g., buffer levels and fragment requests. For this experiment we decided to follow the stream monitoring sample from dash.js [13]. This provided us with the following metrics on a one second interval:

- buffer level of the player (i.e. the current segment amount in the buffer in seconds)
- current video bitrate
- calculated network bitrate (based on decoded byte count differences, based on seconds)

This was saved into a JavaScript Object Notation (JSON) object and downloaded as a file on user demand. For a complete view of the analytic files please refer to the implementation script here [13].

C. Data Gathering

In this section we first go through the experiment types with reproduction steps, and then the specific settings used with the dash.js player and some experiment prediction based on these.

1) Experiment Types: We performed four types of experiments in this paper, with the commonality of gradually decreasing or gradually increasing network bandwidths, or a combination of both. The network bandwidths were throttled and confined to the levels mentioned in the experiment setup II-B2 for 30 second periods. Video bitrate and the buffer level was measured at one second intervals. The quality of received video fragments were noted to ensure nothing had gone astray in the player setup. The specific experiments were:

1) Gradual bandwidth decrease: We throttled bandwidth to 5000 Kbps, gradually decreasing it to 1125 Kbps through intermediate bandwidths. The decrements were made every 30 seconds.
2) Gradual bandwidth increase: We throttled bandwidth to 1125 Kbps, gradually increasing it to 5000 Kbps through intermediate bandwidths. The increments were made every 30 seconds.
3) Gradual bandwidth increase then gradual decrease: We throttled bandwidth to be 1125 Kbps and increased it gradually, every 30 seconds, until it reached 5000 Kbps. Then we decreased the bandwidth again until it reached 1125 Kbps.
4) Gradual bandwidth decrease then gradual increase: We throttled bandwidth to 5000 Kbps and increased it gradually through intermediate bandwidths, every 30 second, until it gets to the 1125 Kbps. Then we increased the bandwidth every 30 seconds until it reaches 5000 Kbps. In this experiment we did one exception in our measurement: once the highest network bandwidth was reached we increased the period from 30 to 60 seconds for this specific bandwidth. We had noted through previous experiment runs that the player was slower at increasing the video bitrate when previous bandwidth history had been low. To allow the player to catch up with the fluctuating bandwidth and not sporadically change between video bitrates in short high network bandwidth periods, doubling the period was sufficient.

The tools were setup and experiments were run on a MacBook Pro, 2019 release, over a local area network.

2) Dash.js Player Settings: Except for the ones below, all settings for the dash.js player were left to default values. For a list of default values the reader is referred to [14].

As the video was encoded into 4 second long segments we wanted to restrict the player buffer time target to something close to the segments sizes. We predicted that this would spread out buffering events during relatively moderate network bandwidths, but still enable the player to play a stream in a stable manner. Relatively moderate network bandwidths mean that each segment download would take long enough for the player to quickly switch stream bitrates when the network bandwidth was changed. A buffer target of 4 seconds was tested, but this resulted in frequent rebuffering events on low network bandwidths, and in some cases player crashes. A buffer target of 5 seconds was also tested, but the rebuffering events were still frequent. The buffer target of 6 seconds was then found to be the sweet spot for minimizing rebuffering events.

Although the buffer target level above was set to 6 seconds, the dash.js player also supports setting the buffer target level to something else once it plays at top quality. We set this top quality target level to 10 seconds to enable the measurement of long term network bandwidth changes. As both of these settings talk about to the buffer time target, we have named the former buffer target the *normal buffer target* and the former the *top buffer target*.

We predicted that the dash.js player is slower at increasing stream bitrate once it has entered and stabilized itself when under low network bandwidth. To account for this, we enabled the `fastSwitchEnabled` setting such that the player could request and append higher quality stream bitrates close to the current view time.

The last setting that we modified was the ABR strategy for the player. The default setting is `abrDynamic` which dynamically uses the two other ABR strategies mentioned in the Dash.js Settings section II-B3. We switched to the `abrTroughput` setting to better control the ABR strategy of the player. `abrTroughput` was chosen as it uses the recent throughput history to predict the future throughput. We expect the player to thus not vary the stream bitrate too aggressively on consistent network bandwidth fluctuations.

III. Part II

In this part of the paper we get to the experimentation phase. We start off with a hypothesis on the adaptive behaviour of the dash.js player, and then proceeding to presentation and analyzation of our processed data. The raw gathered data (i.e. the JSON files from our custom webpage) are available in the experiment GitLab repository [15].

A. Hypothesis

In this experiment the dash.js player has been configured to use the `abrThroughput` ABR setting with the `slidingWindow` as the moving average method. Thus we

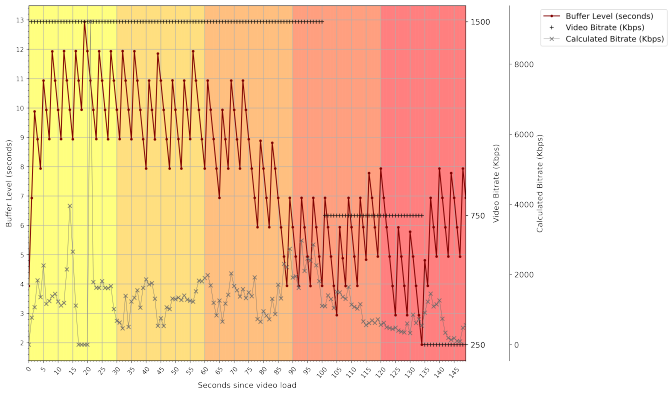

Fig. 1. Effect of bandwidth decrease

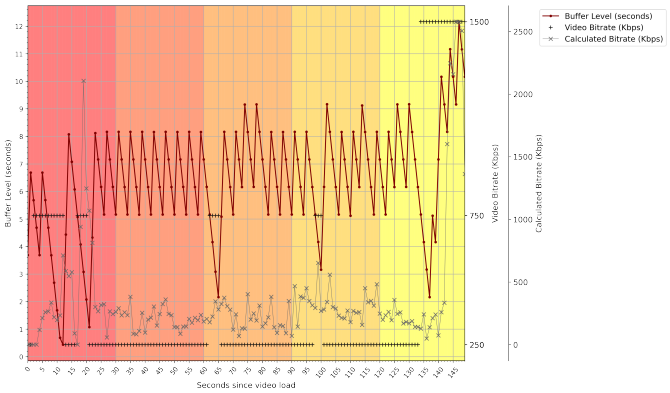

Fig. 2. Effect of bandwidth increase

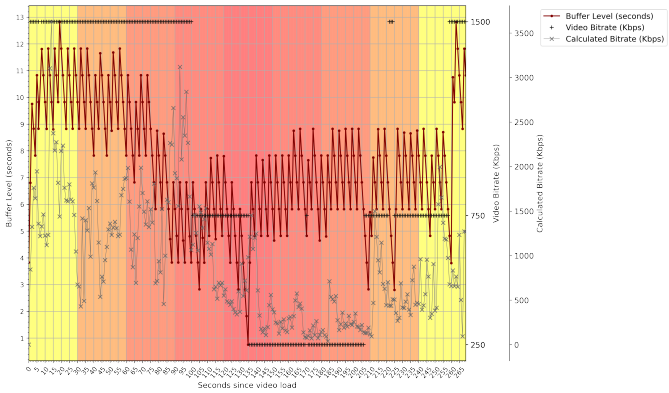

Fig. 3. Effect of decreasing then increasing bandwidth

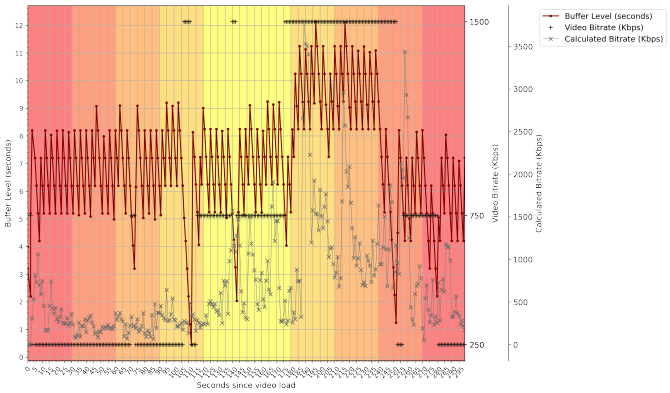

Fig. 4. Effect of Increasing then decreasing bandwidth

are expecting the player not only to optimize for stream stability on network bandwidth changes, but also on previously downloaded stream segments. This means that the player should not try to change the video bitrate too eagerly, but will try to keep the buffer as full as possible. The reader is also referred to the used dash.js player settings section for configuration specific predictions II-C2.

B. Analysis

The analysis relies on the color notation, so they deserve a short explanation. The filler colors in the figures depict the throttled bandwidth, where the more yellow colors depict higher bandwidths and redder depict lower bandwidths. In our case the fully yellow color depicts 5000 Kbps and the fully yellow depicts 1125 Kbps. We refer to the throttled bandwidth level and its corresponding color as a ”bandwidth sequence” due to the sequential variation of them in our experiments.

The figures have evenly spaced ”spikes” for the buffer level. These spikes could be correlated to the defined segment size of the video encoding. The segment size is 4 seconds and each spike measures the buffer level over 4 seconds, as can be seen by the dots on the downwards slope of each spike. As the player attempts to continuously meet the buffer target it will let the buffer empty up to a certain point before requesting a new segment (unless it has been able to request high bitrate segments, in which case it increases the buffer target to the top quality buffer target). Although the figures could have grouped each data point by 4 to smoothen out the graphs, this would have resulted in some crucial small drops being omitted, so we decided to leave them more granular.

The audio bitrate was not considered during the experiment runs as audio segments are not as heavily affected by network bandwidth as video segments [8].

1) Gradually decreased bandwidth: In this experiment we gradually decreased the bandwidth from 5000 Kbps, through the intermediate bandwidths, down to 1125 Kbps at 30 second intervals, totalling an aggregated stream time of 150 second. The result shown in Figure 1 was found.

We notice that the video bitrate of the stream starts off with the highest available, that is 5000 Kbps. The player is able to buffer at the top quality buffer target of 10 seconds for the two largest bandwidth sequences while also serving the highest video bitrate. When entering the middlemost bandwidth sequence, the player is gradually shifting to the normal buffer target of 6 seconds around the elapsed 75 second mark, while still serving the highest video bitrate due to the already buffered high bitrate video. Just as we enter the second lowest bandwidth sequence the player has equalized the buffering and is now in the normal buffer target.

During second lowest bandwidth sequence the player has

to account for the lower network bandwidth in its bitrate adaption. We can see that at the 100 seconds mark, the player switches from the highest available video bitrate to middlemost video bitrate of 750 Kbps. Around the switch we also notice an initial dip in the buffer target, which quickly rebounces back to higher levels. This would be explained by the player being unable to buffer high bitrate segments, and is thus attempting to keep the stream stable by switching. The re-bounce, however, is not stable after the lowest, and last, bandwidth sequence is entered.

In this last bandwidth sequence the player is still able to serve the buffered video bitrate quality of 750 Kbps as it was able to buffer that quality in the earlier bandwidth sequence. The player is though unable to buffer more of this quality and stays under the buffer target of 6 seconds between the 122 to 136 second marks. Here the player is dynamically able to account for this and switches to the lowest video bitrate of 250 Kbps at the 135 second mark. At this quality the player is able to attain the normal buffer target of 6 seconds for the rest of the experiment run.

Although a correlation between video bitrate and buffer levels can be noticed from the figures, the calculated bitrate does not provide as meaningful insights. The calculated bitrate is behaving rather sporadically, where, for example, at the 21 second mark it jumps to is highest measured value of above 8000 Kpbs, before decreasing down to more stable values.

2) Gradually increased bandwidth: In this experiment we gradually increased the bandwidth from 1125 Kbps through the intermediate bandwidth up to 5000 Kbps at 30 seconds interval. The result is shown in the Figure 2 was found. From the result, we can see that the player started by serving low bitrate videos. The player was loading for sometimes before it starts playing. Therefore it was able to fill the buffer. That is why the buffer level starts from 6 second. Since it had buffer from the start, it went for the higher bitrate video. Then, it started to serve medium bitrate, 750 Kbps. This happens because the buffer was empty and the player tries to fill it with the highest quality possible, but the bandwidth is not high enough to serve high bitrate video. Therefore, it requested for the medium bitrate video segments. With low internet speed provided, the player couldn't maintain filling up the buffer with middle quality video bitrate segments. Therefore, it filled the buffer with the lowest bitrate segment, 250 Kbps. This way the player tries to keep the buffer filled. Around the elapsed 60 second mark, the player sensed an increase in bandwidth and tried to get the higher bandwidth. Just like the previous bandwidth sequences, it couldn't maintain the buffer filled up. Another reason is the ABR throughput setting forced the player to low bitrate video as most of the past streams were of low bitrate video. Therefore it went again back to filling the buffer with the lowest bitrate video segment again.

At the 120 second mark, the internet bandwidth was increased to the highest level (5000 Kbps), but the player keeps playing the lowest bitrate video. This is because the lowest bitrate video was in the buffer. Right after the low quality bitrate segments were consumed by the player, at around 134 second mark, the buffer was empty and the bandwidth is still high. Therefore, the player chose fill the buffer with high bitrate video segment to the top quality buffer target of 10 seconds as found in dash setting. That is why there is a noticeable spike increase with buffer target at around 136 second mark.

From this, we can understand that every time an increase in the bandwidth occurs, the player requests video segments with high bitrate. The player then tries to fill the buffer with the highest bitrate possible. If the player is not able to fill the buffer with the highest bitrate video segments, and then the buffer will be emptied. This forces the player to request low quality videos and fill the buffer with low quality video segments, until the bandwidth is high enough to serve either medium or high quality videos.

In this experiment we notice a largest correlation between buffer levels and the calculated bitrate than in the gradually decreasing experiment. For the example, the large decrease in the in the buffer level starting at the 14 second mark correlates with a high increase in calculated bitrate. Similarly other jumps in the figure correspond to a certain degree with the calculated bitrate. Especially in the high buffer level increase in the highest bandwidth sequence the calculated bitrate corresponds heavily, which is reasonable as the player has a higher buffer target within this bandwidth sequence.

3) Gradually decreased, then increased bandwidth: In this experiment we measure the adaptive nature of our chosen player when available network bandwidth is gradually decreased from the highest throttled bandwidth of 5000 Kbps, through the intermediate bandwidths, down to 1125 Kbps, and then back up again, at 30 second intervals, totalling an aggregated stream time of 270 seconds. The result shown in Figure 3 was found.

During the decreasing part of the experiment we notice that the player is behaving almost identical to the decreasing bandwidth experiment in terms of buffer target and video bitrate. Thus we shift our focus to the player behaviour to the increase section of the experiment, starting from the bandwidth sequence change at the 150 second mark.

Although the available bandwidth has started to increase, the player is unable to increase the video bitrate in a similar fashion to the stepped bitrate levels in the decreasing bitrate section of the experiment. This could be explained by the ABR throughput rule and the sliding average method used by the player, where the last previous throughput history has been low. This theory is supported by the fact that the buffer is able to be filled pretty efficiently, which in case the player was configured to use the `abrBola` ABR rule it would more eagerly attempt to up-switch the video bitrate.

After continually meeting the normal buffer target in the middlemost bandwidth sequence, the player switches the video bitrate to 750 Kbps at the 207 second mark. A stark decrease in the buffer level is also observed, which the player is quickly able to buffer. This coincides with the entrance of the next, higher bandwidth sequence, yielding a rapid enough buffering rate for the player to attempt an up-switch in video bitrate to

1500 Kbps at the 222 second mark. This makes sense considering the player configuration, where the previously downloaded segments were quickly requested yielding a high average throughput. However, this rapid increase was unsustainable for the player within this bandwidth sequence, prompting it to down-switch the video bitrate back to 750 Kbps. Only halfway within the highest bandwidth sequence does the player request the highest video bitrate, which it is able to buffer until the end of the experiment run.

It is notable that the player in this experiment was quicker at meeting the normal buffer target, and able to keep a higher buffer level (on average), than the player in the gradually increasing experiment. Why this quicker rebound happened is not explainable within the confinements of our settings or hypothesis as the average throughput calculations do consider a large enough history.

As for the calculated bitrate we notice a larger correlation to the video bitrate. The calculated bitrate stays closer to neighbouring values when the player is serving lower video bitrates, e.g. in the timespan of 135 to 205 seconds. However, it is still behaving rather sporadically, so any meaningful correlations within higher video bitrate regions cannot be found.

4) Gradual increased, then decreased bandwidth: In this experiment we measure the adaptive nature of our chosen player when available network bandwidth is gradually increased from the lowest throttled bandwidth of 1125 Kbps, through the intermediate bandwidths, up to 5000 Kbps and then by decreasing back to the highest bandwidth at 30 second intervals, totalling an aggregated stream time of 270 seconds. The result shown in Figure 4 was found.

The player started off by buffering low bitrate video segments, meeting the buffer target almost instantaneously and staying there for most of the first three bandwidth sequences. This can be because of the need for the player to fill the buffer with available bitrate video segment and ABR throughput setup in dash player. Rather than increasing the quality, it keeps serving the low bitrate video by considering the recent throughput history. After entering the second highest bandwidth sequence, and sustaining an above normal target buffer level for some fragment durations, the player requested unsustainable high video bitrates at the 106 second mark. The player quickly emptied the buffer, presumably due to slow segment response times, making it down-switch back to the lowest quality video bitrate. However, the bandwidth was large enough to maintain uninterrupted streaming, so the player requested medium bitrate video. Although the highest bandwidth sequence was entered, the quality of the video remained at the medium bitrate. This is probably because of the ABR throughput strategy setting we are using. The ABR throughput chooses stream bitrate based on recent throughput history and the player is presumably applying that setting to experiment. This experiment also supports our hypothesis.

After this, the bandwidth keeps decreasing every 30 seconds. Even if the bandwidth is decreasing, the buffer started to be filled with the highest bitrate. The player then fills the buffer with high bitrate to the top quality buffer target of 10 seconds as found in player settings. Then for about 60 seconds, the player serves high bitrate video while maintaining full buffer length of 10 seconds. At 240 second mark, the decrease in bandwidth resulted the decrease in buffer level. The player keeps streaming high bitrate video until the buffer level drops significantly. At this point, the player fills the buffer with medium bitrate video segments. The decrease in bandwidth at 270 second mark resulted in another notable decrease in buffer level. Finally the player couldn't request anymore medium quality video on this bandwidth speed. Therefore, the player keeps filling the buffer with, and streaming low bitrate video segments.

Although the highest video bitrate was not achieved as early in the high bandwidth sequence as in the gradually decreasing experiment, the experiment shows a generally similar trend. The second highest and middlemost bandwidth sequences were able to support high video bitrates up until entering the second lower bandwidth sequence. This would be due to the already buffered high video bitrate. However, the drop to the normal buffer target level was steeper in this experiment.

Similarly to the gradually decreasing then increasing bandwidth experiment the calculated bitrate stays closer to neighbouring values within video bitrate regions. However, measure values are staying lower within high video bitrate regions and high bandwidth sequences than in the other experiment. No proper correlation or insights on the calculated bitrate can be drawn here.

IV. Limitations & Discussion

Although the audio segments do not affect the stream quality as notably as video segments do, they will still have an effect on the network throughput. This effect might not be significant, but the audio segments could have been disabled altogether to eliminate a possible source of interference. We did not investigate how or if the dash.js player accounts for audio segments when calculating throughput, which could also have been a source of it couldn't interference.

We did not consider the effect of our HTTP server choice, nor did we try to tweak any settings for the server. The default Apache HTTP settings do not have any caching enabled, and the server supports MPEG-DASH out-of-the-box. No further investigation into the HTTP servers were done, although further experiments could take server choice into account.

To limit the scope of these experiments we did not investigate the effect of different player buffer targets or the fragment size of video encoding. However, these settings would certainly have an effect on the adaptive bitrate behaviour of the player, such as more stable high video bitrate streaming during decreased network bandwidths. An interesting experiment that could be conducted on the basis of this paper would be if the dash.js player always tries to optimize for high buffer levels, or whether it could decide to rather keep it in a an intermediate state with higher video bitrate.

Our metrics gathering tools supported, and were configured with, the measurement of video segment quality per request.

These data could have aided the experiment analysis by provided insight into whether the multimedia player requested multiple video bitrates concurrently, or if it dropped requests due to rebuffering events. A cursory examination of the gathered data suggests that request duplication is taking place, yet the significance is inconclusive. This was omitted for brevity of the paper, but could yield interesting results and analysis. Furthermore, the downloaded quality metrics could have provided insights into the often sporadica behaviour of the calculated bitrates.

V. Conclusion

In this paper, we used FFmpeg to segment a video into 4 second long segments to be served over HTTP to a multimedia player. The HTTP server was setup to serve the segments of different bitrates based on the bandwidth. After that different experiments were conducted on the served stream. We analyzed four scenarios, when there is a gradual increase, gradual decrease, gradual increase then gradual decrease, and gradual decrease then gradual increase. The analyzation was aided by gathered metrics on each stream run, which were plotted as time-series figures.

From the experiments, we could understand that the dash.js player, with moving average method and ABR throughput settings, tries to keep its buffer level full with suitable quality for the given bandwidth rather than jumping to a new quality video every time there is an increase in bandwidth. The player down-switches video bitrate more eagerly when the network bandwidth is decreasing, and more diligently stays at lower video bitrates during lower network bandwidth segments. This corresponds to our hypothesis on the eagerness of player video bitrate switching, is aligned with the settings of the player. During high bandwidth sequences the player is able to fill the buffer at top quality buffer targets from which it is able to serve higher quality bitrate videos when entering lower quality bandwidth sequences. During low bandwidth sequences the player is using the configured normal buffer target, and thus has to experience longer periods of higher bandwidth before up-switching the quality.

It seems that the dash.js player, although configured with the moving average method and ABR throughput rules, also heavily relies on the buffer level to decide the video bitrate. The ABR throughput rule documentation[11] mentions that the rule uses ”recent throughput history to predict the future throughput”, but no specific length for the history. If the history is larger than the moving average methods 4 fragments setting, then this could be an explanation for the lower video bitrates in, e.g., the highest bandwidth sequence in the gradually increasing the decreasing experiment.

All in all, we noticed that the dash.js multimedia player is able to provide a consistent and stable streaming experience to users even in fluctuation network bandwidths, and with buffer sizes close to the video segment sizes. The hypothesis of our experiment was met, but with the caveat that buffer levels of the player is also heavily considered during video bitrate choice. The calculated network bitrate of each experiment did not provide enough insight into the video bitrate choices by the player. Some correlation could be noted of the calculated bitrate values to the video bitrates, but whether or not the calculated bitrate affected (i.e. is considered in the internals of) the player's video bitrate choices is inconclusive.

References

[1] URL: https://hub.docker.com/_/httpd/.

[2] URL: https://bitmovin.com/mpeg-dash-browser-support-device-compatibility/.

[3] URL: https://developer.mozilla.org/en-US/docs/Web/Media/DASH_Adaptive_Streaming_for_HTML_5_Video.

[4] URL: https://github.com/Dash-Industry-Forum/dash.js/wiki.

[5] URL: https://github.com/Dash-Industry-Forum/dash.js/releases/tag/v3.1.3.

[6] URL: https://reference.dashif.org/dash.js/latest/samples/index.html.

[7] URL: https://version.aalto.fi/gitlab/neergaa1/multimedia-adaptive-streaming/-/blob/master/FFmpegCommand.

[8] URL: https://www.brendanlong.com/the-structure-of-an-mpeg-dash-mpd.html.

[9] URL: http://cdn.dashjs.org/latest/jsdoc/module-Settings.html.

[10] URL: http://cdn.dashjs.org/latest/jsdoc/module-Settings.html#~AbrSettings__anchor.

[11] URL: https://github.com/Dash-Industry-Forum/dash.js/wiki/ABR-Logic.

[12] URL: http://cdn.dashjs.org/latest/jsdoc/module-DashMetrics.html.

[13] URL: http://reference.dashif.org/dash.js/latest/samples/advanced/monitoring.html.

[14] URL: http://cdn.dashjs.org/latest/jsdoc/module-Settings.html#~PlayerSettings__anchor.

[15] URL: https://version.aalto.fi/gitlab/neergaa1/multimedia-adaptive-streaming.

[16] The Apache Software Foundation. URL: http://httpd.apache.org/.