A Experiment over Media Player Application based on Dash.js

Dinesh Kini Bailoor
40231799

Sindoora Sudhakar Rao
40199155

Abstract

A video streaming programme called DASH offers on-demand video playback. It has a web server that runs on HTTP. The streamed video is divided into equal-length chunks. These segment objects are obtained by the client each time an HTTP request for a clip is made. Every video has an own Media Presentation Description (MPD) file, which the client uses to learn about the portions that make up a specific video.In this experiment, we employ a media player approach that relies on dash.js (v4.5.0), which adaptively transitions and plays media streamlets while playing a stored DASH playlist or the mpd file that we want.The effectiveness of the built-in Adaptive BitRate (ABR) algorithms is then tested and compared using a variety of experiments. Some of the algorithms provided are throughput-based, dynamic, and BOLA.By keeping track of key parameters like bit rate, buffer level, throughput, and segment properties like length and time, the efficacy is calculated.

1 INTRODUCTION

The experiment focuses on the three ABR algorithms such as throughput-based,dynamic and Bola algorithm. We have used the testplayer.html page in the dash player to conduct this experiment. The three algorithms are checked in their corresponding html files.All in all we have three testplayer.html files defined and one of the three algorithms is enabled in one of the html files.We then record the network metrics for all the three algorithms by switching the network conditions between Fast3G and no throttling.

1.1 ABR Algorithms

ABR gives a quicker starting time along with smoother feeds. The video can start playing right away because ABR streams often start with a low bitrate stream before the selection algorithm determines the available bandwidth or buffering space. As HTTP is used for delivery, most web servers and CDNs are compatible with HTTP-based ABR streaming technologies. It is therefore more affordable than putting up customized services or maintaining constant connectivity. In Throughput-based ABR, streaming related decision is performed based on the throughput details from the previous downloads of the application.BOLA ABR algorithm uses the details of buffer level at the client side for decision making.While downloading a segment, BOLA utilises a bitrate selection mechanism that converts the buffer level—the overall amount of seconds of video segments saved in the buffer—to the bitrate of the following segment.Dynamic ABR algorithms uses both the factors mentioned above.

1.2 Performance Metrices

1.2.1 Selected BitRate

Bit rate is the measure of how much information is sent through a communication connection in a specific length of time. The term ”bit rate” is frequently used to characterise the transmission quality of music and video in the context of streaming. While requiring more bandwidth and storage space, higher bit rates typically provide audio and video of greater quality. In conclusion, bit rate is a term used to define the quantity of data delivered per unit of time and the quality of streams.Here we are calculating bitrate in Mbps.

1.2.2 Buffer Level

A specific amount of data is put into the buffer when a video file is started before playback starts. More data is stored into the buffer as playback continues in order to keep the buffer level constant. The video can delay or stutter when the player tries to load additional data into the buffer if the buffer level starts to drop.The quantity of video data that is temporarily kept on a user's device during a video streaming session before being seen on the screen is referred to as buffer level. In situations where the network connection may not be quick enough or reliable enough to provide the video data in real-time, this buffer is utilised to make sure that the video playing continues smoothly and without interruption.

1.2.3 Measured Throughput

The pace at which data is delivered across a network connection while a video is being played back is referred to as measured throughput in video streaming. It is a measurement of the real network transmission capacity for streaming video, accounting for network performance-altering variables such as congestion.The player may occasionally send queries to the streaming server for brief bursts of information (referred to as segments) in order to track throughput during video streaming. The quantity of information received divided by the amount of time required is used to compute throughput. Next, the duration required for each segment to download is noted.

1.2.4 Segment Download Time

The length of the period required for a chunk of video data to download from of the streaming server to the user's browser is

referred to as the segment download time in streaming video. A segment is a brief section of the video file that is downloaded independently from the remainder of the movie. With segmented video downloads, the video player can continue playing the clip well before complete file has finished downloading.

1.2.5 Segment Size

In streaming video, the term ”segment size” describes how much data is contained in each segment. The quality and bit rate of the clip, the speed of the network connection, and the capacities of the client browser are only a few of the variables that might affect the segment size. Larger segments often contribute to speedier playing since receiving and analyzing each segment take less time. But, lengthier download periods brought on by bigger chunks may also increase the chance of stuttering or other problems during playing.

2 METHODOLOGY

2.1 Metrics Collection

In the main.js file of the testplayer under low latency, we modified the METRIC_INTERVAL attribute from 1000 to 8000 milliseconds so that the setInterval method iterated every 8 seconds.The BitRate and Buffer is retrieved using the metrics defined the current testplayer page elements. The metrics are returned in html tags and in order to get the required value, we extract the inner value of the tag.The measured throughput is a result returned by getAverageThroughput method defined under the player object held by the current test player object.The segment attributes are retrieved using the HTTP object. To fetch the current HTTP object we have utilised, getCurrentHttpRequest method defined under dashMetrices. The absolute difference between finish time and the response time would give us the total segment time. The _responseHeader of HTTP object inculcates various information including the content length too. This content length is nothing but the segment size that we are interested in. The metrics that we read is stored in respective array. After the entire video is played, we pu the results into single array and then stored as BLOB.

```
//Bit Rate
console.log('Bitrate:', +self.domElements.metrics.videoBitrate.innerHTML);
bitRateArray.push('Bitrate: ' + self.domElements.metrics.videoBitrate.innerHTML + '\r\n');

//Buffer
console.log('Current Buffer:',self.domElements.metrics.bufferTag.innerHTML);
bufferArray.push('Current Buffer: ' + self.domElements.metrics.bufferTag.innerHTML + '\r\n');
```

Figure 1: Bit Rate and Buffer retrieval.

```
//Throughput
let throughput = self.player.getAverageThroughput('video');
console.log('Measured throughput:',throughput);
throughputeArray.push('Measured throughput: ' + throughput + '\r\n');
```

Figure 2: Measured Throughput retrieval.

```
//Download Time
const httpObject = dashMetrics.getCurrentHttpRequest('video');
const segmentTime = Math.abs(httpObject._tfinish.getTime() - httpObject.tresponse.getTime());
console.log('Segment download time',segmentTime + 'ms');
downloadArray.push('Segment download time: ' + segmentTime + 'ms' + '\r\n');

//Segment Size
console.log('Segment size',httpObject._responseHeaders.split('\r\n')[1])
segmentSizeArray.push('Segment size: ' + httpObject._responseHeaders.split('\r\n')[1] + '\r\n');
```

Figure 3: Segment time and length calculation.