

# SERVER AND CLIENT-SIDE ALGORITHMS FOR ENHANCING ADAPTIVE STREAMING

A Dissertation

by

Mehmet Necmettin Akçay

Submitted to the  
Graduate School of Sciences and Engineering  
In Partial Fulfillment of the Requirements for  
the Degree of

Doctor of Philosophy

in the  
Department of Computer Science

Özyeğin University  
August 2022

Copyright © 2022 by Mehmet Necmettin Akçay

# SERVER AND CLIENT-SIDE ALGORITHMS FOR ENHANCING ADAPTIVE STREAMING

Approved by:

---

Associate Professor Ali Cengiz  
Beğen, Advisor  
Department of Computer Science  
*Özyeğin University*

---

Professor M. Reha Civanlar  
Department of Computer Science  
*Özyeğin University*

---

Associate Professor Müge Sayıt  
International Computer Institute  
*Ege University*

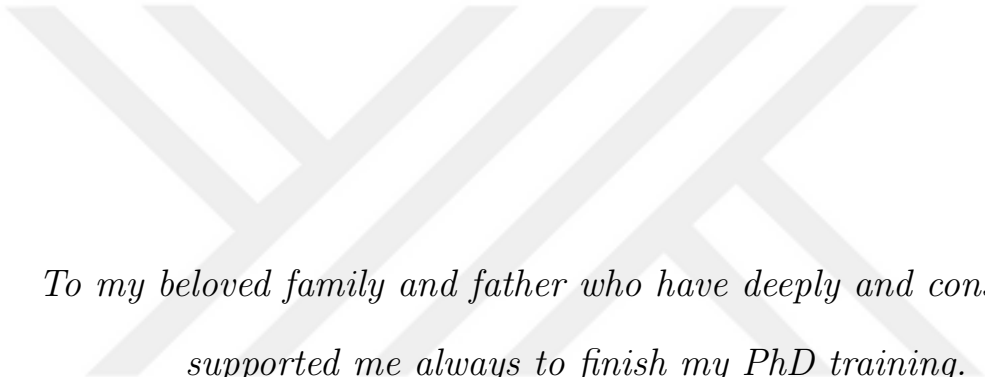
---

Assistant Professor İsmail Arı  
Department of Computer Science  
*Özyeğin University*

Date Approved: 11 August 2022

---

Assistant Professor Tankut Akgül  
Electronics and Communication  
Engineering  
*Istanbul Technical University*



*To my beloved family and father who have deeply and consistently  
supported me always to finish my PhD training.*

# ABSTRACT

HTTP adaptive video streaming is a technique widely used on the internet today to stream live and on-demand content. Server and client-side algorithms play an important role in improving user experience in terms of metrics such as latency, rebufferings and rendering quality. After explaining the commonly used metrics, we analyzed four main aspects of video streaming (*i*) bandwidth prediction accuracy, (*ii*) utilization of playback speed, (*iii*) adaptive streaming for content-aware-encoded videos, and (*iv*) head motion awareness for 360-degree videos. 360-degree video streaming requires much higher bandwidth compared to conventional video streaming. We demonstrate that most of the algorithmic improvements achieved for video streaming can also be applied to Viewport Dependent Streaming (VDS) for 360-degree videos. It is also important that in 360-degree video streaming, we have a Head Mounted Display (HMD) device that is capable of pointing the viewport orientation of the user. We also investigate and improve the rate-adaptation algorithms for 360-degree videos by developing several new algorithms making use of the HMD. The new algorithms proposed in this thesis are Low-on-Latency (LoL), Low-on-Latency<sup>+</sup> (LoL<sup>+</sup>), Bang-on-Bandwidth (BoB), Size-aware Rate Adaptation (SARA), Content-aware Playback Speed Control (CAPSC), Head-motion-aware Viewport Margins (HMAVM). We evaluate the proposed new algorithms using the objective metrics discussed in detail and show significant contributions for these new algorithms including up to 91% decrease in rebuffering duration for on-demand streaming, 61.9% decrease in rebuffering duration and 8.1% decrease in latency compared to L2A for low-latency live streaming, 81.3% bandwidth prediction accuracy for interactive streaming, lastly 20% improvement in viewport quality and 50% reduction in motion-to-high-quality delay for 360-degree video streaming.

## ÖZETÇE

HAS (HTTP adaptive streaming) günümüzde internet üzerinde canlı ve isteğe bağlı içerikleri yayınlamak için oldukça yaygın kullanılan bir tekniktir. Sunucu ve istemci tarafında geliştirilen algoritmalar kullanıcı deneyimini gecikme, donma ve seyir kalitesi metriklerine göre daha iyileştirmek için önemli bir rol oynamaktadır. Genel olarak kullanılan metrikleri açıkladıktan sonra, video yayını dört ana kategori altında analiz ettik (i) bant genişliği tahmini doğruluğu, (ii) oynatma hızının ayarlanması, (iii) içeriğe duyarlı kodlanmış videolar için uyarlanabilir yayın, ve (iv) 360-derece videolar için kafa hareketi duyarlılığı. 360-derece video yayını, geleneksel yayınlara göre çok daha fazla bant genişliğine ihtiyaç duymaktadır. Bu çalışmada video yayınları için geçerli olan çoğu algoritma iyileştirmelerinin 360-derece VDS (Viewport Dependent Streaming) yayınlarında da kullanılabileceğini gösteriyoruz. 360-derece video yayınında HMD (Head Mounted Display) olarak bilinen bir aparat ile kullanıcının nereye baktığının bilgisini tespit etmek mümkün olmaktadır. Ayrıca, HMD cihazını kullanan yeni algoritmalar geliştirmek suretiyle 360-derece videolar için hız uyarlama algoritmalarını inceliyor ve iyileştiriyoruz. Bu tezde geliştirdiğimiz yeni algoritmalar şu şekildedir; Low-on-Latency (LoL), Low-on-Latency<sup>+</sup> (LoL<sup>+</sup>), Bang-on-Bandwidth (BoB), Size-Aware Rate Adaptation (SARA), Content-Aware Playback Speed Control (CAPSC), Head-Motion-Aware Viewport Margins (HMAVM). Geliştirilen bu yeni algoritmaları detaylıca açıklanan objektif metrikleri kullanarak değerlendiriyoruz ve bu yeni algoritmaların, isteğe bağlı akış için donma süresinde %91'e varan azalma, düşük gecikmeli canlı yayın için L2A algoritmasına kıyasla donma süresinde %61.9 azalma ve gecikme süresinde %8.1'e varan azalma, interaktif yayında %81.3 bant genişliği tahmin doğruluğu, son olarak 360-derece video akışında görüntü alanı kalitesinde %20 iyileşme ve hareketten yüksek kaliteye gecikme süresinde %50 azalma dahil olmak üzere kayda değer katkılarını gösteriyoruz.

## ACKNOWLEDGEMENTS

I would like to acknowledge all the colleagues who have supported me during my PhD career, especially my advisor, Associate Prof. Ali C. Beğen and all of the thesis committee members Prof. M. Reha Civanlar, Assistant Prof. İsmail Arı, Assistant Prof. Tankut Akgül and Associate Prof. Müge Sayıt. I would like to thank Prof. Roger Zimmermann, Dr. Abdelhak Bentalab and May Lim with whom I have worked hard during my PhD study. I also would like to acknowledge Saba Ahsan, Igor D.D. Curcio and Emre Aksu with their valuable contributions and guidance throughout this study.

# TABLE OF CONTENTS

<b>DEDICATION</b>	<b>iii</b>
<b>ABSTRACT</b>	<b>iv</b>
<b>ÖZETÇE</b>	<b>v</b>
<b>ACKNOWLEDGEMENTS</b>	<b>vi</b>
<b>LIST OF TABLES</b>	<b>ix</b>
<b>LIST OF FIGURES</b>	<b>x</b>
<b>I INTRODUCTION</b>	<b>1</b>
1.1 HTTP Adaptive Streaming	1
1.2 Extensions for 360-Degree Videos	4
1.3 Measuring Streaming Performance and User Experience	6
1.3.1 General Performance Metrics	6
1.3.2 360-Video-Specific Metrics	7
<b>II ALGORITHMS FOR ON-DEMAND AND LIVE STREAMING</b>	<b>10</b>
2.1 Streaming of Content-Aware-Encoded Videos	10
2.1.1 The SARA Algorithm	11
2.1.2 Evaluation	14
2.2 Algorithms Using Common Media Client Data/Server Data	17
2.2.1 Common Media Client Data (CMCD)	17
2.2.2 Common Media Server Data (CMSD)	21
<b>III ALGORITHMS FOR LOW-LATENCY LIVE STREAMING</b>	<b>23</b>
3.1 Key Enablers for Low Latency	23
3.1.1 Common Media Application Format (CMAF)	23
3.1.2 Chunked Transfer Encoding (CTE)	24
3.2 The LoL and LoL <sup>+</sup> Algorithms	25
3.2.1 Weight Selection	29
3.2.2 Throughput Measurement	31
3.2.3 Playback Speed Controller	32

3.2.4	Evaluation . . . . .	34
3.3	Content-Aware Playback Speed Control . . . . .	36
<b>IV</b>	<b>ALGORITHMS FOR INTERACTIVE STREAMING . . . . .</b>	<b>41</b>
4.1	Bandwidth Prediction for WebRTC . . . . .	41
4.2	The BoB Algorithm . . . . .	42
4.3	Evaluation . . . . .	46
<b>V</b>	<b>ALGORITHMS FOR 360-DEGREE VIDEO STREAMING . .</b>	<b>52</b>
5.1	Viewport-Dependent Streaming with Concurrent Segment Down-loads . . . . .	53
5.2	HTTP/1.1 vs. H2 . . . . .	54
5.3	Head-Motion-Awareness for 360-Degree Videos . . . . .	55
5.3.1	Margin-Aware ABR . . . . .	56
5.3.2	Negative and Complementary Margins . . . . .	63
<b>VI</b>	<b>CONCLUSION AND FUTURE WORK . . . . .</b>	<b>76</b>
	<b>REFERENCES . . . . .</b>	<b>79</b>
	<b>VITA . . . . .</b>	<b>87</b>



## LIST OF TABLES

1	Notations of the QoE model [0]. . . . .	8
2	Comparison of rate-based, dynamic, SARA-Basic, SARA-RLS and SARA-MPC ABRs. . . . .	16
3	Previously defined and the new added CMCD parameters [0]. . . .	17
4	On-demand video streaming with 10 clients [0]. . . . .	20
5	On-demand video streaming with 20 and 30 clients [0]. . . . .	21
6	Results for VoD streaming for 10 and 20 clients using CMSD [0]. . .	22
7	Average scores used to calculate QoE for all ABRs. . . . .	37
8	List of the used notations. . . . .	38
9	List of the key symbols and notations used in BoB. . . . .	43
10	Average simulation results for different network profiles ( $\uparrow$ : higher is better, $\downarrow$ : lower is better). . . . .	50

## LIST OF FIGURES

1	HTTP adaptive streaming. . . . .	1
2	HTTP adaptive streaming using dash.js. . . . .	2
3	Bandwidth measurement in source limited scenario [0]. . . . .	3
4	ERP projection [5]. . . . .	5
5	Cubemap projection [5]. . . . .	5
6	Viewport quality calculation based on tile intersection with the viewport. . . . .	8
7	SARA architecture [0]. . . . .	12
8	SARA ABR flowchart [0]. . . . .	12
9	Bandwidth prediction implementations using RLS and MPC. . . . .	14
10	Encoded content bitrate used in tests. . . . .	15
11	Network profiles used in tests. . . . .	15
12	CMAF chunks for low-latency [7]. . . . .	24
13	Example SOM feature map setup for three bitrate alternatives. . . . .	27
14	Results of LoL <sup>+</sup> , LoL, L2A, STALLION, Dynamic ABRs for 11 different network profiles [0]. . . . .	36
15	Possible CAPSC implementation workflows for low-latency scenario [0].	38
16	Event densities for different test sequences [0]. . . . .	39
17	Results for Default ABR, LoL <sup>+</sup> and CAPSC [0]. . . . .	40
18	Overall workflow of BoB. . . . .	43
19	Learning-based (DRL) rate controller for BoB. . . . .	46
20	The network profiles used in the simulations. . . . .	48
21	Actual and predicted bandwidth for different network profiles. . . . .	51
22	Bandwidth measurement for concurrent dash threads. . . . .	54
23	H2 architecture. . . . .	55
24	Haversine (circular) distance. . . . .	59
25	Angular distance between a tile (red) and head motion vector. . . . .	59
26	Automatic generated head motions used in experiments. . . . .	60
27	Average viewport quality at 40 (top) and 60 Mbps (bottom) band- width. Lower values indicate better quality. . . . .	61

28	Average MTHQD results for the <i>Trolley</i> (top) and <i>Harbor Biking</i> (bottom) sequences at 40 Mbps bandwidth and tile configurations of 6x4 (left), 8x6 (middle) and 12x8 (right). . . . .	62
29	Average MTHQD results for the <i>Trolley</i> (top) and <i>Harbor Biking</i> (bottom) sequences at 60 Mbps bandwidth and tile configurations of 6x4 (left), 8x6 (middle) and 12x8 (right). . . . .	63
30	Proposed HMAVM margins. . . . .	64
31	Average viewport quality with HTTP/1.1 for Harbor. . . . .	69
32	Average viewport quality with HTTP/1.1 for Trolley. . . . .	70
33	Average MTHQD with HTTP/1.1 for Harbor. . . . .	71
34	Average MTHQD with HTTP/1.1 for Trolley. . . . .	72
35	Average MTHQD with H2 for Harbor. . . . .	73
36	Average MTHQD for 40 Mbps (top) and 60 Mbps(bottom) on artificial head motions with tile configurations of 6x4 (left), 8x6 (middle) and 12x8 (right) for the <i>Trolley</i> and <i>Harbor</i> sequences with H2. . .	74
37	Average viewport quality (left) and MTHQD (right) results for 40 Mbps(top) and 60 Mbps(bottom) with H2. . . . .	75

# CHAPTER I

## INTRODUCTION

### *1.1 HTTP Adaptive Streaming*

The importance and need for video streaming services are increasing in our daily lives everyday. The internet video traffic has been forecasted to be more than 82% globally for 2021 [9]. As the demand increases the technologies and methods used in video streaming continue to evolve as well. HTTP Adaptive Streaming (HAS) is one of the most common ways of delivering video content nowadays which gives the opportunity of transferring video in smaller pieces of data over the HTTP protocol. HAS has become quite popular since it depends on HTTP and standard TCP protocol, where a simple HTTP server can be easily used for HTTP based adaptive streaming. An example illustration of how HTTP Adaptive Streaming can be used is shown in Figure 1. As it can be seen in the figure, HAS has a wide range of applications today including mobile terminals and 360-degree video HMD devices.

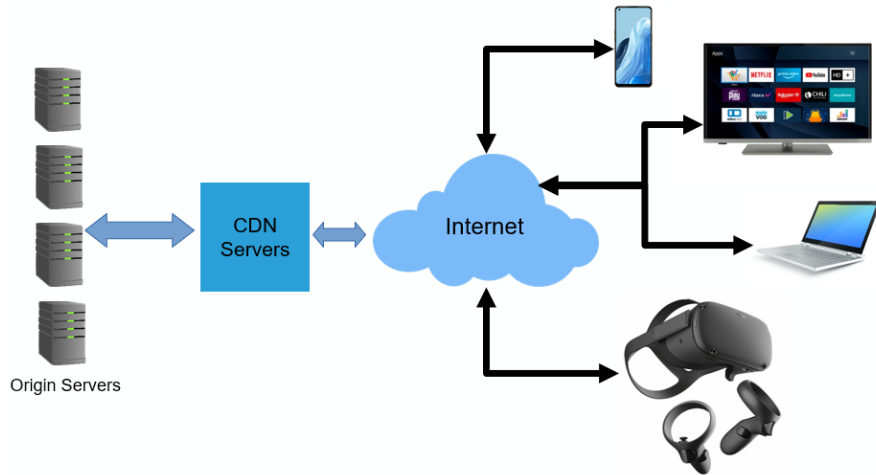


Figure 1: HTTP adaptive streaming.

Since HAS is the architecture used to transfer video content from origin servers

to clients, and the popularity of video content keeps increasing, similarly the applications of HAS are also increasing. And with that increase even in the same home network there might be resource race conditions where a TV can cause a delay in mobile streaming or vice versa. So, the need of improvements in Adaptive Bitrate (ABR) rules is also increasing. In this thesis, we will explain the problems in the ABR algorithms today and will address them with several enhanced algorithms. HAS can be implemented in several ways and one of the most popular implementations for video streaming is dash.js [10] which is shown in Figure 2.

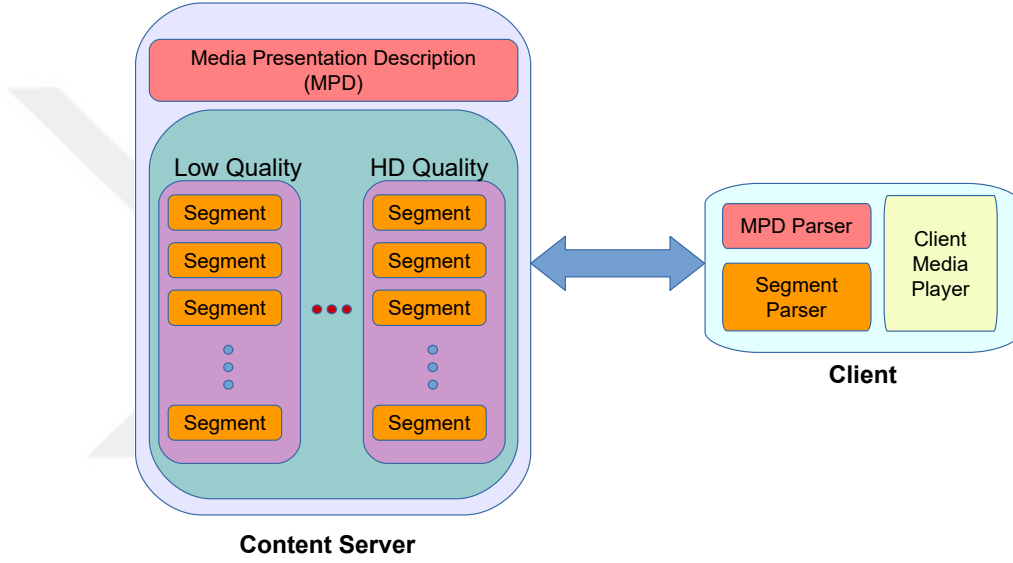


Figure 2: HTTP adaptive streaming using dash.js.

The main idea in this approach is to keep multiple encoded data for the same content with different qualities and let the client choose the appropriate representation based on Adaptive Bitrate (ABR) rules. In order to do that, the server hosts a media presentation description (MPD) file which tells the streaming client about the representations hosted by the server. Then the client makes a decision inside the ABR and switches to the desired quality representation based on observed parameters like buffer, latency, stall period, etc.

Although the latest improvements enable us to stream both conventional and 360-degree video content, there are some problems that need to be addressed to increase the user experience throughout the streaming session. Those problems

can be grouped into four main categories which are (i) Bandwidth Prediction Accuracy, (ii) Utilization of Playback Speed, (iii) Adaptive streaming for content-aware-encoded videos, and (iv) head motion awareness for 360-degree videos.

Bandwidth Prediction Accuracy plays a very important role, especially in low-latency. For the scenario where the content in the server is not ready yet and the content is being delivered to the client while it is being generated, in this case, even if we might have a higher bandwidth resource we will not be able to use that resource since the content is not ready yet and we are source limited. And if we calculate the bandwidth resource we have in the streaming client by simply using downloaded bytes divided by the time passed, we will end up with a value which is very close to the encoding bitrate. The reason for that is the counted idle times shown in Figure 3. That problem has to be addressed in order to allow the clients that are streaming low quality video even though the network resources are enough to allow higher quality streaming. Predicting bandwidth

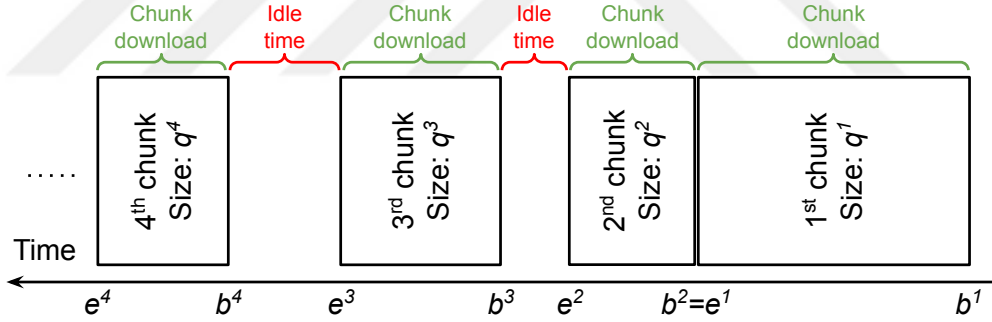


Figure 3: Bandwidth measurement in source limited scenario [0].

accurately is also an existing problem in the popular WebRTC streaming clients. While the client predicts the bandwidth with the idle times counted, the clients suffer from low quality or blurry video content. Considering the 360-degree video streaming scenario, bandwidth prediction is still an issue due to multiple parallel downloads for each tile. When the bandwidth prediction is not accurate, then the ABR decisions for the streaming session might be affected and the overall user experience can suffer from stalls or longer latency values. These problems in adaptive streaming are addressed with several enhanced algorithms introduced

inside this study.

Modern ABRs observe the current state of the video sessions and make a decision for the video streaming quality. Considering the client players we have today, there exists another area to work on and utilize during the video streaming session which is the playback speed of the video. Depending on the content and the observed state of the player, it is possible to create an adaptive playback speed algorithm to increase the user experience of the overall video in order to reduce the rebufferings or catchup the latency. We have developed several playback controller algorithms and evaluated them inside this thesis.

Another problem that needs to be addressed in video streaming is, the adaptation of the video playback sessions with varying segment sizes for the video representation. VBR is an efficient way of encoding the video content since it allows to use more bitrate for complex scenes in the video where the stale scenes can be represented with less data. Using VBR encoding will preserve the encoding quality since for easy scenarios less data is required. On the contrary, for a complex moving scenario, more data will be required to keep the same quality. However, for ease of adaptation, it is also common to use Constant Bitrate (CBR) encoding for the content generation. CBR encoding will have varying qualities throughout the playback session by fixing the encoded bitrate to a specific value which makes it easier for the client ABR to adapt since all segments will be very close in size. When we have a variable bitrate (VBR) encoded content, traditional algorithms may predict the next segment inaccurately and that can lead to extra stalls or poor user experience in these cases. In this study, we have developed an algorithm to solve this problem to adapt better to the changing conditions.

## ***1.2 Extensions for 360-Degree Videos***

By definition HAS has no restriction on the content side as long as it can be served as smaller segments of data. Thus the same method is applied to both conventional and 360-degree video streaming. For 360-degree video streaming,

Omnidirectional Media Format (OMAF [11]) was introduced in order to represent 360-degree videos in a standard way for video streaming, and an overview of the overall streaming experience using dash is discussed in [5]. According to OMAF, in order to create the content for streaming, 360-degree video is projected onto 2D coordinate space and served by the media server. There are mainly two different kinds of approaches for the projection according to the OMAF standard which are equirectangular projection (ERP) and cubemap projection (CMP). ERP is shown in Figure 4, which looks very similar to the world map generation as we know it today with only a small difference with horizontal axis coordinates. Since we assume that the projection is done from the center of the sphere the horizontal directions will be inverse of a standard world map as we use today.

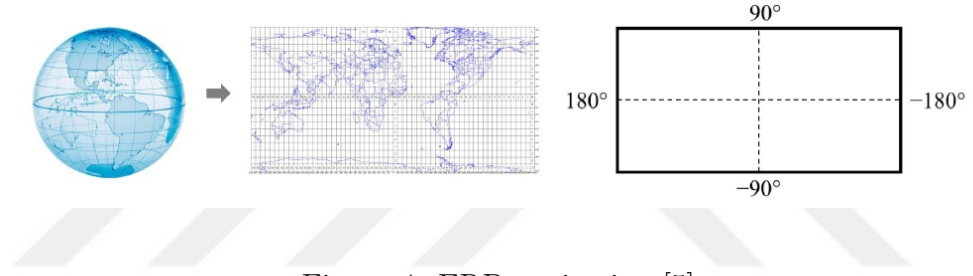


Figure 4: ERP projection [5].

CMP projection as shown in Figure 5 similarly assumes we are in the center of the sphere and projects all six squares for each side of a cube.

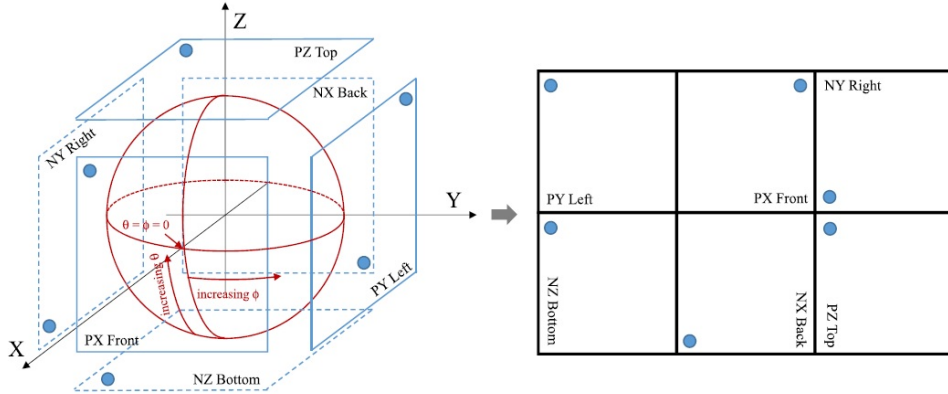


Figure 5: Cubemap projection [5].

After projecting 360-degree video content onto 2D space, it is possible to act the same way we do in conventional streaming for both of the projection methods



by using the required encoding/decoding in the server and client side. Considering required bandwidth resources, conventional streaming is more straightforward nowadays since it is possible to reach HD quality even with a simple connection with a total bandwidth of 2 Mbps. However for 360-degree video streaming, required bandwidth is much higher compared to conventional streaming. So, in order to overcome this issue Viewport Dependent Streaming (VDS [12]) is a commonly used technique. The idea behind VDS is to serve the viewport tiles in high quality where the user is looking in the 3D space and the rest in possibly lower quality in order to save unnecessary bandwidth resource consumption. And in order to do that whole content is divided into smaller pieces called tiles that can be requested in different qualities by the client ABR. Considering all these improvements in 360-degree video streaming, it has been possible to use the same video streaming architecture for both conventional and 360-degree video streaming.

Lastly, in this study head motion awareness is studied which is an additional information we have especially in 360-degree video streaming. Current implementations fail to consider the head motion data coming from the head mounted displays (HMD) in the ABR algorithms. Assuming a user is turning its head real fast, it might end up seeing some low quality tiles before new high quality tiles are downloaded. Because in VDS current viewport, where the user is looking is downloaded in high quality and the rest is downloaded in lower quality. Thus a quick head motion might end up with low quality tiles before the system reacts to that immediate viewport change. In order to solve this problem, we have proposed margin-aware ABR and negative margins concepts which will be explained in detail in the following sections.

### ***1.3 Measuring Streaming Performance and User Experience***

#### **1.3.1 General Performance Metrics**

Evaluating video streaming experience is itself a wide area that has been visited by many researchers recently. The evaluation of an ABR algorithm can be done

with subjective user tests where the content with the enhanced system is presented to users and their experience is collected at the end of the test to convey a result. Another approach that can also support subjective tests is the objective evaluations. Objective evaluations generally have a mathematical representation and evaluate a total score based on the measured metrics. The most recent QoE methods and the idea behind those evaluations are explained in [13].

In our study, we have used a modified QoE model based on the model Yi *et al.* [14] proposed in their work which we think is more suitable for low latency requirements. The model basically considers those 5 metrics bitrate selected by the ABR, total number of bitrate switches, total rebuffering time, live latency and client playback speed. The reasoning behind that is, we have experienced and know that it is possible to create an ABR algorithm that can achieve the lowest possible latency for streaming, but that comes with a cost of less client buffer and increased rebuffering times which can lead to a worse user experience. That is why each metric has a special factor and a weight in the total score. This can be formulated as below which we have published in our paper in [0]. In the equation, it can be seen that the bitrate selected by the ABR is a positive factor whereas the others (total number of bitrate switches, total rebuffering time, live latency and client playback speed) are negative parts of the equation which are working as a punishment that are assumed to decrease the user experience throughout the streaming session.

$$\text{QoE} = \sum_{s=1}^S (\alpha R_s - \beta E_s - \gamma L_s - \sigma |1 - P_s|) - \sum_{s=1}^{S-1} \mu |R_{s+1} - R_s|, \quad (1)$$

The explanations of the symbols used in Equation 1 are shown in Table 1.

### 1.3.2 360-Video-Specific Metrics

For 360-degree video, we have different metrics regarding the viewport dependent streaming (VDS). Since the idea behind VDS is to download the tiles, that the user is viewing, in high quality and the rest in possibly lower quality, viewport

Table 1: Notations of the QoE model [0].

Notation	Meaning
$s$	Segment
$R$	Bitrate selected (Kbps)
$E$	Rebuffering time (seconds)
$L$	Latency (seconds)
$P$	Playback speed (default $1\times$ )
$S$	Total number of segments
$\alpha$	Bitrate multiplier
$\beta$	Rebuffering multiplier
$\gamma$	Latency multiplier
$\sigma$	Playback speed multiplier
$\mu$	Bitrate switch multiplier

quality is still a valid metric to evaluate the performance of tile based streaming. Viewport quality is calculated by the weighted average of the qualities of each tile by considering their area in the viewport tile. In Figure 6, a typical VDS scenario is shown, the gray rectangle represents the viewport area where the user is able to see. As it can be seen that the tiles in the corners and edges are partially visible in the viewport.

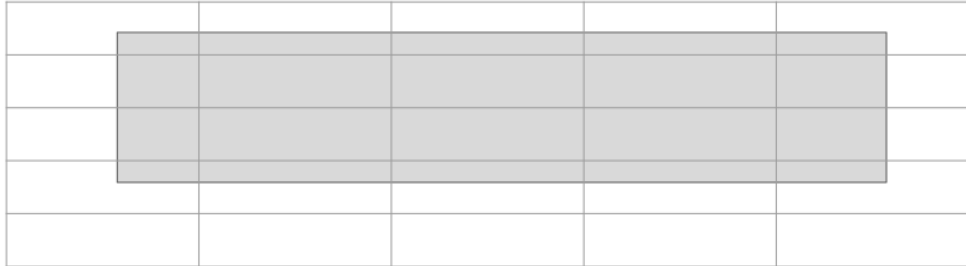


Figure 6: Viewport quality calculation based on tile intersection with the viewport.

Considering each tile represented with rectangles in the viewport, each of those tiles can possibly (but not necessarily) have different viewport quality assigned. In order to measure the overall viewport quality more accurately, the viewport total area and intersection area with the viewport for each tile are considered while calculating an overall viewport quality. In Equation 2 the average weighted viewport quality calculation is shown. Here the  $Q_{Ti}$  is the quality of each tile in the viewport,  $Area_{Ti}$  is the area of the tile visible in the viewport and  $N$  is the

number of tiles in the viewport. With this calculation we try to be fair for the partially visible tiles that are placed at the corners of the viewport.

$$\text{Weighted Average Viewport Quality} = \frac{\sum_{i=1}^N (Q_{T_i} \times Area_{T_i})}{\sum_{i=1}^N (Area_{T_i})} \quad (2)$$

Another important metric which helps to measure the QoE of 360 VDS streaming is the metric called Motion-To-High-Quality-Delay (MTHQD) which was introduced in [15]. MTHQD is the time passed when a tile stays in lower quality till the quality for this tile is switched back to high quality. In VDS when the user is making a head movement faster than the download speed for the next segment of the viewport, it is most likely that the user will hit low quality tiles. When this low quality tile is in the viewport or close to the viewport, ABR will make a decision to switch to high quality for this tile and request the high quality representation for it. During this switch there might be possible low quality tile segment(s) that can be rendered in the viewport and this low quality rendered duration is defined as MTHQD. In this thesis, we proposed several algorithms to decrease MTHQD in the following sections.

## CHAPTER II

# ALGORITHMS FOR ON-DEMAND AND LIVE STREAMING

### *2.1 Streaming of Content-Aware-Encoded Videos*

A common and easy to implement practice in adaptive streaming is using constant bitrate encoding (CBR) during the content creation phase for on-demand video, resulting in equal or very close segment sizes for a specific bitrate. Even though use of CBR simplifies the implementation of the rate adaptation for developers, natural content is not suitable for CBR encoding, since it exhibits a large variation in complexity in the temporal domain. Thus, during a playback session, even if the network conditions do not change and all the segments are requested from one particular representation, a naive client that streams CBR-encoded segments inevitably experiences quality variation between the complex scenes or the ones involving more motion (e.g., an explosion) and the simple or static scenes (e.g., talking faces), unless the encoding bitrate is sufficiently high for even the complex scenes. There is a good amount of prior research on this topic. The first study [16] introduced the concept of consistent-quality streaming and solved this problem using an optimization framework. The proposed method allocated the bits among the segments such that it yielded the best overall quality based on a metric that captured the presentation quality of a group of segments. Later studies (e.g., [17], [18], [19], [20]) tackled the same problem by making certain assumptions and simplifications to come up with a practical solution. For example, the problem at hand gets simplified when one produces each representation in a (near) constant-quality fashion (using capped variable bitrate (VBR) encoding) and take the resulting segment size information into account in rate adaptation in addition to the available bandwidth and the amount of media there is in the

playback buffer. The available bandwidth varies over time and this is a significant constraint in any real-time streaming application, and since we can reliably predict the available bandwidth only for the next several seconds, not minutes, rate adaptation decisions can only be made for a short horizon. In the case of live streaming, the output of the encoder is only known for a few segments and this limits the horizon further. For the segments that are yet to be encoded, actual sizes will be unknown. On the client side, we use the playback buffer as a breathing room for encoding bitrate variability. This buffer typically has a minimum and maximum size limit: if it is drained beyond the minimum threshold, the client should be conservative in rate adaptation as it may soon rebuffer and if more data arrives than the maximum size limit, the excess gets discarded. Normally, the goal is to keep a safe amount of data in the playback buffer between the minimum and maximum size limits.

### 2.1.1 The SARA Algorithm

In this thesis, our target application is video-on-demand (VoD) streaming where the source content is pre-encoded and packaged, and the size information for each segment of every representation is known a priori. For this case, we develop a solution, implement it for dash.js [10] and test it in a number of different scenarios. It is worth noting that this solution is a component inside the rate adaptation logic already built in at the client side and it is orthogonal to the specific video codec/encoder. That is, it works with any open-source or commercial encoder and any video codec such as AVC, HEVC, VVC or AV1. In Figure 7 the overall architecture of the new ABR proposed in this thesis is shown.

SARA selects the most suitable media representation for the next segment to be downloaded from the list of available representations considering how much media is available in the playback buffer, minimum and maximum playback buffer sizes, predicted bandwidth and sizes of the candidate segments from the available representations. Every downloaded segment is placed in the playback buffer, which cannot take more media than  $B_{max}$  (specified in terms of seconds). If the buffer

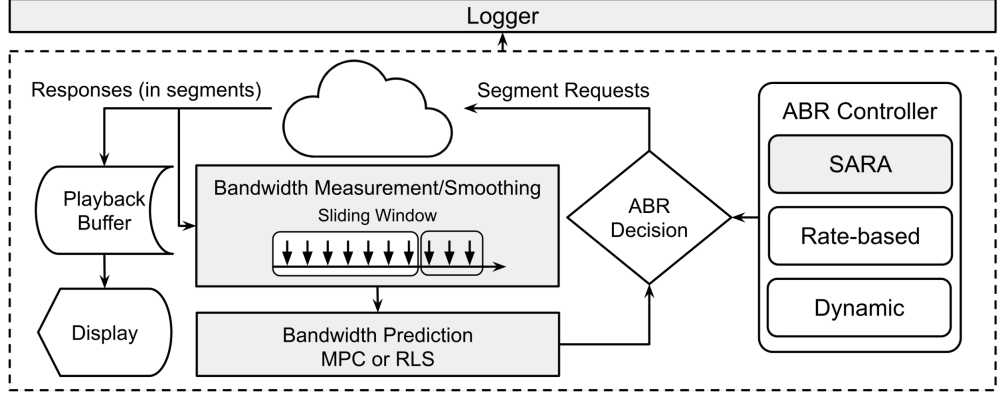


Figure 7: SARA architecture [0].

level drops lower than a minimum size of  $B_{min}$ , the client needs to be careful due to increased risk for a rebuffering. At step i, the SARA ABR rule computes the permutation table by looping through all the available representations and works as follows:

1. Compute the next download time (denoted by  $NDT_{i+1}$ ) for each candidate segment from the available representations by dividing its size (denoted by  $NSS_{i+1}$ , which is available from the segment-size list) by the predicted bandwidth.
2. Compute the next buffer level (denoted by  $B_{i+1}$ ) for each candidate segment from the available representations by adding the segment duration to the current buffer level (denoted by  $B_i$ ) and subtracting the next download time ( $NDT_{i+1}$ ).

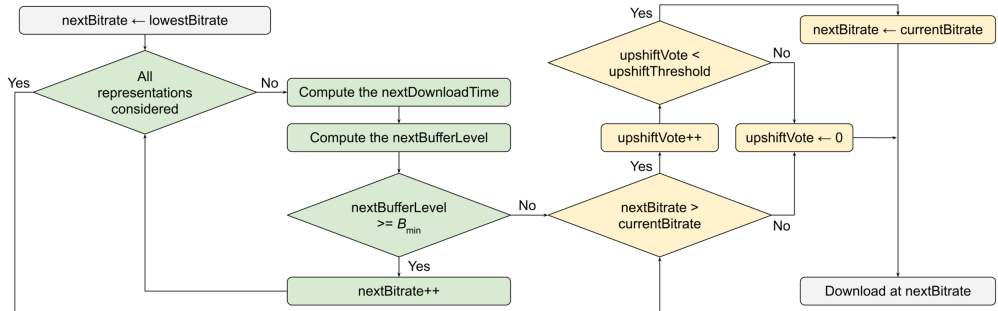


Figure 8: SARA ABR flowchart [0].

SARA picks the segment from the representation with the highest encoding bitrate that ensures the next buffer level  $(B_{i+1}) \geq B_{min}$ . If no candidate segment ensures this condition, the representation with the lowest encoding bitrate is selected. The overall flowchart of the SARA ABR rule is given in Figure 8, which is invoked after downloading each segment to determine the representation for the next segment. In the flowchart, the green boxes handle the size-aware rate adaptation logic and the orange boxes control the upshifting aggressiveness through the `upshiftThreshold` parameter, which is set to five in this work.

In order to make the bandwidth prediction in a more stable way we used two different approaches shown in Figure 9.

RLS [21] is an online linear adaptive filter that recursively finds the filter coefficients that minimize a cost function relating to the input signals for accurate prediction. The RLS-based bandwidth prediction works in two phases at every segment download step  $i$ :

- Phase 1 (Filter Taps): It takes as an input a vector of the last  $M$  ( $M = 4$  in our setup) smoothed bandwidth measurement values given by the bandwidth measurement/smoothing component, and then recursively calculates the gain vector, inverse correlation matrix of the smoothed bandwidth measurement values and the estimated error (denoted by  $\epsilon$  in Figure 9).
- Phase 2 (Update): These are used to update the filter taps vector (denoted by  $W_i$  in Figure 9) of length  $M$ , and finally return the bandwidth prediction for the next  $z$  steps.

RLS has three important benefits. First, it does not require extensive computational capabilities, allowing its deployment over practical real-time systems with commodity mobile devices. Second, it exhibits extremely fast convergence and does not need a prediction model. Third, it is robust against time-varying network conditions through its forgetting factor.

MPC [22] is a widely deployed approach in the industry and has been proven



to have a good forecasting and prediction performance. It falls into the model-based control category, which is known to be simple and practical, and it works in various environments. The MPC approach essentially selects the future bandwidth prediction by looking  $z$  steps ahead, considering the last  $M$  ( $M = 4$  in our setup) smoothed bandwidth measurement values. At each segment download step  $i$ , the MPC approach works in two phases:

- Phase 1 (Process): It takes as an input a vector of the  $M$  most recent smoothed bandwidth values given by the bandwidth measurement/smoothing component.
- Phase 2 (Optimizer and Model): This represents the core of the MPC approach where given  $C_i$  and  $z$ , it predicts the future bandwidth using off-the-shelf MPC optimizer function `fmpc` with a good accuracy (more than 90% in our case). Here, the objective function is to find the bandwidth prediction that minimizes the prediction error.

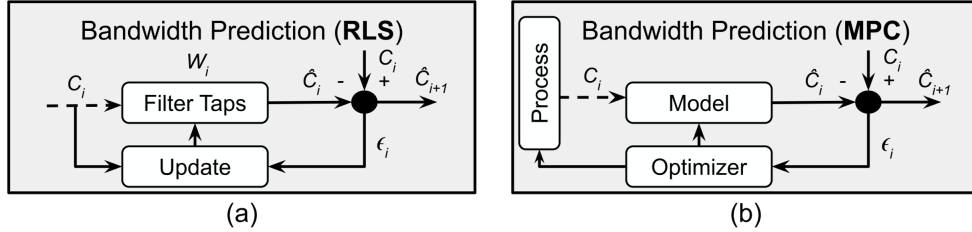


Figure 9: Bandwidth prediction implementations using RLS and MPC.

### 2.1.2 Evaluation

In order to evaluate our enhanced algorithm we have used a VBR content which has a varying bitrate as shown in Figure 10 with 4 different qualities 360p, 540p, 720p and 1080p. The trend of the encoding is also very hard to predict regarding the unexpected peaks in the figure. The test content created consists of a 10 minute video mixed from a variety of videos that exhibited different scene complexities.

We used dash.js [10] environment for the testing environment and compared our new proposed SARA algorithm with rate-based and dynamic ABR algorithms

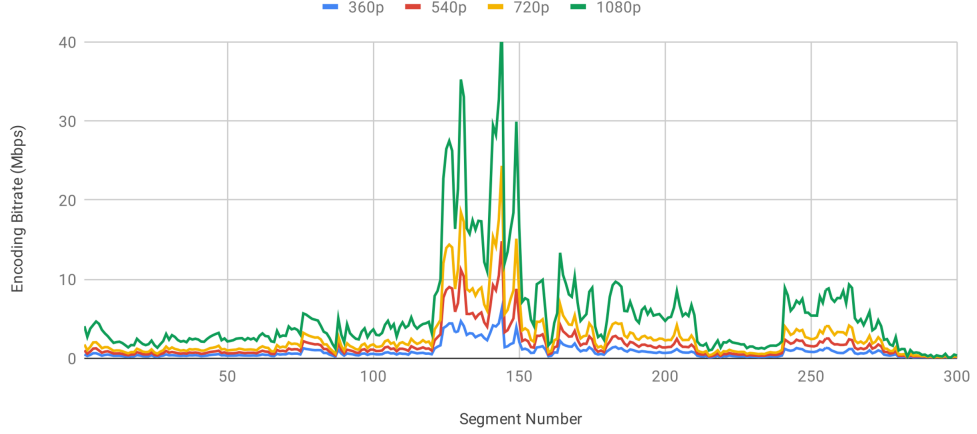


Figure 10: Encoded content bitrate used in tests.

which are well known existing ABR algorithms inside dash.js player. In order to make a fair comparison, we have simulated the network environment with changing bandwidth conditions using tc [23]. We have selected 3 different bandwidth profiles Cascade, Twitch and LTE as shown in Figure 11.

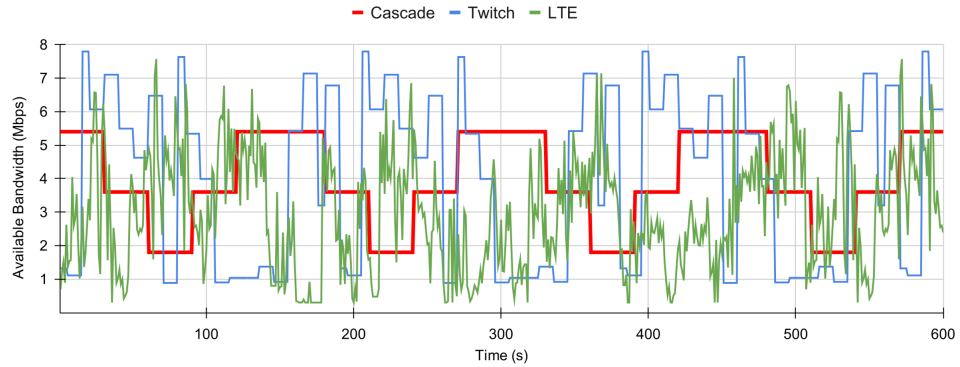


Figure 11: Network profiles used in tests.

We have created 3 versions of the SARA algorithm which are SARA-Basic, SARA-RLS and SARA-MPC as discussed above and compared all those 3 different implementations with dash.js rate-based and dynamic ABR algorithms in Table 2. In the table, total download (TD), total rebuffering duration (TRD), long rebuffering count ( $\geq 40$  ms), short rebuffering count ( $< 40$  ms) and total percentage of HD content rendered ( $\geq 720p$ ) is shown. All three SARA ABR

rules (SARA-Basic, SARA-RLS and SARA-MPC) experience not only a smaller total number of rebufferings but also a shorter total duration of rebufferings. This vastly improves the viewer experience. This is largely due to the fact that SARA takes segment sizes into account in its decisions such that it can avoid fetching larger-than-expected segments if there is a risk of rebuffering and it can carry on fetching smaller-than-expected segments if there is no risk of rebuffering. The rate-based and dynamic ABR rules have the best HD performance for the Cascade profile (i.e., when the network is more stable) but are never able to reach 1080p quality, while SARA-Basic, SARA-RLS and SARA-MPC fetch 87, 115 and 120 segments, respectively, at 1080p. Regarding the Twitch profile, the rate-based and dynamic ABR rules can fetch few 1080p segments and the SARA flavors fetch about half of the segments at 1080p. Similarly, for the LTE profile, rate-based and dynamic ABR rules fetch very few 1080p segments whereas the SARA flavors fetch about one third of the segments at 1080p.

Bandwidth Profile	ABR Rule	TD (MB)	TRD (s)	LRC	SRC	HD (%)
Cascade	Rate-based	184.00	4.92	6	11	88.33
	Dynamic	165.88	3.17	10	8	80.67
	SARA-Basic	178.56	1.26	3	5	65.33
	SARA-RLS	196.10	0.40	1	1	65.00
	SARA-MPC	198.06	0.80	2	1	66.67
Twitch	Rate-based	180.88	2.69	6	8	61.67
	Dynamic	186.57	2.54	10	6	68.00
	SARA-Basic	231.48	1.40	3	7	74.00
	SARA-RLS	199.01	1.00	3	4	55.00
	SARA-MPC	203.62	1.02	4	5	51.33
LTE	Rate-based	188.78	1.90	5	12	87.00
	Dynamic	130.19	1.30	6	7	53.00
	SARA-Basic	140.38	1.27	5	6	40.67
	SARA-RLS	146.10	1.00	3	7	43.67
	SARA-MPC	161.47	0.95	4	5	62.33

Table 2: Comparison of rate-based, dynamic, SARA-Basic, SARA-RLS and SARA-MPC ABRs.

## 2.2 Algorithms Using Common Media Client Data/Server Data

### 2.2.1 Common Media Client Data (CMCD)

Recently, Consumer Technology Association (CTA) released the Common Media Client Data (CMCD; CTA-5004) [24]. Following the CMCD release, after making several discussions about standards and use cases, a new concept of Common Media Server Data (CMSD) is also being discussed recently. After these standards were published, we have investigated if these standards might help ABR algorithms to increase the QoE achieved by the streaming clients. We have used the parameters defined in Table 3.

Parameter	Key	Type	Unit
Encoded bitrate	br	integer	Kbps
Buffer length	bl	integer	ms
Buffer starvation	bs	boolean	-
Deadline	dl	integer	ms
Measured throughput	mtp	integer	Kbps
Requested max. throughput	rtp	integer	Kbps
Object type	ot	token	-
Max buffer (new)	com.example-bmx	integer	ms
Min buffer (new)	com.example-bmn	integer	ms

Table 3: Previously defined and the new added CMCD parameters [0].

In order to show how CMCD might help us we have implemented CMCD aware client and server and implemented a basic ABR which has the ability to share the client data with the server. We shared the buffer data with the server and made the bandwidth allocation in the server side based on the buffer data which is sent from the client side as shown in the Listing 2.1.

```
function bufferAwareBandwidthAllocation(req) {  
    var cmcd_params = processQueryArgs(req);  
    var r = 0;  
    var C = getAvailibleTotalCapacity();
```

```

if (!( 'bl' in cmcd_params) || !( 'com.example-bmn' in
    cmcd_params) || !( 'com.example-bmx' in cmcd_params)
    || !( 'ot' in cmcd_params)) {
    return 0; /* Disable bandwidth allocation */
}
if (cmcd_params['ot'] != 'v' && cmcd_params['ot'] != '
    av') { /* not video object */
    return 0; /* Disable bandwidth allocation */
}
/* Buffer-to-rate mapping */
var  $C_{min}$  = C x (1 -  $\alpha$ );
var  $C_{max}$  = C x  $\alpha$ ;
var  $B_{min}$  = cmcd_params['com.example-bmn'];
var  $B_{max}$  = cmcd_params['com.example-bmx'];
var Bufferlength = cmcd_params['bl'];
/* Case S1 */
if (Bufferlength <  $B_{min}$  || ('bs' in cmcd_params)){
    r =  $C_{max}$ ;
}
/* Case S2 */
else if (Bufferlength >  $B_{max}$ ) {
    r =  $C_{min}$ ;
}
/* Case S3 */
else {
    var  $B_{range}$  =  $B_{max}$  -  $B_{min}$ ;
    var  $C_{range}$  =  $C_{max}$  -  $C_{min}$ ;
    r = ((1 - ((Bufferlength -  $B_{min}$ ) /  $B_{range}$ )) *  $C_{range}$ )
        +  $C_{min}$ ;
}

```

```

    }
    return r;
}

```

Listing 2.1: Buffer-aware server-side bandwidth allocation algorithm [0].

The objective of this algorithm is to find the correct rating with a given buffer length, so that both number of rebufferings and rebuffering durations can be reduced without affecting the playback quality. In order to address this problem in the algorithm, we have defined three cases which we refer as S1, S2 and S3 scenarios defined as follows:

- (S1): Buffer underflow case where  $bl < B_{min}$ . The algorithm will pick the maximum available rate. ( $r = C_{max}$ ).
- (S2): Buffer overflow case where  $bl > B_{max}$ . The algorithm will pick the minimum available rate. ( $r = C_{min}$ ).
- (S3): Buffer is safe and stable case where  $B_{min} \leq bl \leq B_{max}$ . The algorithm will compute the rating based on the buffer value using the equation

$$\begin{aligned}
 B_{range} &= B_{max} - B_{min} \\
 C_{range} &= C_{max} - C_{min} \\
 r &= C_{min} + ((1 - ((bl - B_{min})/B_{range})) \times C_{range}).
 \end{aligned}$$

In this equation  $C_{max} = \alpha \times C$ ,  $C_{min} = (1 - \alpha) \times C$  and  $C$  is the total available maximum bandwidth capacity. Here, we have used a safety factor in the algorithm denoted as  $\alpha = 0.9$  which is the default value in dash.js [10] reference player implementation.

The results for 10 clients running on-demand video sessions are shown in Table 4. We used Spike and Cascade network profiles for those scenarios which can be accessed from [25]. In Table 4, the effect of enabling CMCD can be seen clearly. The Average Rebuffering Duration (RD), Max Rebuffering Duration and Average

	CMCD	NO CMCD	CMCD	NO CMCD
	CascadeX10		SpikeX10	
<b>Avg. BR</b>	3.13	3.33	2.61	3.20
<b>Min. BR</b>	2.90	3.12	2.30	2.68
<b>Avg. RD</b>	5.36	20.84	12.43	71.90
<b>Max. RD</b>	10.72	38.84	18.49	83.54
<b>Avg. RC</b>	4.72	11.04	8.68	25.48
<b>Avg. SC</b>	35.80	36.70	49.14	53.90

Table 4: On-demand video streaming with 10 clients [0].

Rebuffering Counts for both Spike and Cascade profiles are noticeable reduced. For Cascade profile the reductions for Average RD, Max RD, Average RC are 74%, 72% and 57% respectively. Similarly for Spike network profile those improvements were 83%, 78% and 66% for Average RD, Max RD, Average RC, respectively.

We also evaluated CMCD vs NO CMCD scenarios with 20 and 30 clients to see how CMCD scales when the concurrent streaming clients increases. The results with Spike and Cascade profiles with the increasing number of clients, are shown in Table 5. 20 client scenarios are shown in the table with *CascadeX20*, *SpikeX20* and similarly 30 client scenarios are shown with *CascadeX30*, *SpikeX30* results. For Cascade profile, 41% and 15% reduction in Average RD, 38% and 17% in Max RD, 16% and 6% in Average RC was observed for 20 and 30 client scenarios respectively. Similarly for Spike network profile, 48% and 1% reduction in Average RD, 35% and 0% in Max RD, 36% and 1% in Average RC was observed for 20 and 30 client scenarios respectively. Another result that we can tell based on those results is the need for a better algorithm, since the improvements achieved using CMCD decreases as the number of clients are increasing. Here we have used a basic bandwidth allocation to focus on the effects of CMCD. With a more advanced server-side algorithm we believe the improvements will be more than the results we have achieved.

	CMCD	NO CMCD	CMCD	NO CMCD	CMCD	NO CMCD	CMCD	NO CMCD
	CascadeX20		SpikeX20		CascadeX30		SpikeX30	
<b>Avg. BR</b>	3.20	3.39	2.78	3.16	3.32	3.34	3.07	3.11
<b>Min. BR</b>	2.88	3.03	2.22	2.55	2.93	2.97	2.33	2.41
<b>Avg. RD</b>	14.42	24.58	32.14	61.87	31.57	37.00	47.99	48.39
<b>Max. RD</b>	27.57	44.43	51.13	78.13	59.86	71.70	70.66	70.21
<b>Avg. RC</b>	9.24	11.05	14.41	22.36	14.17	15.13	19.03	19.26
<b>Avg. SC</b>	38.10	34.84	42.78	50.36	35.86	35.54	46.73	46.81

Table 5: On-demand video streaming with 20 and 30 clients [0].

### 2.2.2 Common Media Server Data (CMSD)

After the CTS shared initial ideas about CMSD specs we have further investigated quick benefits of having CMSD in our setup that we used in CMCD. And we have extended our server implementation to return a new parameter `com-example-dl` which is the delay for the request in the server side. With the delay amount the client can ignore those extra waiting times and calculate the download duration and throughput more accurately.

As stated earlier in Listing 2.1 our CMCD solution will deliver the bandwidth resource to the clients based on the buffer levels in the clients which creates better results for the entire system as shown previously. However while doing that resource allocation in the server side, the clients that have higher buffer levels will probably get a lower bandwidth with this algorithm and this server behaviour might misguide the client to measure the available bandwidth and cause unnecessary downshifts. With our new data sent from server using CMSD, which is the delay caused in the server, will help the client to measure the available bandwidth accurately by considering the delay duration in the server side. We have used NGINX, node.js and puppeteer to run our experiments using dash.js. For the dataset we have used five different representations which are 80p, 360p, 430p, 570p, 720p representations of the famous *Big Buck Bunny* video.

We have evaluated our setup with CMSD enabled and the results can be seen in Table 6. We have observed 33% and 56% reduction in Avg. Rebuffering Duration (RD), 30% and 27% reduction on Avg. Rebuffering Count (RC) for 10



	<b>CascadeX10</b>		<b>CascadeX20</b>	
<b>Metric</b>	<b>CMSD</b>	<b>NO CMSD</b>	<b>CMSD</b>	<b>NO CMSD</b>
<b>Avg. BR</b>	3.46	3.55	3.20	3.26
<b>Min. BR</b>	3.15	3.27	2.45	2.59
<b>Avg. RD</b>	3.52	5.26	0.51	1.16
<b>Max. RD</b>	15.0	14.5	4.15	8.21
<b>Avg. RC</b>	1.52	2.18	0.40	0.55

Table 6: Results for VoD streaming for 10 and 20 clients using CMSD [0].

(CascadeX10) and 20 (CascadeX20) client scenarios, respectively. We have observed a small negligible difference for Average Bitrate (BR) which is 2% during our tests. Average Bitrate doesn't change a lot even though we have introduced a new delay on the server side which is also sent to the client side for the measured throughput calculations. This is an initial design which shows us that CMSD can be used to fine tune the client side ABR decisions with the extra implementation or information details supplied by the server side.

## CHAPTER III

# ALGORITHMS FOR LOW-LATENCY LIVE STREAMING

### *3.1 Key Enablers for Low Latency*

#### **3.1.1 Common Media Application Format (CMAF)**

As the delivery of video content over HTTP is becoming popular nowadays, the need for lower latency values with online content is also increasing. As discussed earlier in order to stream video, it is divided into smaller segments to send to the client over HTTP and those playable segments are streamed by the client. Considering the need of low latency for online content, the configuration of the segment size on the server side is playing an important role to achieve low latency. For example, if we have a segment size of 2s, then the expected latency will be equal or higher than 2s. So in order to solve this problem chunks are introduced for low-latency live streaming. There have been multiple improvements in that area like Apple's HLS, DASH-LL using Common Media Application Format (CMAF). One of the most common standards was introduced by MPEG to create a standard way of communication between server and client which is defined as Common Media Application Format (CMAF [26]). CMAF takes the fragmented mp4 content and delivers media content with the chunks shown in Figure 12.

In Figure 12, a CMAF fragment example is shown with a single movie fragment box (moof). Moof box is defined as a combination of movie fragment header (mfhd) and track fragment (traf) where the details of those boxes can be found in ISO/IEC 23000-19:2020 [26]. In the bottom part of the Figure 12, it is shown how CMAF data format might be very useful for low latency scenarios especially when the video content is not ready and generated live. In the figure, the same data which is shown at the top is separated into 5 CMAF fragments, thus the

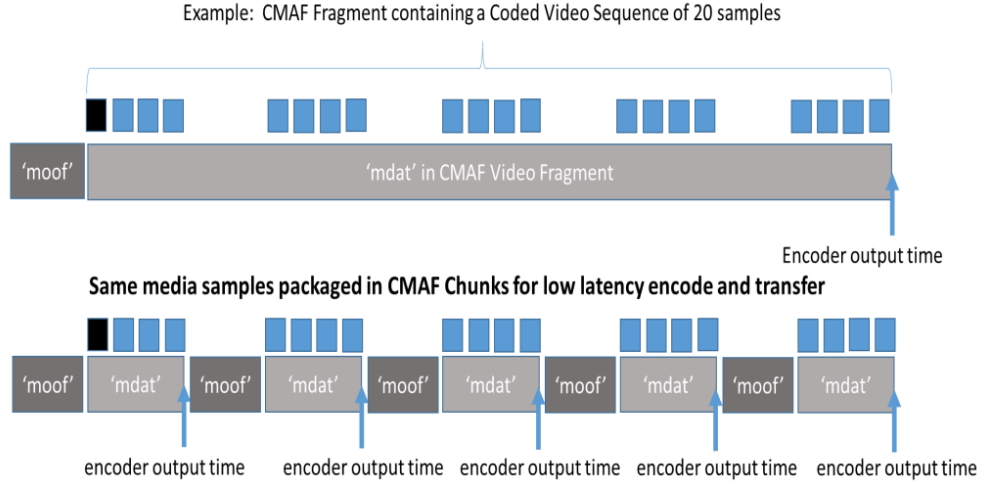


Figure 12: CMAF chunks for low-latency [7].

client might have those packages earlier and with that architecture for the clients, it will be possible to achieve lower latency results. The important achievement here is that a CMAF chunk is actually smaller than the segment size defined in the server size which allows us to initiate the playback of the content before the creation of the segment finishes.

### 3.1.2 Chunked Transfer Encoding (CTE)

After generating CMAF chunks in the transport layer there is also a standard protocol supported by HTTP/1.1 which is called Chunked Transfer Encoding (CTE). CTE is a more general term and it is not only designed for low latency, but also it can be used to transfer a big image file from a web server to the clients. In HTTP/1.1 this is accomplished with HTTP header *Transfer-Encoding: chunked*. Possible other values for this header are compress, deflate and gzip which are defined in the HTTP RFC [27]. CTE is designed to transfer content from server to client in small chunks. In CTE the data that needs to be transmitted is divided into smaller chunks of data and each of them is transferred with this new header included in the header.

When CTE is enabled the *Content-Length* header will not be used and instead each chunk will have its own size at the beginning of the data. The last chunk

created will be a zero length chunk to indicate the end of the transfer. With CTE the video streaming client and the media server will maintain a persistent connection to stream the data in smaller bits of data. One of the most common approaches in low-latency live streaming is to use CMAF with CTE enabled. We have also used them together to implement our proposed algorithm using dash.js in the next section.

### ***3.2 The LoL and LoL<sup>+</sup> Algorithms***

As stated earlier CMAF chunks help us accomplish low-latency streaming since it enables us to package smaller media content and deliver that content to the client to start playback earlier which helps us to decrease the latency between the server content creation and client playback. Even though the measured latency value in the client improves using CMAF, there is another challenging part here which is not a negligible problem for the client side. Most of the ABRs measure the values like buffer, bandwidth, latency, etc.. and try to make a decision based on these observed metrics. However, predicting the bandwidth accurately or calculation on the client side may not be possible due to the idle times between the arriving CMAF fragments.

Traditional ABRs count the bytes received and divide that value by the duration which eventually results in the encoding rate of the server especially when the content is being produced live. When the encoder is waiting to produce new chunk data and the client request is blocked till the content becomes ready, while calculating the available bandwidth, this waiting time has to be discarded from the calculations. Otherwise, it is very likely to end up with the encoding rate instead of measuring bandwidth accurately.

Recently, there have been many researches in that area to predict the bandwidth and adapt better using improved ABR algorithms. In [28] and [29] Lie et al. proposed a bitrate estimation algorithm and dealt with network related problems with extra smoothing functions by using their SFT and ESFT methods. In

their work, they also considered parallel segment fetching scenarios. Rainer et al. [14] also proposed a bandwidth prediction based on previous experiences. In their work, they have tried to predict the next segment based on the previously measured values. In [30] Z. Li et al. propose a new bandwidth prediction algorithm called Probe AND Adapt (PANDA) in order to avoid ABR switch oscillations where multiple clients are competing for the same resource on the shared network scenario. A physical layer bandwidth prediction module was mentioned in [31], similarly [32] has also created a bandwidth estimation module that uses future base layer data by using a heuristic bandwidth shaping algorithm they have proposed. FESTIVE [33] and Squad [34], are ABR algorithms that were introduced to track and make decisions based on the buffer level of the client player. If the client's buffer level gets lower, lower bitrate quality representations are selected in order to avoid possible future stalls during the video playback session. Also, authors have proposed heuristic ABR algorithms in BBA [35], BOLA [36], and Quetra [37] in order to adapt to the changing environment conditions at segment boundaries before starting to download. In their work authors in [38] proposed a practical bandwidth prediction algorithm in order to avoid oscillations due to frequently changing network conditions.

There have been several other studies recently to address this problem and increase the quality of experience (QoE) for low latency streaming recently by predicting bandwidth in low latency streaming considering the content being created synchronously with the player requests. For bandwidth prediction, Bentaleb *et al.* [39, 40] proposed an ABR called ACTE for Chunk Transfer Encoding (CTE) using CMAF. ACTE predicts bandwidth and considers video playback buffer to make ABR decisions. Gutterman *et al.* [41] designed an algorithm called STALLION consisting of a sliding window. STALLION calculates the mean and standard deviation and makes ABR decisions based on those values to increase stability. In [42], Peng *et al.* implemented another new algorithm with a heuristic playback rate controller for low-latency streaming. The authors proposed a low

latency ABR in [43] that considers different paths based on frame sizes.

In this thesis, we have developed a new algorithm to overcome this problem. Considering ABRs trying to choose one of the representations in each segment boundary, this problem can be thought of as a classification problem between the alternative representations on each segment boundary. For classification problems, a well known approach using unsupervised learning techniques is the Self Organizing Maps (SOM) proposed by Kohonen [44]. The same approach has been applied to even some NP-hard problems like the traveling salesman problem [45]. For each segment boundary, we want to find the closest representation to an imaginary ideal point. The feature map is shown in Figure 13. Each state is represented by measured throughput, live latency, buffer level and calculated QoE. In our setup we used the QoE model based on [46] with some extra customization mentioned in Section 1.3.1.

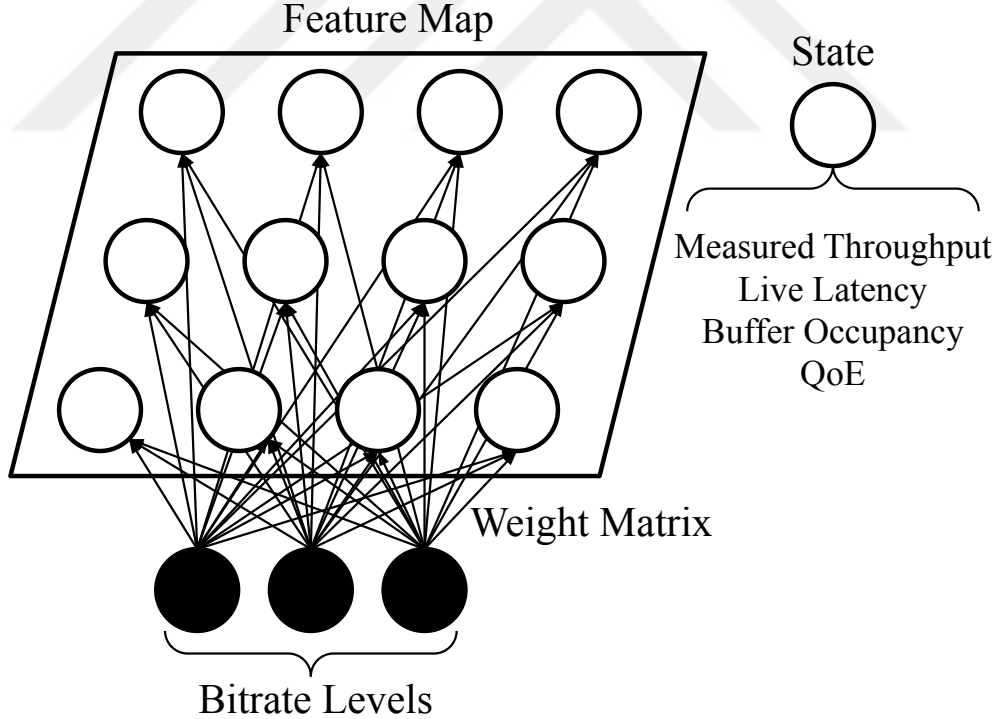


Figure 13: Example SOM feature map setup for three bitrate alternatives.

In each segment boundary, the distance from the ideal point with latency = 0, rebuffering = 0 and total number of switches = 0 which is an impossible state

to reach in reality, is calculated. For each quality representation, we evaluate the current values for the state and update the corresponding neuron. Then we search for the closest neuron to the ideal point that we created in the feature space. That closest neuron closest to the imaginary ideal point we created is called as Best Matching Unit which will be referenced as BMU for the rest of this thesis. In order to find the BMU we have used the euclidean distance formula shown in Equation 3.

$$d(a, b) = \sqrt{\sum_{i=1}^4 w_{s_i}^n \times (a_i - b_i)^2}. \quad (3)$$

In Algorithm 1 the decision of the next bitrate selection by the ABR is shown which has two major neuron updates. Considering each bitrate representation in the MPD file is corresponding to a neuron in this system, the first update is to move the current selected neuron to the measured point  $\{X_{s_i}, L_{s_i}, E_{s_i}, H_{s_i}\}$ . Then the algorithm finds the BMU by checking the distance for all neurons. After finding the BMU, the algorithm makes the second update which is to move the found BMU towards the impossible to reach ideal point that we created. In our setup we have created that ideal point with latency=0, rebuffering duration=0 and total bitrate switch numbers=0. Checking those two updates again we can say that the first update is an update that keeps the system with the realistic observed values whereas the second is completely opposite and moves the neuron to the ideal point. These two updates may seem to be contradicting each other, but the truth is the system creates a balance between those two updates. After finding the BMU in the second update, it is also important to emphasize that all the neighbor neurons are also updated with a Gaussian neighborhood function. The parameter  $\lambda$  in the algorithm which is also called as the learning rate for SOM model is chosen as 0.01. This is the default value introduced in the original SOM paper [44].

---

**Algorithm 1** Bitrate selection using SOM model [0].

---

```
1: function GETNEXTBITRATE()
2:    $\mathcal{D}_{s_i}^* \leftarrow 0$ ,  $\text{BMU} \leftarrow \emptyset$ ,  $R_{s_{i+1}}^* \leftarrow 0$ ,  $\lambda \leftarrow 0.01$ 
3:   for Each segment downloading step  $s_i \in S$ ,  $i > 0$  do
4:      $X_{s_i} \leftarrow \text{CalculateThroughput}(s_i)$ 
5:      $B_{s_i} \leftarrow \text{GetBufferLevel}(s_i)$ 
6:      $L_{s_i} \leftarrow \text{GetLatency}(s_i)$ 
7:      $E_{s_i} \leftarrow \text{GetRebuffering}(s_i)$ 
8:      $H_{s_i} \leftarrow \text{GetSwitches}(s_i)$ 
9:      $w_{s_i} \leftarrow \text{WeightVectorSelection}(s_i)$ 
10:     $\text{Normalize}(\{X_{s_i}, L_{s_i}, E_{s_i}, H_{s_i}\}, [0, 1])$ 
11:     $\text{Update}(m_{s_i}^{\hat{j}}, \{X_{s_i}, L_{s_i}, E_{s_i}, H_{s_i}\})$ 
12:    for all neurons  $\forall m_{s_i}^{\hat{j}} \in M_{s_i}$ ,  $1 \leq \hat{j} \leq j$  do
13:       $R_{s_i} \leftarrow R_{s_i}^*$ 
14:       $\mathcal{D}_{s_i} \leftarrow \text{GetDistance}(w_{s_i} \times \{X_{s_i}, L_{s_i}, E_{s_i}, H_{ideal}\})$ 
15:      if ( $\mathcal{D}_{s_i} < \mathcal{D}_{s_i}^*$ ) then
16:        if ( $L_{s_i} \leq L_{target}$ ) && ( $B_{s_i} \geq B^{low}$ ) then
17:           $\mathcal{D}_{s_i}^* \leftarrow \mathcal{D}_{s_i}$ 
18:           $\text{BMU} \leftarrow m_{s_i}^{\hat{j}}$ 
19:           $R_{s_{i+1}}^* \leftarrow \text{BMU.bitrate}$ 
20:           $R_{s_{i+1}} \leftarrow R_{s_{i+1}}^*$ 
21:        else
22:           $\text{BMU} \leftarrow \text{SelectNeuron}(X_{s_i})$ 
23:           $R_{s_{i+1}}^* \leftarrow \text{BMU.bitrate}$ 
24:           $R_{s_{i+1}} \leftarrow R_{s_{i+1}}^*$ 
25:        end if
26:      end if
27:    end for
28:     $\text{Update}(m_{s_i}^{\hat{j}}, \{X_{s_i}, L_{ideal}, E_{ideal}, H_{s_i}\})$ 
29:     $\text{Return}(R_{s_{i+1}})$ 
30:     $\text{NextMaxRate}()$ 
31:  end for
32: end function
```

---

### 3.2.1 Weight Selection

As also stated with the previous studies in [47, 48], the performance of Self Organizing Maps (SOM) is directly affected by the selection of the initial weights used in the SOM updates. Based on those values the system might converge quite differently on each run. In order to solve this problem, we have restated that problem as an assignment problem for the weights with a provided utility function to maximize. And we proposed a method to assign the initial values of the SOM weights with the solution of this new assignment problem.



We restate choosing SOM initial weights problem as an *assignment problem* [49] with additional linear inequality constraints. If we check the objective function in 4, we will see that the equation satisfies the condition of being concave and having non-negativity constraints.

We will use the notation  $M_{s_i}$  for the SOM state consisting of neurons for the segment number  $i$ . For each neuron in  $M_{s_i}$ , our goal is to find the best weight vector which satisfies the constraints defined in Equation 4 and the utility function that we want to maximize, which is the QoE defined in Equation 1 in this case. To simplify the weight selection problem we have used normalized values from the set of  $V = \{0.2, 0.4, 0.6, 0.8, 1\}$ .

The constraints that are needed to be satisfied for any weight assignment combination is shown in Equation 4.

$$s.t. \begin{cases} \mathbf{A1} : L_{s_{i+1}} \leq L_{target} + |L_{s_i} - L_{s_{i-1}}| \\ \mathbf{A2} : B_{s_{i+1}} \geq B^{low} \\ \mathbf{A3} : V^{min} \leq w_{s_{i+1}}^{\hat{v}}(.) \leq V^{max}, \quad \forall w_{s_{i+1}}^{\hat{v}} \in W_{s_{i+1}} \end{cases} \quad (4)$$

where  $\mathbf{w}_{s_{i+1}}^*$  is the weight vector that maximizes our utility function which is QoE and the weight that should be used in the next segment ( $s_{i+1}$ ). The details of these constraints are:

- **A1** is the constraint to check the latency value will still be in an acceptable range.
- **A2** is the constraint that the playback buffer will not suffer from being too low which may lead to stalls in the playback session. So this condition will check that the buffer is above a safe threshold.
- **A3** ensures that the weight selection is inside the allowed set which is between 0 and 1 for our case.

---

**Algorithm 2** Initial Weight Assignment Algorithm [0]

---

```
1: function WEIGHTVECTORSELECTION( $s_i$ )
2:    $\mathbf{w}_{s_{i+1}}^* \leftarrow \{\}, W_{s_i} \leftarrow \{\}, u_{s_i}^* \leftarrow 0, V \leftarrow \{0, 0.2, 0.4, 0.6, 0.8, 1\}$ 
3:    $W_{s_i} \leftarrow \text{WeightVectorGenerator}(V)$ 
4:   for Each segment downloading step  $s_i \in S, i > 0$  do
5:      $M_{s_i} \leftarrow \text{GetNeurons}(\text{SOM})$ 
6:     if Initial Stage ( $s_1$ ) then
7:        $\mathbf{w}_{s_1}^* \leftarrow \text{k-means++}(\{w_{s_1}^X, w_{s_1}^L, w_{s_1}^E, w_{s_1}^P, w_{s_1}^H\})$ 
8:     else
9:       for Each SOM Neuron  $m_{s_i}^{\hat{j}} \in M_{s_i}, 1 \leq \hat{j} \leq j$  do
10:        for Each weight vector  $w_{s_i}^{\hat{v}} \in W_{s_i}, 1 \leq \hat{v} \leq v$  do
11:           $u_{s_i}^{(m_{s_i}^{\hat{j}}, w_{s_i}^{\hat{v}})} \leftarrow \text{ComputeQoE}(R_{s_i}, E_{s_i}, L_{s_i}, P_{s_i})$ 
12:           $\mathcal{M}_{s_i}^{(M, W)} = \text{Push}(u_{s_i}^{(m_{s_i}^{\hat{j}}, w_{s_i}^{\hat{v}})})$ 
13:          if A1, A2, A3 in (4) are satisfied then
14:            if  $u_{s_i}^{(m_{s_i}^{\hat{j}}, z_{s_i}^{\hat{v}})} > u_{s_i}^*$  then
15:               $u_{s_i}^* \leftarrow u_{s_i}^{(m_{s_i}^{\hat{j}}, w_{s_i}^{\hat{v}})}$ 
16:               $\mathbf{w}_{s_{i+1}}^* \leftarrow w_{s_i}^{\hat{v}}$ 
17:            else
18:              GoTo(10)
19:            end if
20:          else
21:            if  $\hat{v} = v$  then
22:               $\mathbf{w}_{s_{i+1}}^* \leftarrow \mathbf{w}_{s_i}^*$ 
23:            end if
24:          end if
25:        end for
26:      end for
27:    end if
28:     $\text{Return}(\mathbf{w}_{s_{i+1}}^*)$ 
29:  end for
30: end function
```

---

### 3.2.2 Throughput Measurement

As discussed previously, traditional ways of calculating the throughput for a source limited low latency system, ends up with the encoding bitrates due to the idle times counted in the data transmissions. And that creates an issue for the ABR, even though there may be enough bandwidth resources to download high quality segments, ABR will not measure the available bandwidth and that causes the ABR to stay in lower quality encoded values. So in order to be able to switch to the higher quality encoded video representations, measuring the throughput correctly

---

**Algorithm 3** Generate Possible Weight Permutations [0]

---

```
1: function WEIGHTVECTORGENERATOR( $V$ )
2:    $W_{s_i} \leftarrow \{\}, w_{s_i} \leftarrow \{\}, w_{s_i}^{\hat{v}} \leftarrow \{\}$ 
3:   for Each possible weight vector  $w_{s_i}^{\hat{v}}$  permutation from  $V$  do
4:      $w_{s_i}^{\hat{v}} \leftarrow \text{Generate}(\{w_{s_i}^{X_{\hat{v}}}, w_{s_i}^{L_{\hat{v}}}, w_{s_i}^{E_{\hat{v}}}, w_{s_i}^{P_{\hat{v}}}, w_{s_i}^{H_{\hat{v}}}\}, V)$ 
5:     if  $w_{s_i}^{\hat{v}} \neq w_{s_i}$  then
6:        $W_{s_i} = \text{Push}(w_{s_i}^{\hat{v}})$ 
7:        $w_{s_i} \leftarrow w_{s_i}^{\hat{v}}$ 
8:     end if
9:   end for
10:  Return( $W_{s_i}$ )
11: end function
```

---

is important for the ABR.

When calculating the estimated time to download a segment, it is important to point to the idle times and remove those values from the time calculation which we will denote by  $T_{s_i}$  (time to download the segment  $i$ ). To address this issue an algorithm called ACTE [39] was proposed previously. ACTE works well overall, however, it still suffers from overestimation in some cases where there are increasing idle times during the chunk transmissions.

In order to solve the problems in ACTE we have designed a new throughput measurement method Algorithm 4 which addresses the issues. Basically, the algorithm parses moof box data to identify the start and ends of the chunks and calculates the time between chunks by eliminating idle times during the transmission. The algorithm ignores the first and last data since those chunks may mislead the calculations, the algorithm detects the  $e_{s_i^c}$  and  $b_{s_i^c}$  values accurately as shown in Algorithm 4.

### 3.2.3 Playback Speed Controller

After implementing our first algorithm which we call as LoL and received second place in the Twitch grand challenge [50], we have evaluated the feedbacks and further improved our implementation with dynamic weight assignment as described above and also playback controller to avoid video stalls. Initial LoL was paying attention to the latency and as long as latency wouldn't increase it allowed small

---

**Algorithm 4** Throughput Measurement [0]

---

```
1: function CALCULATETHROUGHPUT( $s_i$ )
2:    $|\tilde{z}_{s_i}| \leftarrow 0$ ,  $\text{Sum} \leftarrow 0$ ,  $X_{s_i} \leftarrow 0$ 
3:   for Each segment downloading step  $s_i \in S$ ,  $i > 0$  do
4:     for Each chunk downloading step  $c > 0$ ,  $\forall c \in s_i$  do
5:        $\text{Flag}_1 \leftarrow \text{ParsePayload}(\text{'moof'}, c)$ 
6:       if ( $\text{Flag}_1 == 1$ ) then  $\triangleright$  'moof' is present in chunk  $c$ 
7:          $b_{s_i^c} \leftarrow \text{performance.now}()$ 
8:          $\text{Flag}_2 \leftarrow \text{ParsePayloadCompleted}(\text{'mdat'}, c, \text{end})$ 
9:         if ( $\text{Flag}_2 == 1$ ) then  $\triangleright$  'mdat' end of chunk  $c$ 
10:           $e_{s_i^c} \leftarrow \text{performance.now}()$ 
11:           $q_{s_i^c} \leftarrow \text{ChunkSize}(c)$ 
12:        end if
13:        if ( $c = 1 \parallel c = z_{s_i}$ ) then
14:           $\text{ChunkFilter}(\text{'noise'})$ 
15:           $x_{s_i^c} \leftarrow 0$ 
16:        else
17:           $x_{s_i^c} \leftarrow \text{ChunkThroughput}(q_{s_i^c}, e_{s_i^c}, b_{s_i^c})$ 
18:        end if
19:         $\text{Sum} \leftarrow \text{Sum} + x_{s_i^c}$ 
20:         $|\tilde{z}_{s_i}| \leftarrow |\tilde{z}_{s_i}| + 1$ 
21:      end if
22:    end for
23:     $X_{s_i} \leftarrow \text{SegThroughput}(|\tilde{z}_{s_i}|, \text{Sum}, \text{SWMA})$ 
24:     $\text{Return}(X_{s_i})$ 
25:  end for
26: end function
```

---

rebufferings. Thus we have improved LoL and called the algorithm as LoL<sup>+</sup>. Playback speed controller is one of the improvements we have accomplished shown in Algorithm 5. We have designed a hybrid playback controller module that pays attention to the current and target latency and the current playback buffer level at the same time. The algorithm keeps an eye on the buffer level if it is below a safe threshold value which is denoted as  $B^{low}$  in the algorithm. This improved algorithm works more robustly and adapts to the changing environment conditions faster than the default algorithm used in dash.js [10] reference player for low latency scenarios.

**Case 1:** Current playback buffer is too low ( $< B^{low}$ ). LoL<sup>+</sup> will slow down the playback speed.

**Case 2:** Current playback buffer is in safe range ( $\geq B^{low}$ ):

- 2a:** The current latency is approximately very close ( $\epsilon = \pm 2\%$ ) to the configured target latency. In this case, LoL<sup>+</sup> will keep the playback speed at normal ( $1x$ ) speed.
- 2b:** Current latency is lower than the configured target latency. In this case, LoL<sup>+</sup> will slow down the playback speed.
- 2c:** Current measured latency is higher than the configured target latency. In this case, LoL<sup>+</sup> will speed up the playback speed to catch the target latency value.

In our setup, we have used the default dash.js playback speed control range which is between  $0.7\times$  and  $1.3\times$ . The playback speed values are assigned by the CalculateSpeed functions shown in Algorithm 5. This function will return a bigger value if the buffer and latency difference between the target and actual values are bigger and it respects the maximum and minimum values as shown in the algorithm.

#### 3.2.4 Evaluation

We have evaluated our new proposed algorithm using 11 different network profiles which can be found in [51]. Five network profiles named CASCADE, SPIKE, INTRA-CASCADE, SLOW-JITTERS and AST-JITTERS were extracted from Twitch [50]. We also manually captured real network traces named TW-LOW, TW-MED from Twitch by measuring the bandwidth on every five seconds interval. We also extracted TRAIN, BICYCLE, TRAIN-MODIFIED and TRAM network traces from the dataset [52]. We compared the results of LoL<sup>+</sup> with the L2A and STALLION algorithms which were the first and the third algorithms of the Twitch grand challenge [50] in addition to the default algorithm in dash.js [10] which is dynamic ABR.

In Figure 14 the detailed results are shown. As it can be seen from the results in Figure 14b LoL<sup>+</sup> gives the best result in all of the 11 network profiles, after

---

**Algorithm 5** Playback Speed Control (Pseudocode) [0]

---

```
1: function PLAYBACKSPEEDSELECTION( $L_{s_i}, L_{target}, B_{s_i}, B^{low}$ )
2:   if ( $B_{s_i} < B^{low}$ ) then
3:      $B_{delta} \leftarrow |B^{low} - B_{s_i}|$ 
4:      $P_{s_i} \leftarrow \text{CalculateSpeed}(B_{delta}, \text{Slower}, \text{Limit}=0.7)$ 
5:   end if
6:   if ( $B_{s_i} \geq B^{low}$ ) then
7:     if ( $L_{s_i} \approx L_{target}$ ) then
8:        $P_{s_i} \leftarrow \text{CalculateSpeed}(\text{Normal}, \text{Speed}=1)$ 
9:     end if
10:    if ( $L_{s_i} < L_{target}$ ) then
11:       $L_{delta} \leftarrow |L_{target} - L_{s_i}|$ 
12:       $P_{s_i} \leftarrow \text{CalculateSpeed}(L_{delta}, \text{Slower}, \text{Limit}=0.7)$ 
13:    end if
14:    if ( $L_{s_i} > L_{target}$ ) then
15:       $L_{delta} \leftarrow |L_{target} - L_{s_i}|$ 
16:       $P_{s_i} \leftarrow \text{CalculateSpeed}(L_{delta}, \text{Faster}, \text{Limit}=1.3)$ 
17:    end if
18:  end if
19:  Return( $P_{s_i}$ )
20: end function
```

---

making the improvements on LoL and creating LoL<sup>+</sup> based on the feedbacks from the Twitch grand challenge. Compared to LoL, on average the improvement in rebuffering durations is 62.6%. Also, we can see the latency results for all 11 network profiles in Figure 14c. Average latency improvement for all profiles is 8.5% considering all network profiles in the results. The major feedback received by LoL was, it was achieving a better latency and bitrates with a rebuffering tradeoff. Compared to LoL, in Figure 14a the reduction in the bitrate for LoL<sup>+</sup> is visible to create a better rebuffering experience.

Compared to L2A, LoL<sup>+</sup> achieves 61.9% reduction in average rebuffering duration and 8.1% reduction in average latency considering all profiles.

Similarly compared to STALLION, improvements of 42.7%, 5.2% and 12.5% were observed in LoL<sup>+</sup> for average rebuffering duration, latency and average bitrate respectively.

The results of LoL<sup>+</sup> compared to dynamic are not very different in terms of average bitrate and latency. However, in terms of rebuffering, LoL<sup>+</sup> performs much

better compared to dynamic ABR. The average rebuffering duration reduction observed for LoL<sup>+</sup> compared to dash.js default dynamic algorithm is 12.3%.

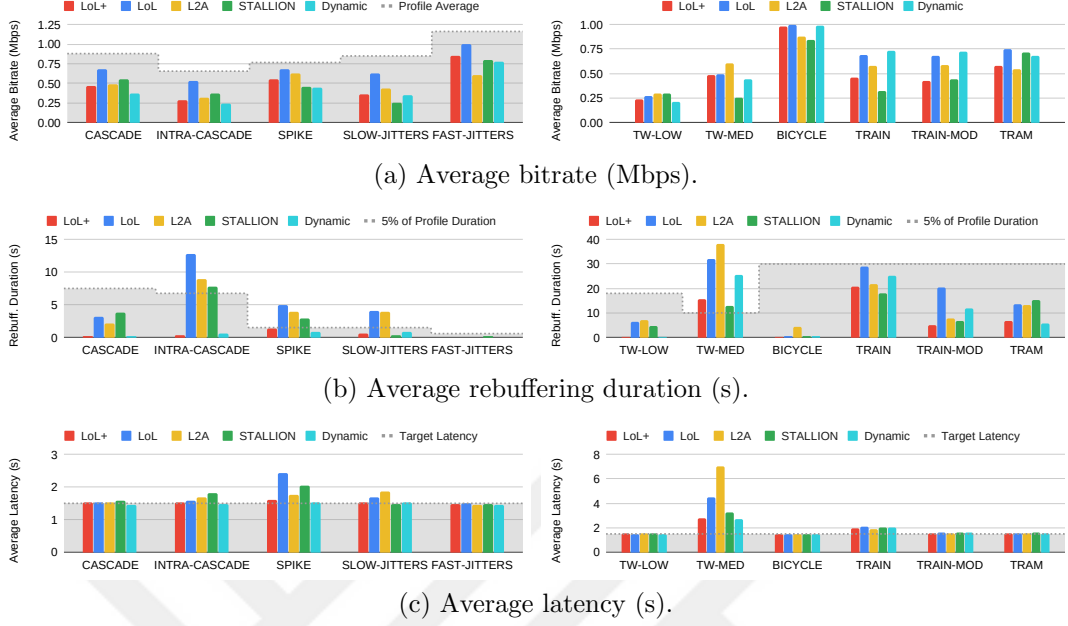


Figure 14: Results of LoL<sup>+</sup>, LoL, L2A, STALLION, Dynamic ABRs for 11 different network profiles [0].

The overall average summary for each ABR algorithm is shown in Table 7. According to the table results, LoL has the highest score in average bitrate with a cost of having lowest score in latency and rebufferings which complies with the initial feedback we received to build LoL<sup>+</sup>. STALLION has the highest score in the average number of switches and has the lowest score in average bitrates selected by the ABR. Dynamic ABR has the highest score in latency. L2A is more in the middle in all of the metrics since it has no highest or lowest score in any of these metrics shown in Table 7. Finally, LoL<sup>+</sup> has the highest score in average rebuffering with an expense of average bitrate switches. As a result, we have extended LoL with LoL<sup>+</sup> to cause less rebufferings based on the higher bitrate selections while keeping the same latency.

### 3.3 Content-Aware Playback Speed Control

After evaluating LoL<sup>+</sup> with the playback controller we have developed, we have observed that LoL<sup>+</sup> can still suffer from rebufferings due to misjudged playback

ABR	Avg. Bitrate Score	Avg. Rebuffering Score	Avg. Latency Score	Avg. Switches Score
LoL <sup>+</sup>	2.53	<b>4.76</b>	3.77	<b>1.09</b>
LoL	<b>4.69</b>	<b>1.97</b>	<b>2.40</b>	4.13
L2A	2.92	2.36	3.08	3.76
STA	<b>2.26</b>	3.19	2.84	<b>4.45</b>
DYN	2.76	4.34	<b>4.52</b>	2.53

Table 7: Average scores used to calculate QoE for all ABRs.

speed controller increases or decreases. So we have investigated the playback rate controller to keep the latency and rebufferings low at the same time. The idea of being content-aware during the streaming session has been previously studied by several researchers. The authors of [53] have worked with on-demand video players around the idea of “scenic car driving,” such that the playback speed is updated based on the scene’s being interesting or not by using predefined rules for the user. In their studies, both [54] and [55] also worked on creating a summary for the video by changing the playback speed. Another important recent work done for the low-latency streaming applications is the work done in [56], where the authors created a model to depict how playback speed control interacts with rate adaptation. While this study provides useful insights for controlling the playback speed of the streaming session, it still does not focus on being content-aware to change the playback speed for low-latency live streaming.

In order to be content-aware, the metadata needs to be generated for the video content that will be streamed. That can be achieved either during the content creation as shown in Figure 15a or the metadata can be extracted during the playback as an alternative to the first scenario which is shown in Figure 15b.

In our implementation, we calculated the event density value which is ranging between 0 and 1 (0: unimportant scene, 1: important scene). By using the event density value we have calculated the playback speed according to Algorithm 6. If the buffer is lower than the  $B_{low}$  then a playback speed which is higher than 1 will be selected otherwise the playback speed will be smaller than 1.

We have evaluated our solution with  $B_{low} = 1$ ,  $D_{max} = 0.3$  and target latency = 3s. In Figure 16 the event densities for the test sequences are shown. White



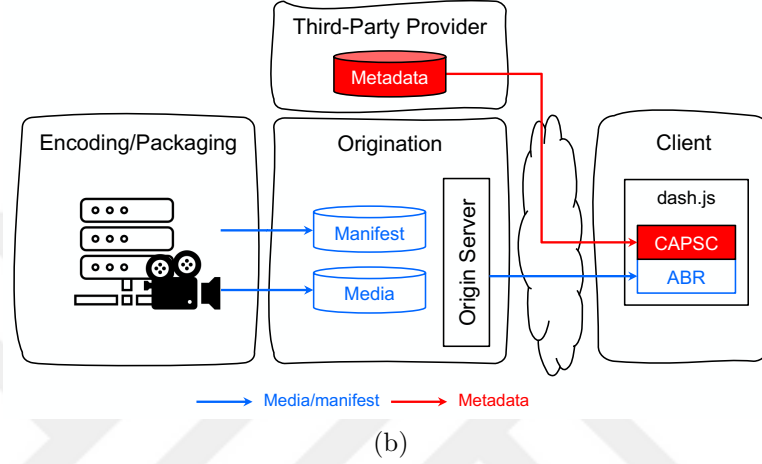
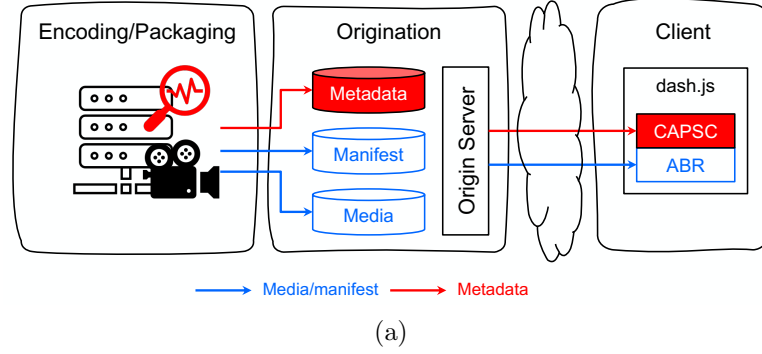


Figure 15: Possible CAPSC implementation workflows for low-latency scenario [0].

Table 8: List of the used notations.

Notation	Definitions
$P$	Playback speed
$s_i$	Segment $i$
$L_{s_i}$	Latency for segment $s_i$
$L_{target}$	Target latency
$B_{s_i}$	Playback buffer level for segment $s_i$
$B_{low}$	Lower bound for playback buffer
$D_{max}$	Upper bound for deviation from normal playback speed
$\gamma_{s_i}^c$	Event density at $s_i$ for chunk $c$

refers to 0 as density and black refers to the maximum value which is 1.

In Figure 17, we evaluated and compared default (17a), LoL<sup>+</sup> (17b) and CAPSC (17c) algorithms. Note that when the buffer is empty, the playback speed needs to decrease too since the playback already stalls, which is also observed by the experiment. Figure 17a shows that the Default algorithm increases the playback when the measured latency increases. However, the drop in the buffer leads to stalls. The LoL<sup>+</sup> algorithm (Figure 17b) addresses this issue by decreasing the

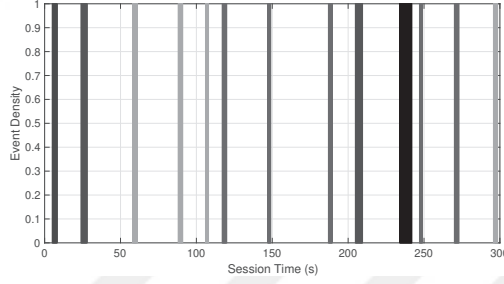
---

**Algorithm 6** Content-aware Playback Speed Control (CAPSC) [0]

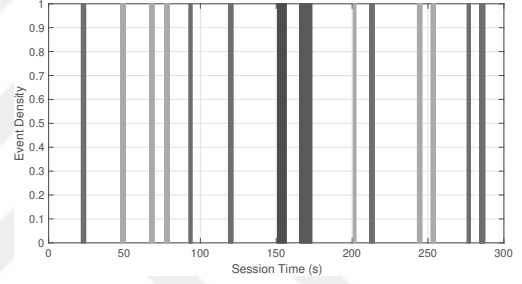
---

```
1: function PLAYBACKSPEEDSELECTION( $L_{s_i}, L_{target}, B_{s_i}, B_{low}, D_{max}, \gamma_{s_i}^c$ )
2:    $\Delta \leftarrow 0$ 
3:   if ( $B_{s_i} < B_{low}$ ) then
4:      $\Delta \leftarrow B_{s_i} - B_{low}$ 
5:   else
6:     if  $|L_{s_i} - L_{target}| \leq 0.02 \times L_{target}$  then
7:        $\Delta \leftarrow 0$ 
8:     else
9:        $\Delta \leftarrow L_{s_i} - L_{target}$ 
10:    end if
11:  end if
12:   $P = \frac{2 \times D_{max}}{1 + e^{-\Delta \times (1 - \gamma_{s_i}^c)^{0.8}}} + (1 - D_{max})$ 
13:  Return( $P$ )
14: end function
```

---



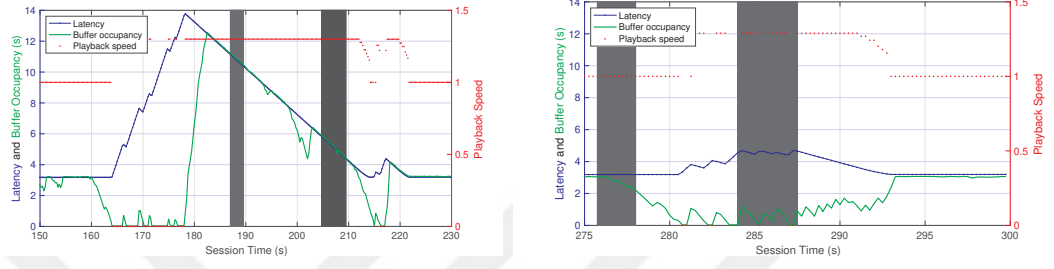
(a) Test-1.



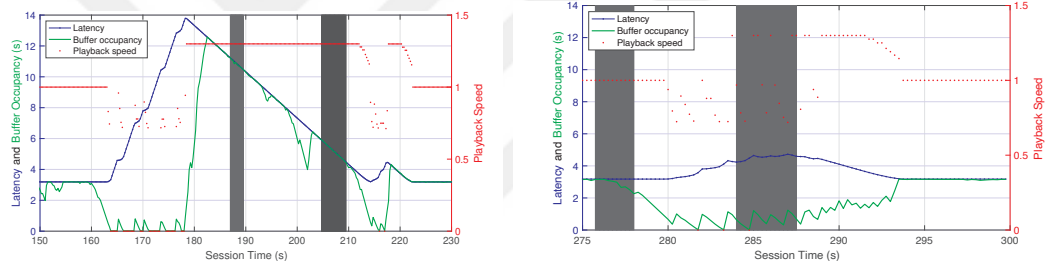
(b) Test-2.

Figure 16: Event densities for different test sequences [0].

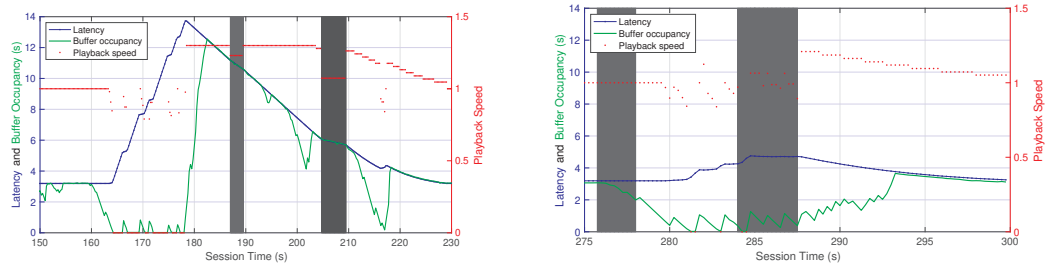
playback as needed, although some of the rebufferings still remain unavoidable because of the insufficient bandwidth resources.



(a) Default ABR results for the Test-1 (left) and Test-2 (right) sequences.



(b) LoL<sup>+</sup> results for the Test-1 (left) and Test-2 (right) sequences.



(c) CAPSC results for the Test-1 (left) and Test-2 (right) sequences.

Figure 17: Results for Default ABR, LoL<sup>+</sup> and CAPSC [0].

## CHAPTER IV

### ALGORITHMS FOR INTERACTIVE STREAMING

#### 4.1 *Bandwidth Prediction for WebRTC*

RTC video streaming services have gained quite an importance recently. And several researchers have worked on improving QoE on RTC systems. In order to improve the performance of RTC streaming, heuristics and learning based algorithms have been developed so far.

In the network layer, there have been several improvements about congestion control solutions using RTC. Reno [57] and NewReno [58] proposed heuristic based algorithm which is named additive-increase-multiplicative-decrease (AIMD) which makes decisions based on packet loss. Cubic [59], Vegas [60] and BBR [61] also proposed congestion algorithms based on delay instead of packet loss. In addition, in BBRv2 [62] authors proposed new solutions for the initial version of their work to address fairness and network issues.

As the popularity of learning algorithms increases, more researchers are working on contributing to RTC systems with learning algorithms. Winstein *et al.* [63] proposed an algorithm named Remy, a distributed congestion control algorithm. Remy approached the congestion control problem as an optimization problem and tries to solve the congestion control problem with dynamic programming. PPC-Vivace [64] is another online learning algorithm to select the best sending rates. Another solution was using the imitation learning used in Indigo [65]. Also Deep Reinforcement Learning (DRL) techniques are being used widely in RTC systems nowadays. Similarly, Aurora [66], Orca [67] used DRL techniques and proposed new congestion control algorithms. NADA [68] and SCREAM [69] are also available congestion algorithms that can be used with RTC systems. The details of these congestion control algorithms and how they operate on packet loss and delay

can be found in the [70].

Fang *et al.* [71] proposed a learning algorithm to select the sending rate for RTC systems. Tianrun *et al.* [72] proposed a new learning model called Gemini, to estimate the bandwidth for RTC systems. Gemini uses Google Congestion Control (GCC) [73] with a learning model to select the bandwidth using a hybrid approach. However, based on our experiments, Gemini suffers to estimate the bandwidth accurately in challenging network conditions ending up with overestimation or underestimation in most of the cases. And in order to address these issues, we have developed a hybrid algorithm named BoB.

## 4.2 The BoB Algorithm

Bandwidth prediction in WebRTC plays an important role in user experience. Based on the predicted bandwidth value, WebRTC scales the sender resolution which affects the user experience throughout the video session. However the accuracy of the prediction is also playing a very important role; when the bandwidth prediction is underestimated, then the video resolution will be low quality and network resources will not be utilized no matter what the real network conditions are. If the bandwidth prediction is overestimated, then the playback will suffer from stalls, since the video will be up-scaled and will possibly require more than the available bandwidth. In order to solve this bandwidth measurement related adaptation problem, we have proposed a new hybrid based algorithm which we call BoB. This algorithm uses GCC [73] based heuristic and a DRL based learning modules which is explained below in detail. The overall architecture for training and testing is shown in Figure 18.

For model training we used AlphaRTC GYM simulator [74] using network traces collected from Belgium 4G/LTE [75], Norway 3G/HSDPA [76] and NYU LTE [77]. The simulator feedback receiving rate, packet loss and packet delay which is measured from the transport layer. We have implemented a bandwidth predictor using our model with these data supplied by the simulator environment.

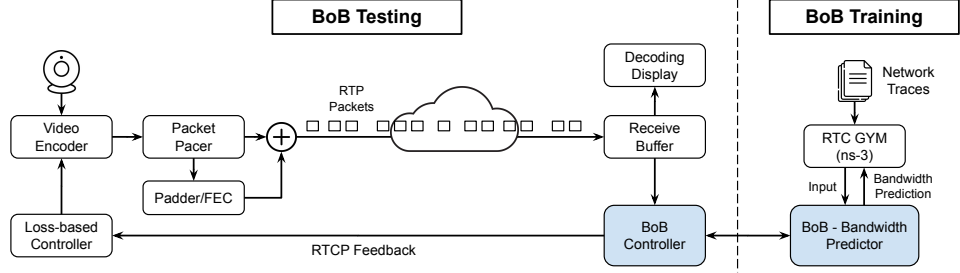


Figure 18: Overall workflow of BoB.

In the training phase of the model, we have used the Adam optimizer with the following equation:

$$\nabla R_t = \frac{1}{\Theta} \sum_{\theta=1}^{\Theta} \sum_{t=1}^{T_{\theta}} A_t^{\pi_{\theta}}(s_t, a_t) \nabla \log \pi_{\theta}(a_t, s_t),$$

where  $A^{\pi_{\theta}}(s_t, a_t)$  is the function that defines the variance in the reward after choosing the action  $a_t$  and the expected value.

On the receiver side of WebRTC, we have implemented a hybrid bandwidth estimator to increase the video streaming experience. The symbols and notations used in the algorithm are listed in Table 9.

Notation	Definitions
$c_t$	Receiving rate
$l_t$	Loss ratio
$n$	Last bandwidth prediction
$\vec{X}_t^r = \{x_{t-1}^r, x_{t-2}^r, \dots, x_{t-n}^r\}$	Bandwidth Samples
$T$	Total number of time windows
$W_t$	$t$ -th time window

Table 9: List of the key symbols and notations used in BoB.

At each interval  $W_t$ , the delay-based rate controller predicts the bandwidth  $x_t^r$  as shown in detail in Algorithm 7. The Kalman filter coefficients are picked as  $\beta = 1.08$  and  $\alpha = 0.85$ . The algorithm first uses the packet arrival filter that divides and groups the received packets into 200-ms slices and then computes the slope factor ( $m_t$ ) based on a delay gradient between the groups of received packets and decides the trend of the delay change. After that,  $m_t$  is fed to the adaptive threshold, which sets the threshold used by the overuse detector. Then, the overuse detector produces a signal that drives the network state (denoted by

$\tau$ ): *underuse*, *overuse* or *normal* based on  $m_t$  and threshold. The network state is then mapped to a controller state *increase*, *decrease* or *hold*. If the controller state is *decrease*, then the controller sets the rate control *region* to state NearMax. Once the controller state is changed to *increase* and the rate control *region* is in state NearMax, the controller sets  $\bar{x}_t = c_t$ . Otherwise, if the controller state is *increase* and the rate control *region* is in the state of MaxUnknown, the controller sets  $\bar{x}_t = \beta \times c_t$ . Therefore, the controller additively increases  $x_t^r$  based on the rate control *region*.

With BoB, we have implemented a reinforcement learning technique that interacts with the environment and the sender-receiver communication mechanism in the RTC system. For the training part for BoB, the packet-level statistics are periodically gathered over a predetermined time frame of  $W_t = 200 \text{ ms}$  and aggregated as the environment state. The agent then makes a prediction about the bandwidth for an action value. Formally, the learning agent communicates with the environment that creates the state space represented by the symbol  $\mathcal{S}$ . The RL agent receives a state  $s_t \in \mathcal{S}$  from the environment at each time window  $W_t$  (at time epoch  $t$ ) and then performs an action  $a_t \in \mathcal{A}$  (bandwidth prediction for the next time window  $W_{t+1}$ ) while earning a reward  $r_t \in \mathcal{R}$ .

The agent's main goal is to identify the best possible policy  $\pi^* : \mathcal{S} \rightarrow \mathcal{A}$  that maps states to actions and optimizes total reward (*i.e.*, determining the bandwidth that maximizes reception rate while reducing packet loss and delay). Following the execution of the bandwidth prediction action  $a_t$ , the bob environment monitors the new receiving rate, packet loss, latency, and the expected bandwidth before moving on to the next mathematical state  $s_{t+1} \in \mathcal{S}$  and updating the reward  $r_{t+1} \in \mathcal{R}$ . The DRL controller designed with BoB is shown in Figure 19.

We have implemented an adaptive selector algorithm shown in Algorithm 8 to implement the hybrid approach to decide when to switch between the heuristic and learning-based rate controllers. We allow a hybrid bandwidth prediction using this functionality, which also improves the DRL controller's long-term accuracy.

---

**Algorithm 7** Delay-based Rate Controller

---

```
1: function HEURISTICCONTROLLER( $c_t, d_t, l_t, X_t^r$ )
2:    $\alpha \leftarrow 0.85, \beta \leftarrow 1.08, region \leftarrow \text{MaxUnkown}$ 
3:   for Each time window  $W_t, W_t > 0$  do ▷ every 200 ms
4:      $\sigma \leftarrow \text{GetControllerState}()$  ▷ overuse detector
5:      $\hat{c}_t \leftarrow \text{std}(C_t)$  ▷ standard deviation of  $C_t$ 
6:      $\bar{c}_t \leftarrow \text{Average}(C_t)$  ▷  $C_t = \{c_t, c_{t-1}, \dots, c_{t-n-1}\}$ 
7:     if ( $\sigma == \text{'Increase'}$ ) then
8:       if ( $c_t > \bar{c}_t + 3 \times \hat{c}_t$ ) then
9:          $\tau \leftarrow \text{'Underuse'}$  ▷  $\tau$ : network state
10:         $region \leftarrow \text{MaxUnkown}$ 
11:      end if
12:      if ( $\tau == \text{'Underuse'}$ ) then
13:        if  $region == \text{MaxUnkown}$  then
14:           $\bar{x}_t \leftarrow \beta \times c_t$ 
15:           $x_t^r \leftarrow x_{t-1}^r + \bar{x}_t$ 
16:        end if
17:        if  $region == \text{NearMax}$  then
18:           $\bar{x}_t \leftarrow c_t$ 
19:           $x_t^r \leftarrow x_{t-1}^r + \bar{x}_t$ 
20:        end if
21:      end if
22:    end if
23:    if ( $\sigma == \text{'Decrease'}$ ) then
24:       $x_t^r \leftarrow \alpha \times c_t$ 
25:       $x_t^r \leftarrow \text{Min}(x_t^r, \alpha \times x_{t-1}^r)$ 
26:       $region \leftarrow \text{NearMax}$ 
27:    end if
28:    if ( $\sigma == \text{'Hold'}$ ) then
29:       $x_t^r \leftarrow x_{t-1}^r$ 
30:    end if
31:     $\tau \leftarrow \text{'Underuse'}$ 
32:    Return( $x_t^r$ )
33:  end for
34: end function
```

---

At the beginning of a session, when the values given from the DRL controller are mostly related to the training dataset, bandwidth prediction is likely to be wrong due to bandwidth underprediction caused by a lack of data.

To overcome this possible inaccuracy, we compare the prediction results obtained from the DRL controller with those from the heuristic controller and validate their accuracy. To do so, we use the symmetric mean absolute percentage error (sMAPE) function. First, we compute the absolute difference ( $Diff_t$ ) between



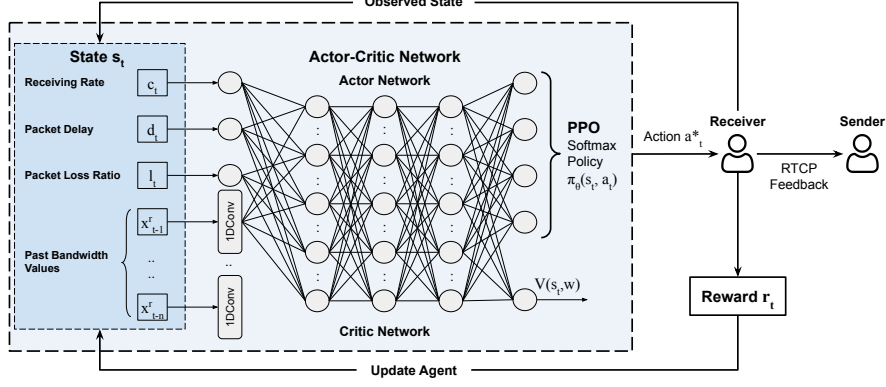


Figure 19: Learning-based (DRL) rate controller for BoB.

the predicted bandwidth values given by the heuristic controller ( $Heuristicbw_t$ ) and the DRL controller ( $DRLbw_t$ ). Then, we compute the average predicted bandwidth value ( $Avg_t$ ) of the both controller values. If the output from the percentage ( $\frac{Diff_t}{Avg_t} \geq 30\%$ ), the algorithm decides not to use the DRL controller and feeds the output of the heuristic controller to the DRL controller for later use. Over time, the DRL controller begins to forecast outcomes more accurately as the difference between the two controllers' output percentages decreases. Algorithm 8 highlights the crucial steps of the adaptive selector. We should point out that this algorithm additionally keeps track of the variance between DRL and heuristic controllers in the event of departures (corner cases) from the anticipated convergence of predictions from both controllers. It returns to the heuristic controller if there is a deviation. Under some network conditions, we did not frequently notice this circumstance. In the long run, the DRL controller maintains operating well once it gets going. We point out that the choice to set the percentage at 30% when choosing which controller to use is based on our tests with different network conditions.

### 4.3 Evaluation

The network profiles we used in the evaluation, re-purposed for this work from [0], are shown in Figure 20. The profiles are extracted randomly from 20% of network traces assigned for testing, namely: LTE, Twitch, Cascade, FCC Amazon and

---

**Algorithm 8** Adaptive Selector

---

```
1: function ADAPTIVSELECTOR
2:   for Each time window  $W_t, W_t > 0$  do
3:      $Heuristicbw_t \leftarrow \text{HeuristicController}(c_t, d_t, l_t, X_t^r)$ 
4:      $DRLbw_t \leftarrow \text{DRLController}(c_t, d_t, l_t, X_t^r)$ 
5:      $Dif_t = |DRLbw_t - Heuristicbw_t|$ 
6:      $Avg_t = \frac{DRLbw_t + Heuristicbw_t}{2}$ 
7:
8:     if  $\frac{Dif_t}{Avg_t} \geq 0.3$  then
9:        $x_t^r \leftarrow Heuristicbw_t$ 
10:    else
11:       $x_t^r \leftarrow DRLbw_t$ 
12:    end if
13:  end for
14:  Return( $x_t^r$ )
15: end function
```

---

Synthetic. For FCC Amazon and Synthetic, we fixed the delay to 50 ms and loss to 0.08%.

Figure 21 shows the time series plots for several solutions for each network profile. First two columns of Table 10 show the overall average bandwidth prediction accuracy and prediction error in terms of sMAPE. In Figure 21, the actual bandwidth for the network profiles is shown as red solid lines. A better solution needs to choose a bandwidth that is close to these solid lines. Overall, we notice that BoB achieves the best bandwidth prediction accuracy compared to its competitors. Especially, BoB improves the overall average bandwidth prediction accuracy by 67.63% (Cascade: 61.72%, LTE: 41.71%, Twitch: 81.64%, FCC Amazon: 73.27%, Synthetic: 79.80%), and reduces the overall average bandwidth prediction error by 49.11% (Cascade: 38.95%, LTE: 46.45%, Twitch: 62.14%, FCC Amazon: 71.07%, Synthetic: 26.94%) compared to the other solutions across all the network profiles. However, only in the Cascade profile, HRCC is slightly better than BoB in terms of the average bandwidth prediction accuracy with a marginal improvement of 0.19%. Therefore, HRCC was able to achieve a better receiving rate, network, video and total scores compared to BoB in the Cascade profile.

Looking at the results further, BoB is able to achieve a higher average receiving

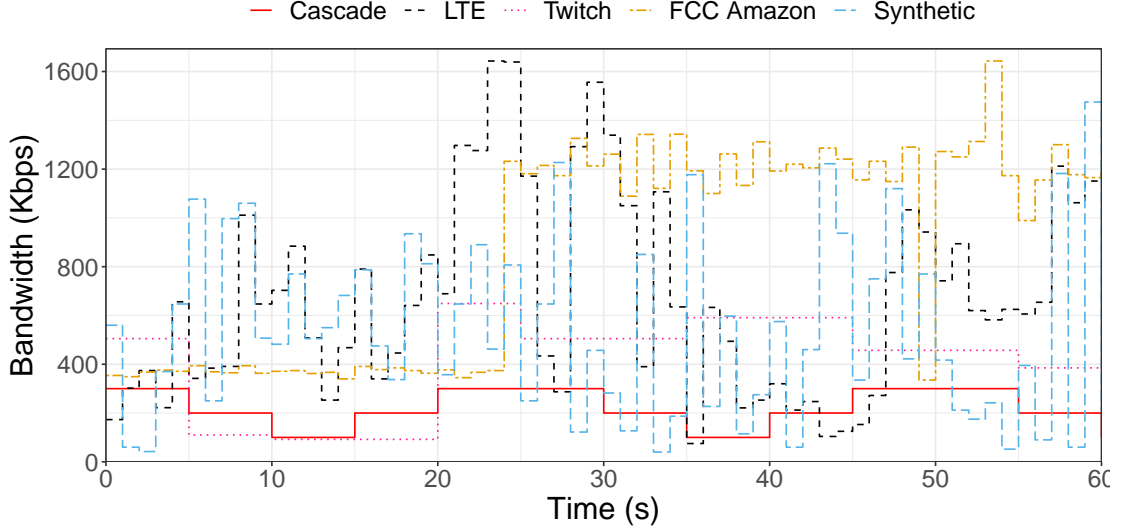


Figure 20: The network profiles used in the simulations.

rate score of 73.64%, compared to Gemini (45.28%), HRCC (54.77%) and Heuristic (31.78%) across all the network profiles. This comes at a price of a smaller delay score and a comparable loss score for BoB in each network profile. In fact, BoB’s formulation strives to ensure a good trade-off between these conflicting metrics (*i.e.*, receiving rate, delay and packet loss). Such a trade-off by BoB is confirmed through achieving higher network, video and total scores with an improvement of 35.85% (Cascade: 29.21%, LTE: 31.53%, Twitch: 58.00%, FCC Amazon: 45.77%, Synthetic: 14.74%), 5.90% (Cascade: 6.41%, LTE: 4.59%, Twitch: 6.73%, FCC Amazon: 7.40%, Synthetic: 4.37%), and 23.03% (Cascade: 18.38%, LTE: 18.95%, Twitch: 36.19%, FCC Amazon: 31.49%, Synthetic: 10.12%), respectively, compared to its competitors across all five network profiles.

Compared to BoB, we also observe that other solutions generally suffer from either bandwidth overprediction or underprediction due to their designs. As shown in Figure 21, Gemini tends to underutilize the bandwidth, which expectedly produces a low receiving rate score but also a higher packet delay and loss score than BoB and HRCC. This happens because Gemini fails to timely switch between the learning and heuristic-based prediction. For example, for the FCC Amazon profile (see Figure 21d), the learning-based prediction for Gemini fails to track the increase in the actual bandwidth. Similarly, HRCC generally fails

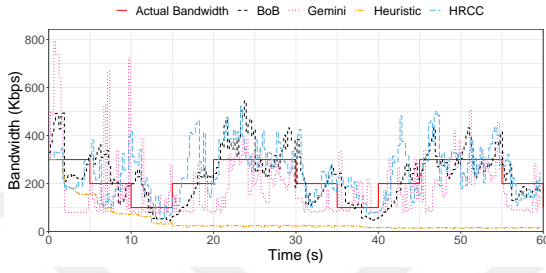
to learn suitable parameters for the heuristic-based algorithm, which leads to a bandwidth underprediction issue for various network profiles, which is most visible in Figs 21b, 21c and 21d. As a result, HRCC suffers from poor video quality. This also confirms that HRCC is more suitable for more stable, low-bandwidth scenarios.

Also, one of the important findings we have observed is that the Heuristic method is unable to compensate for bandwidth underestimations during the entire RTC session, which lowers the quality of the video (see the score results in Table 10). This result demonstrates how crucial it is to have a hybrid solution that involves both learning and heuristic-based methods, as well as how difficult and important bandwidth prediction is in the RTC [78]. BoB, on the other hand, successfully combines both techniques and attempts to forecast the bandwidth within a tiny margin of its actual value during the RTC session. BoB also performs admirably over a variety of network characteristics.

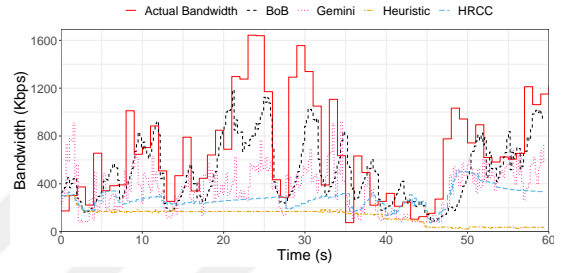
BoB has been integrated into AlphaRTC and the results show the superiority of BoB for bandwidth prediction in RTC. BoB achieves up to 15.62% and 27.87% better bandwidth prediction accuracy than Gemini and HRCC (the winning and runner-up solutions, respectively, in the ACM MMSys'21 grand challenge)

Table 10: Average simulation results for different network profiles ( $\uparrow$ : higher is better,  $\downarrow$ : lower is better).

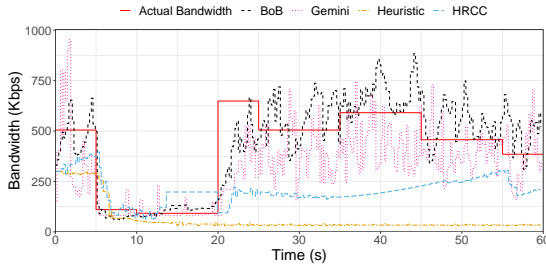
	Avg. Prediction Accuracy (%)	Avg. Prediction Error (sMAPE)	Avg. Receiving Rate Score (%)	Avg. Delay Score (%)	Avg. Loss Score (%)	Avg. Network Score (%)	Avg. Video Score (%)	Avg. Total Score (%)
<b>Cascade</b>								
BoB	84.89	<b>0.30</b> $\downarrow$	63.26	15.71	95.45	42.75	91.38	70.16
Gemini	74.85	0.50	46.56	35.78	97.64	36.62	87.73	62.94
HRCC	<b>85.06</b> $\uparrow$	<b>0.30</b> $\downarrow$	<b>63.91</b> $\uparrow$	21.27	91.69	<b>43.25</b> $\uparrow$	<b>92.23</b> $\uparrow$	<b>70.92</b> $\uparrow$
Heuristic	31.22	1.38	22.36	<b>37.22</b> $\uparrow$	<b>99.38</b> $\uparrow$	24.84	78.79	48.48
<b>LTE</b>								
BoB	<b>78.94</b> $\uparrow$	<b>0.42</b> $\downarrow$	<b>56.98</b> $\uparrow$	32.05	88.57	<b>40.55</b> $\uparrow$	<b>92.59</b> $\uparrow$	<b>68.33</b> $\uparrow$
Gemini	68.07	0.64	33.35	71.31	99.01	33.71	88.85	60.36
HRCC	63.65	0.73	35.77	38.00	97.51	31.44	91.82	58.98
Heuristic	42.63	1.15	20.39	<b>77.56</b> $\uparrow$	<b>99.66</b> $\uparrow$	27.92	85.17	53.47
<b>Twitch</b>								
BoB	<b>88.84</b> $\uparrow$	<b>0.22</b> $\downarrow$	<b>94.87</b> $\uparrow$	52.11	92.29	<b>61.88</b> $\uparrow$	<b>92.80</b> $\uparrow$	<b>89.72</b> $\uparrow$
Gemini	82.85	0.34	60.39	45.24	<b>98.49</b> $\uparrow$	44.57	91.12	71.91
HRCC	65.92	0.68	74.74	<b>54.57</b> $\uparrow$	97.06	52.53	91.50	79.98
Heuristic	29.33	1.41	31.77	29.81	95.98	28.46	79.36	52.27
<b>FCC Amazon</b>								
BoB	<b>86.60</b> $\uparrow$	<b>0.27</b> $\downarrow$	<b>100</b> $\uparrow$	60.02	97.17	<b>65.72</b> $\uparrow$	<b>95.19</b> $\uparrow$	<b>94.28</b> $\uparrow$
Gemini	63.91	0.72	55.09	82.56	99.71	45.77	88.14	72.21
HRCC	50.42	0.99	60.36	79.52	99.84	48.11	91.46	75.55
Heuristic	40.74	1.19	44.91	<b>93.72</b> $\uparrow$	<b>100</b> $\uparrow$	41.83	86.45	67.76
<b>Synthetic</b>								
BoB	<b>65.88</b> $\uparrow$	<b>0.68</b> $\downarrow$	<b>53.08</b> $\uparrow$	28.58	96.82	<b>39.08</b> $\uparrow$	<b>92.59</b> $\uparrow$	<b>66.85</b> $\uparrow$
Gemini	62.15	0.76	30.99	65.77	97.94	31.87	84.08	57.09
HRCC	60.43	0.79	39.06	42.28	97.56	33.52	92.11	61.15
Heuristic	20.31	1.59	39.49	<b>75.32</b> $\uparrow$	<b>99.42</b> $\uparrow$	37.22	90.35	64.32
<b>Avg. ALL</b>								
BoB	<b>81.03</b> $\uparrow$	<b>0.38</b> $\downarrow$	<b>73.64</b> $\uparrow$	37.69	94.06	<b>50.00</b> $\uparrow$	<b>92.91</b> $\uparrow$	<b>77.87</b> $\uparrow$
Gemini	70.37	0.59	45.28	60.13	98.56	38.51	87.98	64.90
HRCC	63.36	0.73	54.77	47.13	96.73	41.77	91.82	69.32
Heuristic	32.84	1.34	31.78	<b>62.73</b> $\uparrow$	<b>98.89</b> $\uparrow$	32.05	84.02	57.26



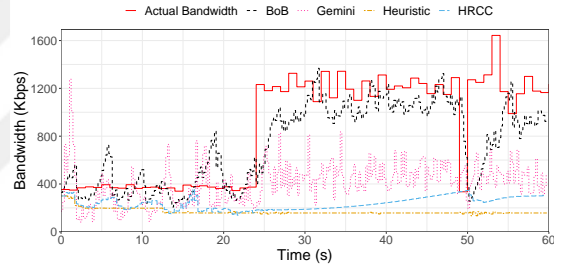
(a) Profile: Cascade.



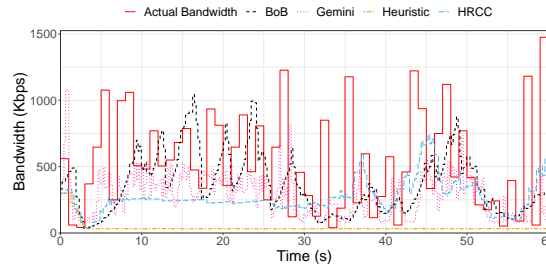
(b) Profile: LTE.



(c) Profile: Twitch.



(d) Profile: FCC Amazon.



(e) Profile: Synthetic.

Figure 21: Actual and predicted bandwidth for different network profiles.

## CHAPTER V

### ALGORITHMS FOR 360-DEGREE VIDEO STREAMING

As the popularity of immersive media increases nowadays, for 360-degree video streaming there has been many researches to improve the QoE of streaming sessions by using viewport dependent streaming (VDS). Viewport-dependent streaming (VDS) enables the delivery of the viewport in high quality while serving the surrounding (background) content in lower quality compared to the viewport. It is mentioned in recent researches [79, 80, 12, 81, 82, 83, 84, 85] that VDS can reduce bandwidth consumption by 30-70%. However, when there is a head motion that changes the viewport tiles, the user may view low-quality content until all the new viewport tiles are downloaded and rendered in high quality. Viewport prediction techniques can be used to predict future incoming tiles and proactively download these tiles correlated with the head motion and reduce the latency of the viewport updates [86, 87]. Viewport predictions can also be used to reduce the bandwidth consumption which is used in [88]. Authors created Multi-View Coding (MVC) compliant server and used viewport prediction to reduce the bandwidth consumption, where a communication between client and server introduced in their work which requires an intelligent server encoding. Viewport changes due to head motion also mean that some of the downloaded high-quality tiles do not get viewed by the user resulting in bandwidth waste. One study observed that up to 25% of streaming data waste could be prevented with accurate head trajectory predictions [89]. Several studies using learning-based approaches for viewport prediction and tile bitrate selection were presented in [90, 91, 92]. Similarly, the authors of [93] assigned a weighted portion of the available bandwidth to each

tile based on its distance from the viewport to address the problem of multiplicity in bandwidth allocation for tiled video (quality must be assigned for each tile instead of the whole video). An adaptive 360-degree solution for mobile devices with viewport prediction was proposed in [94]. The algorithm fetches a subset of the tiles, incurring stalls of up to 0.96 seconds per playback minute when a tile in the visible region is missing. A comprehensive study on tile selection methods for VDS is provided in [95].

### ***5.1 Viewport-Dependent Streaming with Concurrent Segment Downloads***

As discussed previously, briefly VDS is a technique used in downloading the tiles in separate qualities for viewport and background tiles. And in order to do that we need to create multiple segment files for each tile. For example for an 8x6 tile configuration, we need to generate 48 separate files for each segment giving the ability to ABR to download each of the tiles in different encoding quality. And yet again measuring the bandwidth plays a very important role here to increase/decrease viewport quality or utilize downloaded tiles. For low-latency in traditional ABRs we had a similar challenge which we overcame by parsing moof boxes. For 360-degree VDS, we have a similar challenge but here we have another problem with concurrency. We designed and tested our systems using HTTP/1.1 and H2 (with multiplexing) which creates concurrent requests and responses at the same time. In order to solve this problem, we have designed a circular buffer to measure the bandwidth created by concurrent threads shown in Figure 22. Each thread saves the time request sent and passes bytes received to the circular buffer when a response to the request arrives from the server. The circular buffer first tries to find an appropriate buffer to append that new chunk information to calculate the overlapping transfers together to calculate the bandwidth accurately. It checks the existence of a buffer that has an end time later than the request time and start time before the response receive time as shown in Figure 22. If that buffer is found then that chunk data information is merged to this buffer by extending



start time or end time and adding the byte information to the buffer. Otherwise, if the buffer is not found the next buffer from the circular buffers is used.

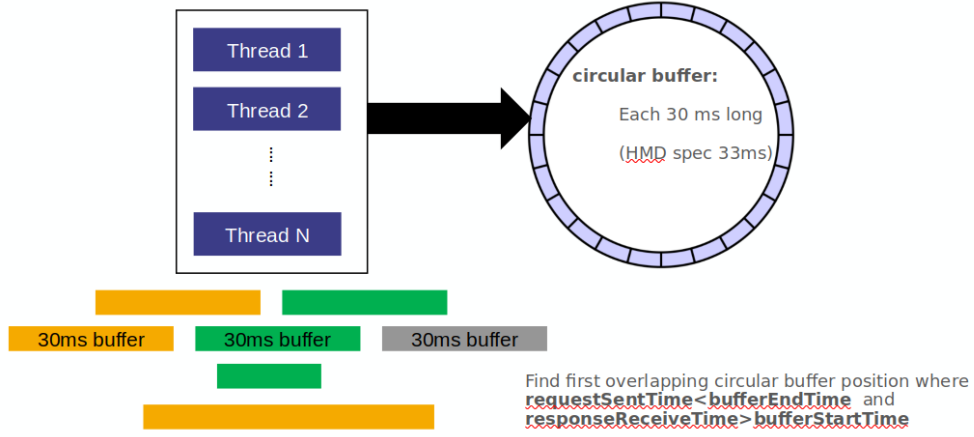


Figure 22: Bandwidth measurement for concurrent dash threads.

After saving all these circular buffers the bandwidth is calculated for all of the buffers and the value having the maximum value is used. Having that maximum value is the one which has the smallest idle periods, that is why this value is picked as the bandwidth representation.

## 5.2 HTTP/1.1 vs. H2

Regarding the need of concurrent segment downloads for each tile representation in VDS, it is also important to implement and evaluate our solution for both HTTP/1.1 and H2. Since human head motions can be varying a lot during the streaming session with the HMD device, it is very likely to make a change in the middle of downloading a future viewport or margin tile. However, HTTP/1.1 does not have the capability of cancelling a request previously sent to the server which will cause unnecessary network in case of those sudden head movements.

As a solution to this problem, we have implemented H2 and implemented the abortion of the unnecessary requests that would probably create an unnecessary bandwidth consumption. Another improvement with H2 is the ability to use the same connection for multiple requests which is called multiplexing. Multiplexing

enables us not to hit the client player resource boundaries like maximum connection and thread configurations on the physical machine, especially for 12x8 tile configuration. In HTTP/1.1 the client should be capable of handling 96 connections with the server simultaneously. However, with H2 it is possible to create a single connection and request different data using the same connection. The architecture we used in H2 implementation is shown in Figure 23.

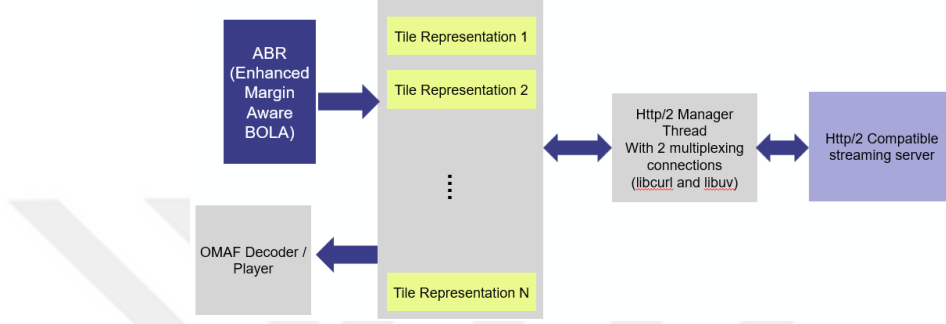


Figure 23: H2 architecture.

Even though just a single connection was enough to use multiplexing, we have created two connections for a robust solution in case of possible connection issues. Having an http manager in this architecture, has another advantage for the download requests trying to get the network resource. Assuming low quality tiles are smaller in size compared to high quality tiles, it is very likely that the download of low quality tiles for a particular segment  $i$  will finish earlier than the high quality viewport tile for the same segment. In this case, it would be a bad resource allocation if we start the download of low quality tile for segment number  $i + 1$  before whole segment downloads for  $i$  are completed. In our implementation, the http manager thread in the middle also respects the order of the requests and solves this problem by managing the multiplexing connection handlers.

### 5.3 Head-Motion-Awareness for 360-Degree Videos

Head motion data coming from the HMD devices in 360-degree video can help improve the user experience and lower the MTHQD metric discussed earlier. By using head motion data we might predict the tiles that will be part of the viewport

in the future and start downloading these tiles ahead of time. We call these tiles that we start downloading a priori as margin tiles.

### 5.3.1 Margin-Aware ABR

It is important to use the head motion data to predict the possible margin tiles that will very likely be inside the viewport which we call a margin hit. In Algorithm 9, the calculation of possible margins is shown. The head motion is read from an HMD device with  $\approx 33ms$  intervals. Since human head motions are not robotic the data could have back and forth data. That is why those data are saved in a sliding window with a size of 20 and the average head motion is fed to the *getPrediction* method in the algorithm in each sampling interval coming from HMD. Based on the motion vectors, a very basic x and y prediction is calculated with the *getPrediction* method. *getPrediction* basically multiplies the motion vector with  $2 \times \text{maximum tile width or height}$ . Then for each tile in the background it is adding them to *marginsAdded* array if they are in the motion direction and less than the predicted distance or if the user is stationary ( $motion_x = 0$ ) it will add one tile distance from both sides of the coordinate system. For the tiles in the vertical direction the algorithm combines all viewport tiles with the horizontal tiles and then adds the vertical tiles which are closer than the  $pred_y$  value or  $\text{tileHeight}$  if the user is stationary ( $motion_y = 0$ ).

While all possible margin tiles are being computed in Algorithm 9, *addMarginTile* function is called where the prioritization of the margin tiles is managed inside that function defined in Algorithm 10.

While using the head motion data and trying to calculate the margin tile priorities in Algorithm 10 for the same direction with the head motion, it is important to remember again that we are traveling over a sphere and 2D motion approximation might mislead us especially for the corner tiles. In 2D space the tiles on the corners might look far away from the center where it might be closer than the tiles that are closer to the viewport center in the 2D coordinate system. In Figure 24 circular distance calculation is shown for two points which is also known

---

**Algorithm 9** Find Possible Margin Tiles

---

```
1: function FINDPOSSIBLEMARGINS
2:    $X, marginsAdded \leftarrow \emptyset$ 
3:    $motion_x \leftarrow \Delta HMD_x$ 
4:    $motion_y \leftarrow \Delta HMD_y$ 
5:    $\{pred_x, pred_y\} \leftarrow getPrediction(motion_x, motion_y)$ 
6:   for each background  $tile_i$  do
7:      $d \leftarrow closestHorizontalDistance(viewport, tile_i)$ 
8:     if ( $motion_x = 0$  and  $|d| < tileWidth$ ) or ( $sign(motion_x) = sign(d)$ 
and  $|d| < pred_x$ ) then
9:        $addMarginTile(tile_i, marginsAdded)$ 
10:       $marginsAdded \leftarrow marginsAdded \cup tile_i$ 
11:    end if
12:  end for
13:  for each tile  $tile_i$  in viewport tiles  $\cup$  marginsAdded do
14:     $X \leftarrow X \cup tile_{i,x}$ 
15:  end for
16:  for each background  $tile_i$  do
17:    if  $tile_{i,x} \in X$  then
18:       $d \leftarrow closestVerticalDistance(viewport, tile_i)$ 
19:      if ( $motion_y = 0$  and  $|d| < tileHeight$ ) or ( $sign(motion_y) =$ 
 $sign(d)$  and  $|d| < pred_y$ ) then
20:         $addMarginTile(tile_i, marginsAdded)$ 
21:      end if
22:    end if
23:  end for
24:  return  $marginsAdded$ 
25: end function
```

---

as haversine [96] distance. This haversine distance on the sphere is calculated and used in Algorithm 10.

Circular (haversine) distance calculation of two given points on the sphere with longitude and latitude values are shown in Equation 5. The longitude and latitude values in the equation are the azimuth and elevation, respectively.

$$\alpha = \sin^2\left(\frac{lat_1 - lat_2}{2}\right) + \cos(lat_1) \times \cos(lat_2) \times \sin^2\left(\frac{lon_1 - lon_2}{2}\right) \quad (5)$$
$$c = 2 \times \arctan2(\sqrt{\alpha}, \sqrt{1 - \alpha})$$

In Algorithm 10 the circular distance from the viewport center to the margin tile that is being added ( $tile_i$ ) is calculated. The angle is the angle between the motion vector and the vector starting from the viewport center and ending in the

---

**Algorithm 10** Add Possible Margin Tile

---

```
1: function ADDMARGINTILE( $tile_i$ ,  $possibleMargins$ )
2:    $distance \leftarrow getCircularDistanceFromViewport(tile_i)$ 
3:    $angle \leftarrow |getAngularDistanceFromMotion(tile_i)|$ 
4:    $area \leftarrow getViewportIntersection(tile_i)$ 
5:    $possibleMarginsSize \leftarrow getPossibleMarginsSize()$ 
6:    $j \leftarrow 0$ 
7:   if  $angle > \frac{\pi}{4}$  then
8:     for  $j$  to  $possibleMarginsSize$  step 1 do
9:        $tile \leftarrow possibleMargins_j$ 
10:       $tileAngle \leftarrow getTileAngle(tile)$ 
11:      if  $tileAngle \geq angle$  then
12:        break;
13:      end if
14:    end for
15:  end if
16:  for  $j$  to  $possibleMarginsSize$  step 1 do
17:     $tile \leftarrow possibleMargins_j$ 
18:     $tileAngle \leftarrow getTileAngle(tile)$ 
19:     $tileDistance \leftarrow getTileDistance(tile)$ 
20:    if  $tileAngle < area$  or  $tileDistance > distance$  then
21:      break;
22:    end if
23:  end for
24:   $possibleMargins_j \leftarrow tile_i$ 
25: end function
```

---

center of  $tile_i$ . Angular distance for a tile (red color) is shown in Figure 25. The angle between the head motion vector and the margin tile is calculated and used as input to prioritization in Algorithm 10. Similarly, the area of the margin tile that is being added to the margins array is calculated in the algorithm.

In Algorithm 10 firstly the angular distance in radians ( $angle$ ) is checked if it is bigger  $\frac{\pi}{4}$ , then in the possibleMargins array that new  $tile_i$  is placed in the correct spot in ascending order based on the angle in the array. If the angle is less than  $\frac{\pi}{4}$ , considering it might be in both directions, an area of  $\frac{\pi}{2}$  is used as a threshold to accept the  $tile_i$  in the same direction with the motion. Tiles with an angle less than  $\frac{\pi}{4}$  are considered as the same and high priority in terms of angular distance. After angle prioritization is checked the algorithm then checks the circular distance.  $getCircularDistanceFromViewport$  function calculates the

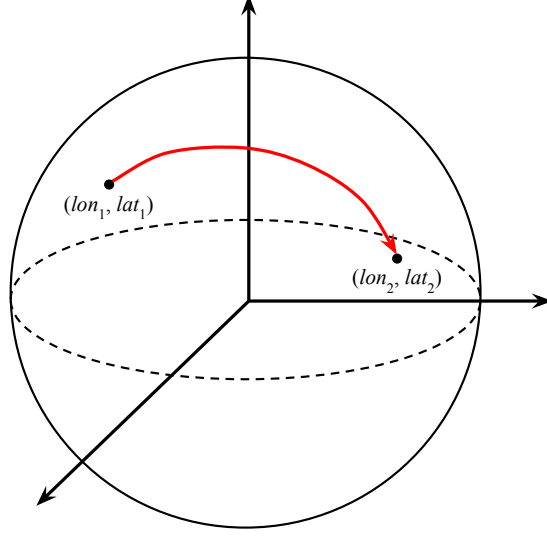


Figure 24: Haversine (circular) distance.

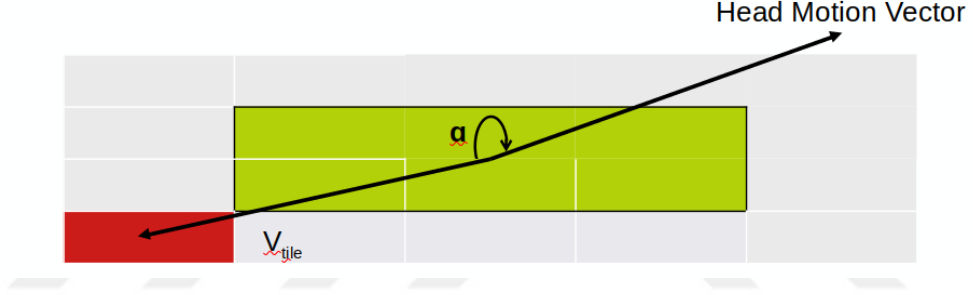


Figure 25: Angular distance between a tile (red) and head motion vector.

circular distance defined in Equation 5. In this second loop the circular distance and angular distance are checked together. If a shorter distance or closer angular distance location is found then the algorithm puts the given  $tile_i$  in that found spot. Angular distance and circular distance are used together to put the closest motion aware margin tiles at the top of the returning *possibleMargins* array.

After calculating and prioritizing margins based on the head motion and position of the margin tile, these calculated margin tiles are consumed by the ABR with the Algorithm 11. This algorithm is basically calculating the 30% of the current viewport and adding from these calculated possibleMargins to the viewport margins and rendering them in high quality.

In our experiments, we used both human head motions and auto generated head motions to show the speed effect to our system. Because when the head speed

---

**Algorithm 11** Get Viewport Margin Tiles

---

```
1: function GETVIEWPORTMARGINS
2:    $marginsPercentage \leftarrow 30\%$ 
3:    $totalPercentage \leftarrow 0$ 
4:    $viewportMargins \leftarrow \emptyset$ 
5:    $possibleMargins \leftarrow findPossibleMargins()$ 
6:    $possibleMarginsSize \leftarrow getPossibleMarginsSize()$ 
7:   for  $i := 0$  to  $possibleMarginsSize$  step 1 do
8:     if  $totalPercentage < marginsPercentage$  then
9:        $margin \leftarrow possibleMargins_i$ 
10:       $viewportMargins \leftarrow viewportMargins \cup margin$ 
11:       $p \leftarrow area(margin)/area(viewport)$ 
12:       $totalPercentage \leftarrow totalPercentage + p$ 
13:    else
14:      break;
15:    end if
16:  end for
17:  return  $viewportMargins$ 
18: end function
```

---

is increasing then the possibility of hitting a background tile is increasing, so it makes sense to investigate varying head speeds starting from 25 degree per second (dps) to 120 dps with an auto generated 60 seconds long head motion which is shown in Figure 26. In this head motion, we have selected diagonal movements by purpose to challenge our proposed algorithm. When moving in diagonal direction VDS will hit more tiles partially which will challenge our algorithm to make a good prioritization based on circular and angular distances.

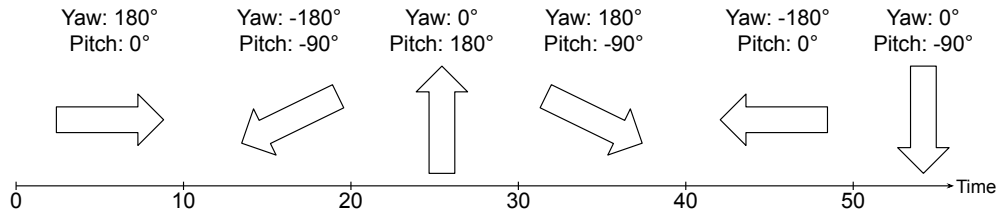
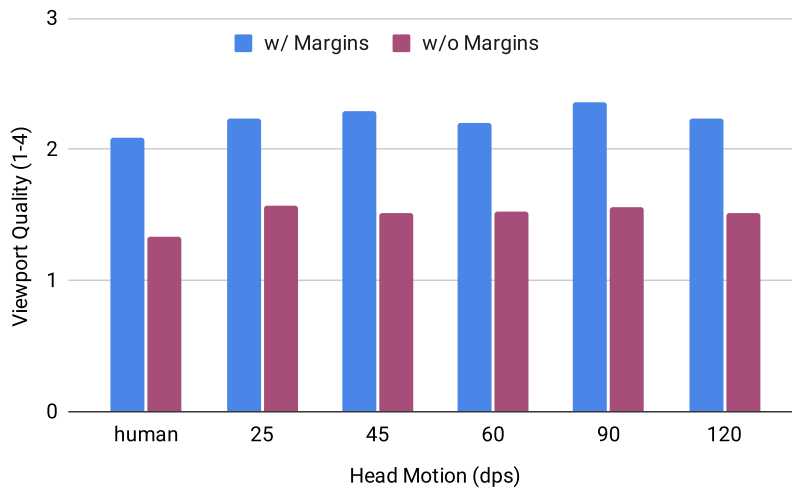


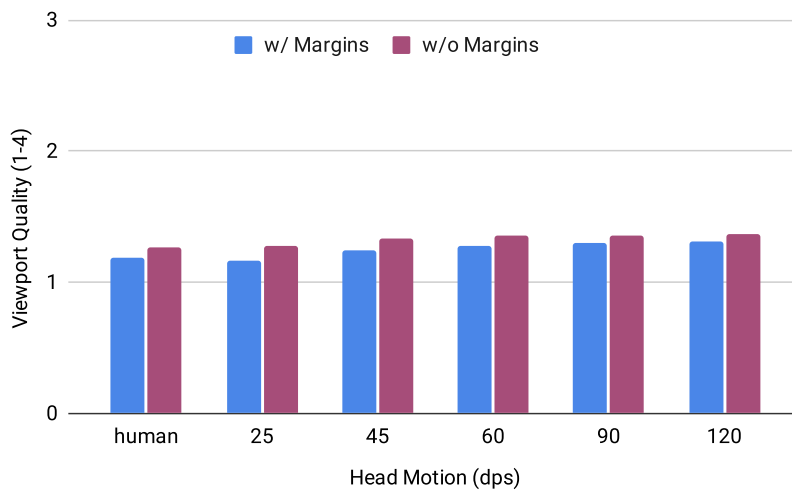
Figure 26: Automatic generated head motions used in experiments.

The viewport quality measurement as discussed in the Section 1.3.2 calculated by considering the tile viewport intersection area. The viewport quality is ranging between 1 and 4 where 1 is the highest quality and 4 is the lowest (background) quality. We have tested our solution with two bandwidth conditions 40 Mbps and 60 Mbps. We have chosen 40 Mbps by purpose since the bandwidth here will not

be enough to download all tiles in the viewport and margin in highest quality. Because margins will create an extra download overhead and we wanted to see the impact of that in the viewport. As it can be seen from Figure 27a the average viewport quality is decreased with margins since the download overhead created an issue our ABR decreased the viewport quality to one level down not to create a stalling or stop during the playback. Since in 60 Mbps we don't have a resource issue it is clearly seen from Figure 27b that viewport quality is improved with the margins.



(a) 40 Mbps



(b) 60 Mbps

Figure 27: Average viewport quality at 40 (top) and 60 Mbps (bottom) bandwidth. Lower values indicate better quality.



In Figure 28 the MTHQD is shown for *Trolley*(top row) and *Harbor*(bottom row) sequences for 40 Mbps. For the trolley video, as it can be seen, margins help decrease the MTHQD, especially for 6x4 tile configuration. The reason for the improvement being bigger for smaller tile configuration is that the tile sizes are bigger and the total number of the tiles being smaller makes the effect of margins more eminent. When the tile configuration increases to 8x6 and 12x8, the tile switching and quality change numbers increase which affects the overall system as shown in the results. The bottom results are for the Harbor sequence which is a more dynamic test sequence and has faster movements. Since the content is more dynamic here we still have improvements however the improvement is less compared to the trolley sequence due to the fast changing environment conditions.

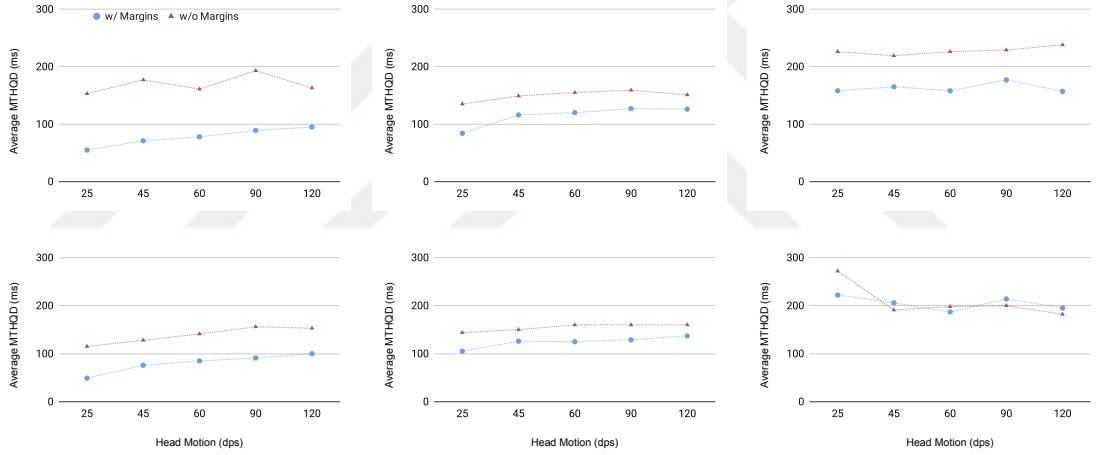


Figure 28: Average MTHQD results for the *Trolley* (top) and *Harbor Biking* (bottom) sequences at 40 Mbps bandwidth and tile configurations of 6x4 (left), 8x6 (middle) and 12x8 (right).

The best visible improvement happens in 6x4 tile configurations in Figure 28. Another important point in the results is the tile configuration impact on the content selection. According to our observations, 8x6 tile configuration is not affected too much by the content selection and it is more stable based on the other tile configuration. On the other hand, 12x8 tile configuration is the most fragile in terms of content selection based on our observations.

In Figure 29, 60 Mbps MTHQD results are shared for each tile configuration.

Since the bandwidth is more than enough in this case, the selected test sequence Trolley or Harbor doesn't play an important role. The benefit of using margins can be seen from the MTHQD decreases in all of the graphs for different tile configurations. Another important point that we can see in Figure 29 is that the improvement in MTHQD is more eminent in slow head motions. When the head speed is more than 90 dps the improvement in the MTHQD also diminishes. This pattern is more visible in 12x8 tile configuration.

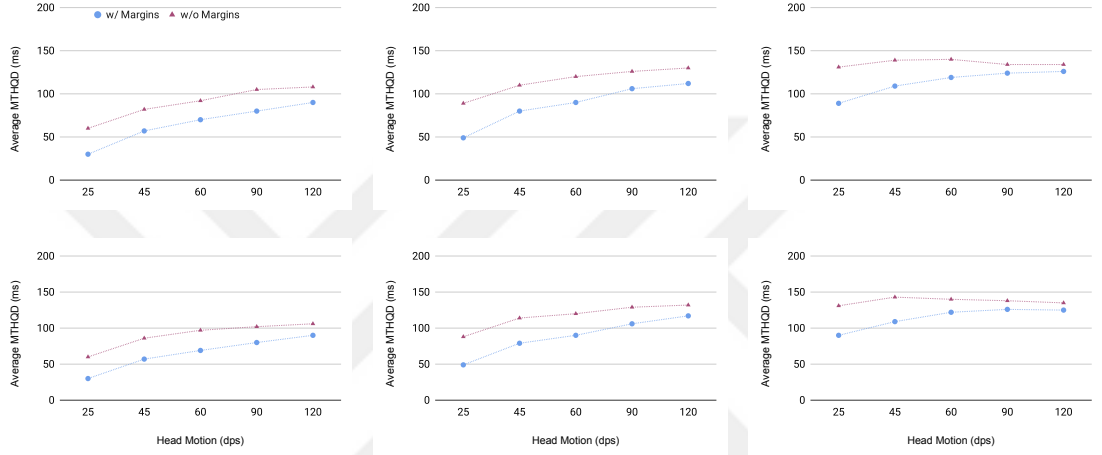


Figure 29: Average MTHQD results for the *Trolley* (top) and *Harbor Biking* (bottom) sequences at 60 Mbps bandwidth and tile configurations of 6x4 (left), 8x6 (middle) and 12x8 (right).

In conclusion, we have implemented an ABR which is motion and margin aware and improved MTHQD up to 64% with a cost of viewport quality decrease as shown in Figure 27a. So that led us to a new direction. With the help of motion aware margins we have improved MTHQD but for lower bandwidth conditions that causes the ABR to decrease the whole viewport quality. Throughout the playback session, most of the time, instead of the highest viewport quality (1), the second highest quality (2) was selected by the ABR. In order to solve this problem, we have published new algorithms explained in Section 5.3.2.

### 5.3.2 Negative and Complementary Margins

Previously, it has been shown that head-motion-aware margins which will be referenced as positive margins (PM) in this section, improve the MTHQD with a

cost of bandwidth resource. This cost is coming from the extra downloaded margin tiles in higher quality than the background tiles in order to decrease MTHQD. However, this extra created network traffic can be a real problem in low bandwidth conditions as shown in Figure 27a where in this scenario the ABR sensed the drop in playback buffer and decreased the whole quality of the playback session. So In order to increase the MTHQD, the average viewport quality was downshifted in this case. In order to avoid this problem, we have investigated the problem in detail and proposed a new bandwidth neutral algorithm by enhancing the existing margins. In Figure 30 the new margin concepts are shown with an example scenario. In this figure, the viewport and background tiles are used with the same meaning as we discussed previously. The margins calculated by Algorithm 11 are now enhanced not to be used by measuring the bandwidth value continuously, and these margins are called Positive Margins (PM). Now the new enhanced PM will not select margin tiles when there is not enough bandwidth resource, instead, it will use Negative Margins (NM) and Complementary Margins (CM).

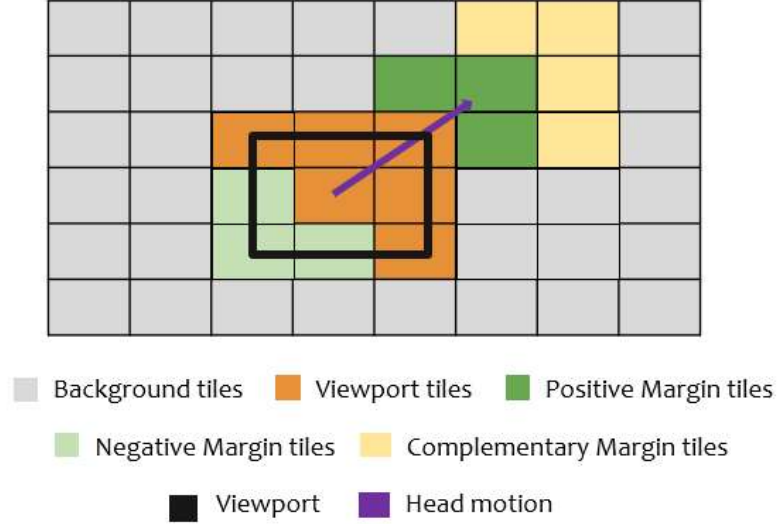


Figure 30: Proposed HMAVM margins.

The enhanced PM Algorithm is shown in Algorithm 12 where it monitors the bandwidth and selects a new margin tile only if there is enough bandwidth for the new margin tile. In Algorithm 12 if the playback buffer is less than 3

segment duration or the current bitrate selected by the ABR is less than the previous selected bitrate, in other words, if the ABR is downshifting then the algorithm will stop calculating further positive margins and return what it has already found. If those conditions are met then it will add a maximum of 30% positive margins as long as there is enough bandwidth. If the bandwidth is not enough to add another PM tile it will return the tiles calculated that satisfy the bandwidth and total percentage conditions. In the algorithm, we also calculate the remaining bandwidth by using  $B_{FS}^q$  values which were explained in detail in FRISBE paper [97].  $B_{FS}^q$  represents the minimum bitrate required to download viewport and background tiles in quality  $q$  which is calculated with an estimation algorithm explained in detail in the paper.

---

**Algorithm 12** Get Positive Margins

---

```

1: function GETPOSITIVEMARGINS(possibleMargins)
2:    $minBuffer \leftarrow 3 \times segmentDuration$ 
3:    $maxPercent \leftarrow 30\%$ 
4:    $totalPercent \leftarrow 0$ 
5:    $positiveMargins \leftarrow \emptyset$ 
6:    $q \leftarrow currentBitrateIndex$ 
7:    $q' \leftarrow previousBitrateIndex$ 
8:    $r = getDashThroughputInBps() - B_{FS}^q$ 
9:   if ( $currentBuffer < minBuffer$ ) or ( $q < q'$ ) then
10:     return  $positiveMargins$ 
11:   end if
12:   for each  $margin$  in  $possibleMargins$  do
13:     if ( $totalPercent < maxPercent$ ) and ( $r > 0$ ) then
14:        $positiveMargins \leftarrow positiveMargins \cup margin$ 
15:        $p \leftarrow area(margin)/area(viewport)$ 
16:        $totalPercent \leftarrow totalPercent + p$ 
17:        $r \leftarrow r - bitrate(margin)$ 
18:     else
19:       break;
20:     end if
21:   end for
22:   return  $positiveMargins$ 
23: end function

```

---

After calculating PM tiles that can fit in the remaining bandwidth, further improvements will be done with our new bandwidth neutral algorithm to find Negative and Complementary margins stated in Algorithm 13. The idea behind

the negative margin is to mark the viewport tile which is in the opposite direction of the head motion and start downloading this possibly unnoticeable tile in background quality. Instead of this unnoticeable tile, a margin tile is downloaded which we call as Complementary Margin (CM). In the algorithm, we calculate a *speedFactor* which is used to download more margin tiles when the head speed is higher. A maximum negative margin percentage of 30% is assigned linearly depending on the head speed. Also when the user is moving its head real fast, these viewport tiles marked in the viewport which are in the opposite direction will not be noticeable. The algorithm terminates when the totalArea added is less than the calculated  $p$  percentage value. For each possible complementary margin, the algorithm finds the farthest tile to that complementary margin in the opposite direction of the head motion and adds that farthest tile as a negative margin. Every negative margin must have a complementary margin which is the maximum distance by using the haversine circular distance.

For a complementary margin which is denoted as *margin* in Algorithm 13, the farthest tile is marked as negative margin in Algorithm 14. This algorithm iterates on all viewport tiles which are not already part of the Negative Margins (NM), then it calculates circular distance  $d$  and angularDistance  $a$  between the head motion vector and the vector starting from the viewport center to the tile center. The algorithm finds the tile with maximum circular and angular distances. Negative margins should also be in the opposite direction with the head motion vector,  $a > \frac{\pi}{2}$  condition is to find the tile in the opposite direction with the head motion vector.

In order to evaluate our proposed solution, we have used the OMAF Player in Windows 10 environment. OMAF player is publicly available on <sup>1</sup>. We have evaluated our system using HTTP/1.1, detailed viewport quality results are shown in Figures 31 and 32. In this graph, the details for 40 Mbps and 60 Mbps with different head motion speed 25 dps, 45 dps, 60 dps, 90 dps, 120 dps and the

---

<sup>1</sup><https://github.com/nokiatech/omaf>

---

**Algorithm 13** Get Negative Margins

---

```
1: function GETNM(unusedPMTiles, VPTiles)
2:    $maxNM \leftarrow 30\%$ 
3:    $maxHeadSpeed \leftarrow 120$ 
4:    $speed \leftarrow getHeadMotionSpeedInDps()$ 
5:    $speedFactor \leftarrow 1$ 
6:    $NM \leftarrow \emptyset$ 
7:    $complementary \leftarrow \emptyset$ 
8:    $totalPercent \leftarrow 0$ 
9:   if ( $speed < maxHeadSpeed$ ) then
10:     $speedFactor \leftarrow \frac{speed}{maxHeadSpeed}$ 
11:   end if
12:    $p \leftarrow speedFactor \times maxNM$ 
13:   for each unusedPMTile in unusedPMTiles do
14:     if ( $totalPercent \geq p$ ) then
15:       break
16:     end if
17:      $margin \leftarrow unusedPMTile$ 
18:      $f \leftarrow findFarthestVPTile(margin, VPTiles, NM)$ 
19:      $area \leftarrow getVPIntersection(f)$ 
20:     if  $totalPercent + area < p$  then
21:        $totalPercent \leftarrow totalPercent + area$ 
22:        $NM \leftarrow NM \cup f$ 
23:        $complementary \leftarrow complementary \cup margin$ 
24:     end if
25:   end for
26:   return  $\{NM, complementary\}$ 
27: end function
```

---

average of subjective human head motion (Harbor) is shown in detail for each scenario. We used BOLA algorithm for streaming decisions, *BOLA\_NO* refers to no margin case, *BOLA\_30* refers to BOLA with a PM of maximum 30% and *BOLA\_30\_NEGATIVE* refers to the whole HMAVM solution which includes all PM, NM and CM. Looking in Figure 31 it can be seen that viewport quality is best in HMAVM cases for both 40 Mbps and 60 Mbps. As the speed increases our Algorithm 13 will pick more NM and CM tiles and that behavior can be seen in 40 Mbps where we have a bandwidth bottleneck and the improvement increases as the head motion speed increases. For Trolley in Figure 32, the improvement is more visible since that video sequence has a static content for most of the playback session and other than that the results look very similar to the Harbor sequence.

---

**Algorithm 14** Find Farthest Viewport Tile

---

```
1: function FINDFARTHESTVPTILE(marginTile, VPTiles, NM)
2:   result  $\leftarrow \emptyset$ 
3:   maxDistance  $\leftarrow 0$ 
4:   maxAngularDistance  $\leftarrow 0$ 
5:   for each VPTile in VPTiles do
6:     if VPTile  $\in$  NM then
7:       continue
8:     end if
9:     d  $\leftarrow$  getCircularDistanceFromVP(marginTile)
10:    a  $\leftarrow$  getAngularDistanceFromMotion(VPTile)
11:    if d  $>$  maxDistance and a  $>$  maxAngularDistance and a  $>$   $\frac{\pi}{2}$  then
12:      maxDistance  $\leftarrow d$ 
13:      maxAngularDistance  $\leftarrow a$ 
14:      result  $\leftarrow$  VPTile
15:    end if
16:  end for
17:  return result
18: end function
```

---

Even though it seems that NM and CM won't affect the average viewport quality assuming we render one of the tiles in the viewport in low quality and a CM tile is downloaded instead of this unnoticeable viewport tile, the way they work makes a non-negligible improvement as observed in the results. The reason for that improvement is the working harmony of CM and NM together. When ABR decides to mark a viewport tile as NM tile and instead of that tile pick a CM tile, then the ABR will use already downloaded high quality content which was already downloaded for that viewport tile and it starts low quality download for the segments it didn't download till that moment, opposed to the CM tile where for CM tiles low quality downloaded data will be thrown away and high quality data will be re-downloaded. And together NM and CM utilize the resources in a very good way so that future background tile (NM) and future viewport tile (CM) is downloaded with correct qualities before the head motion finishes. And based on this prediction's accuracy, viewport quality also improves as it can be seen in Figures 31 and 32. Another important result looking in the viewport quality results from the graphs is that HMAVM results have the best viewport quality not only in artificial speed tests, but also for subjective human tests that are visible

in all configurations. The improvement in viewport quality is best visible in the 12x8 tile configuration which was expected since there are more tiles to utilize the bandwidth resource with smaller tile sizes, especially for 40 Mbps scenario.

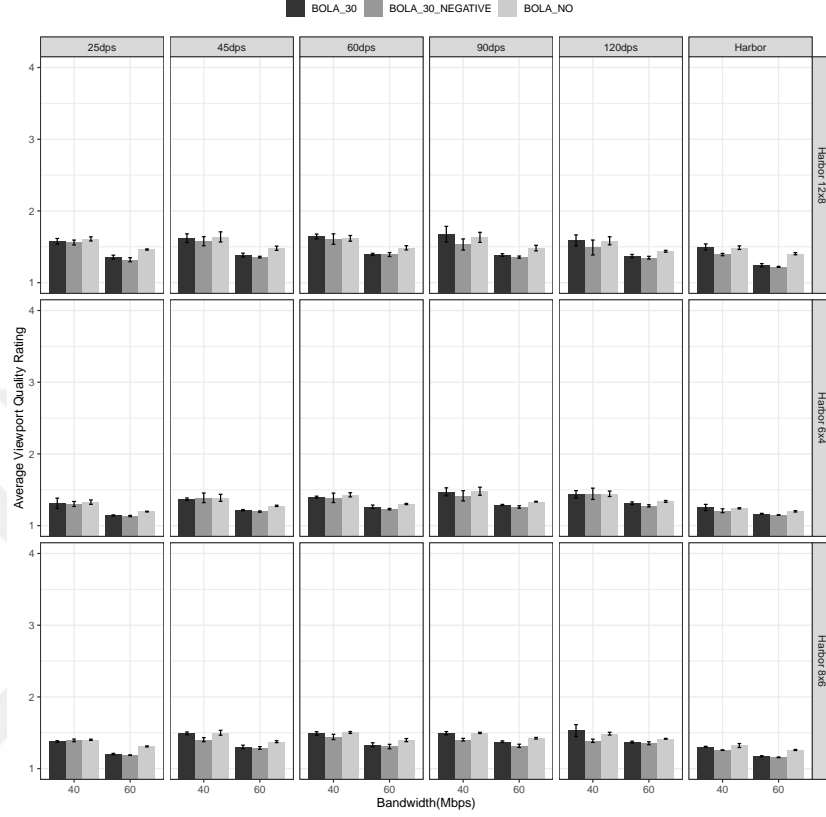


Figure 31: Average viewport quality with HTTP/1.1 for Harbor.

Similarly, in Figures 33 and 34 MTHQD for Harbor and Trolley test sequences for varying head motion speeds and bandwidth conditions are shown. Looking into those results we can say that MTHQD is not improved much on HMAVM for the 40 Mbps scenario and a very slight improvement for the 60 Mbps case. Even though the viewport quality is increased using HTTP/1.1, the MTHQD is not improved a lot since there is no request cancellation in HTTP/1.1. As an example, in HTTP/1.1 for 8x6 tile configuration there are 48 connections initiated for a segment to be able to be decoded and played back. This creates race conditions in the communication layer which affects the MTHQD directly. If you look at the 12x8 tile configuration and 90 dps head motion speed, in Figure 34 the error bar is showing that problem. Since the number of tiles is bigger for 12x8 tile



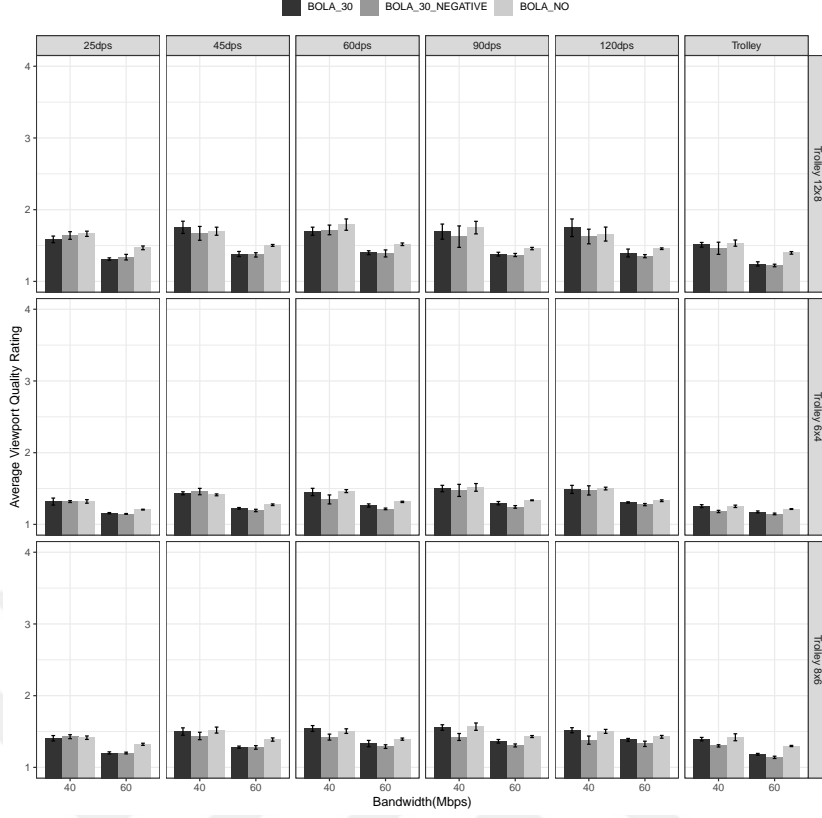


Figure 32: Average viewport quality with HTTP/1.1 for Trolley.

configuration, this HTTP/1.1 communication problem effect also increases as tile configuration increases.

In order to solve this problem, we have changed the architecture of our system to use H2 and multiplexing which solves the HTTP request cancellation problem which is shown in Figure 23. We have created a new thread which is responsible for managing the new added, removed and completed curl handles in the multiplexing connection. We have used two multiplexing connections with libuv 1.42.0 and libcurl 7.80.0 [98]. With H2 now we are able to schedule, cancel or abort an existing request. Also with H2 multiplexing, it has been possible to reuse the same HTTP connection to request different tiles, whereas in HTTP/1.1 that was a limitation and was causing HTTP connection thread race condition issues at the operating system level even with unlimited bandwidth conditions. H2 solves that concurrent communication problem, in the architecture it can be seen that for each tile representation, the client will request from the HTTP connection

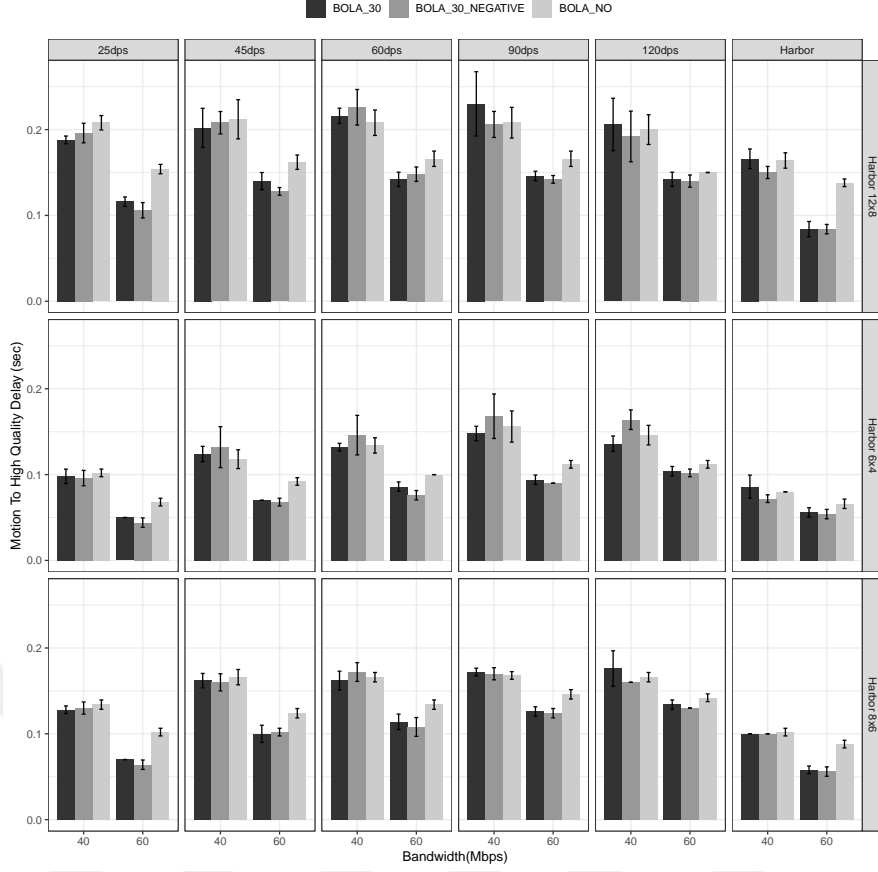


Figure 33: Average MTHQD with HTTP/1.1 for Harbor.

manager which is a separate thread. Then, the connection manager thread will schedule the requested downloads from multiplexed connections and deliver the binary data back to the requester when the download finishes. This enhanced H2 architecture results in MTHQD improvements that can be seen in Figure 35 especially for the 12x8 tile configuration since it has 96 concurrent downloads for each tile.

With H2, we have made ten runs for each tile configuration, bandwidth condition and head motion data for Trolley and Harbor sequences shown in Figure 36. With H2, the HTTP communication problem is solved and the results further improved with our new communication architecture shown in Figure 23. For 40 Mbps as it can be seen, the improvement is not too much as expected since HMAVM cannot use PM tiles with this limited bandwidth. For the 40 Mbps case, the improvement is only coming from NM and CM tiles which is a bandwidth neutral

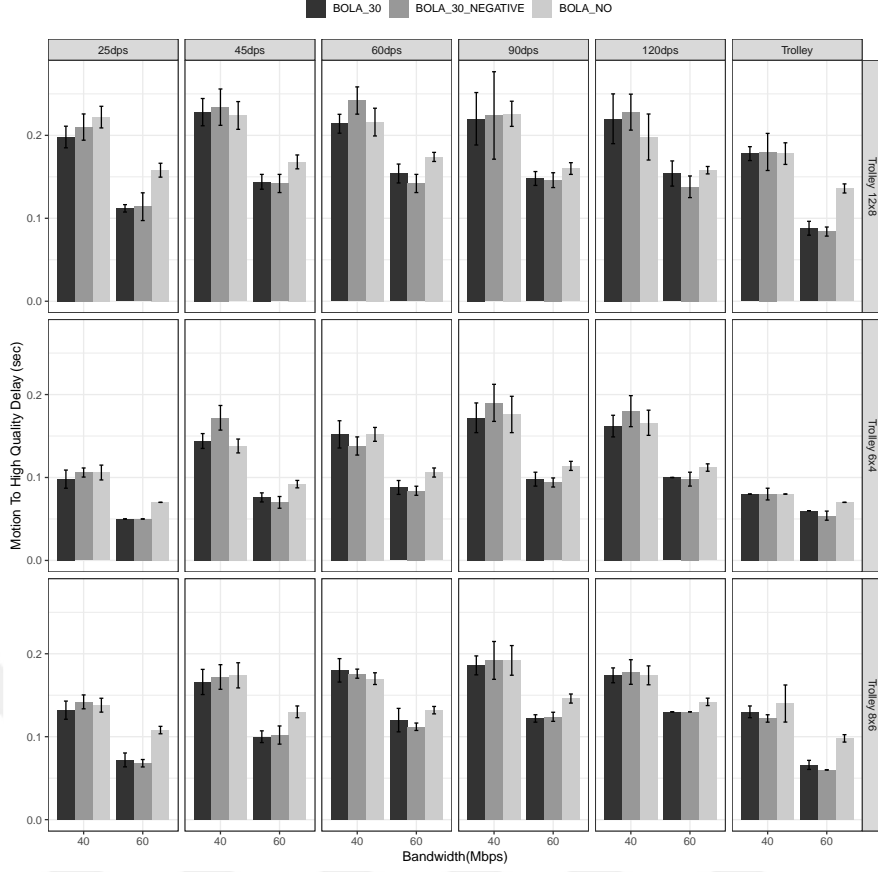


Figure 34: Average MTHQD with HTTP/1.1 for Trolley.

solution. And considering our Algorithm 13 is sensitive to head motion speed, the improvement in 40 Mbps is maximum in fast head motions and higher tile configurations which can be seen in Figure 36e. In this figure, the MTHQD delay with HMAVM improvement is visible in 60 dps, 90 dps and 120 dps. As a result, we have observed an average viewport quality increase up to 20% and an average MTHQD reduction up to 50%. In Figure 36 the results on the right-hand side are for 60 Mbps bandwidth condition which is enough to download viewport and margin tiles. As opposed to the 40 Mbps bandwidth condition, with 60 Mbps bandwidth resource our HMAVM solution will use PM, CM and NM all together since the bandwidth is enough to use PM too. And the additional improvement that is gained by the addition of PM tiles is more evident in the results. For 60 Mbps HMAVM produces an improvement for all of the tile configuration and

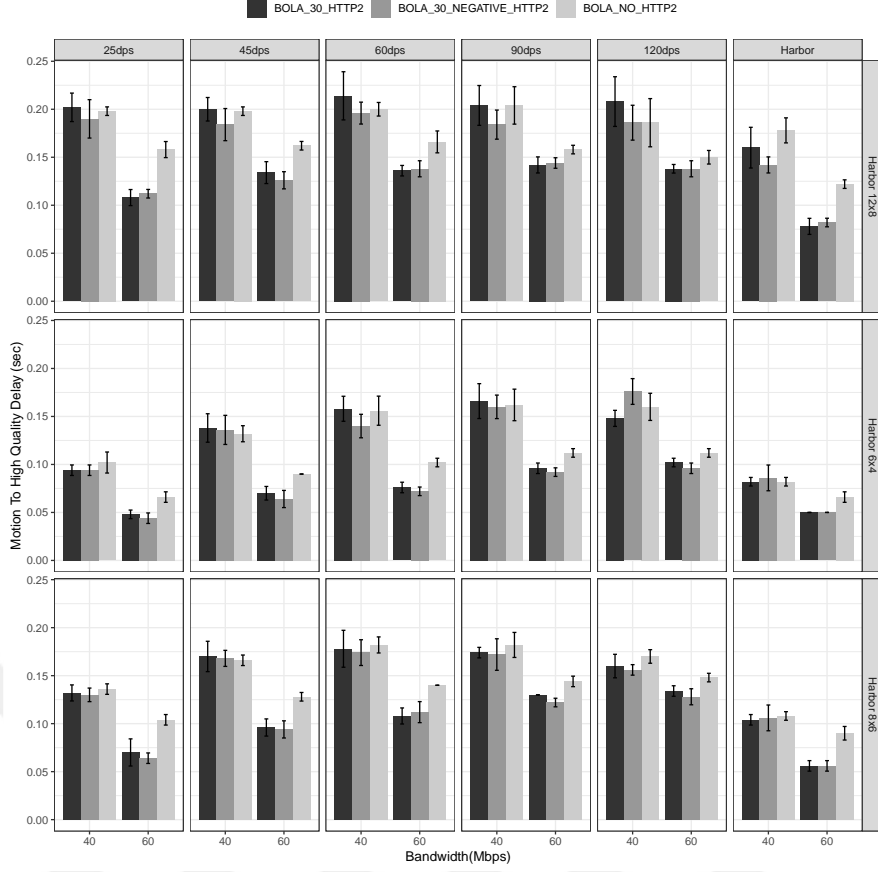
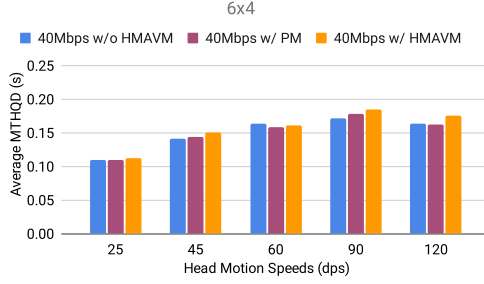


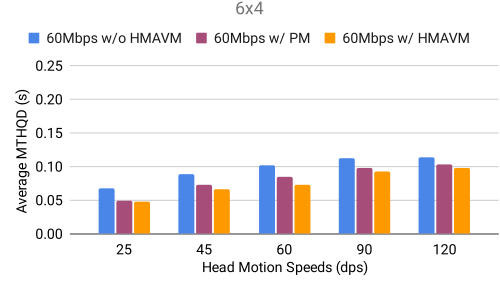
Figure 35: Average MTHQD with H2 for Harbor.

head motion speeds. The maximum MTHQD improvement is 50 ms(26% improvement compared to with-out margin scenario) for 8x6 tile configuration and 60 dps.

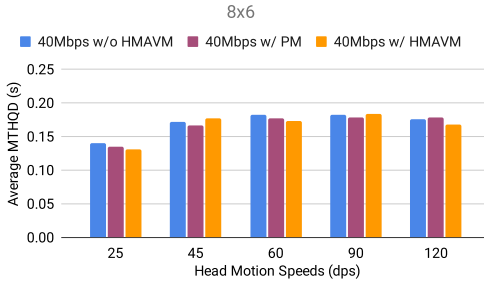
After implementing H2 we have re-evaluated our proposed solution with additional datasets which are **RollerCoaster** and **Timelapse** sequences from the dataset [99] which can be seen in Figure 37. With H2, our proposed HMAVM solution works much better and we could make a significant improvement both in viewport quality and MTHQD. With this dataset, we have increased our subjective testing since it contained 63 user head motion data. With these tests, we have observed a viewport quality increase up to 20% and a reduction in MTHQD up to 50%.



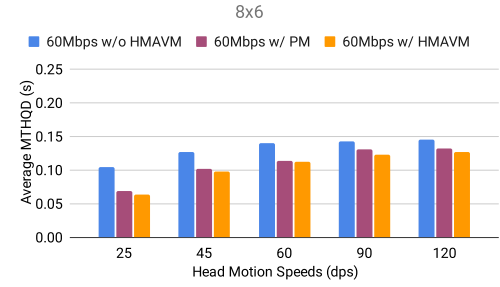
(a) 40 Mbps, 6x4



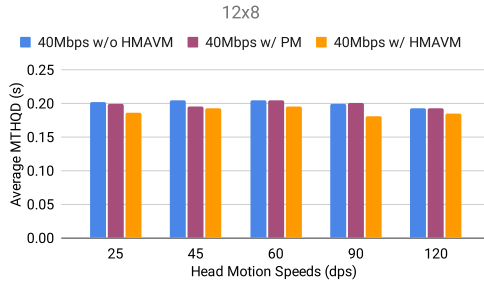
(b) 60 Mbps, 6x4



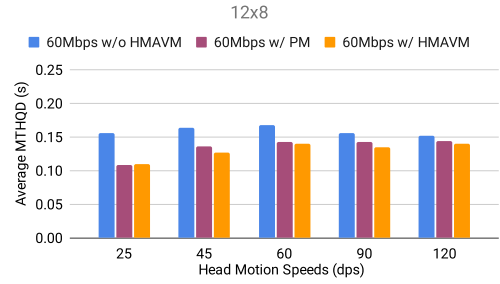
(c) 40 Mbps, 8x6



(d) 60 Mbps, 8x6



(e) 40 Mbps, 12x8



(f) 60 Mbps, 12x8

Figure 36: Average MTHQD for 40 Mbps (top) and 60 Mbps (bottom) on artificial head motions with tile configurations of 6x4 (left), 8x6 (middle) and 12x8 (right) for the *Trolley* and *Harbor* sequences with H2.

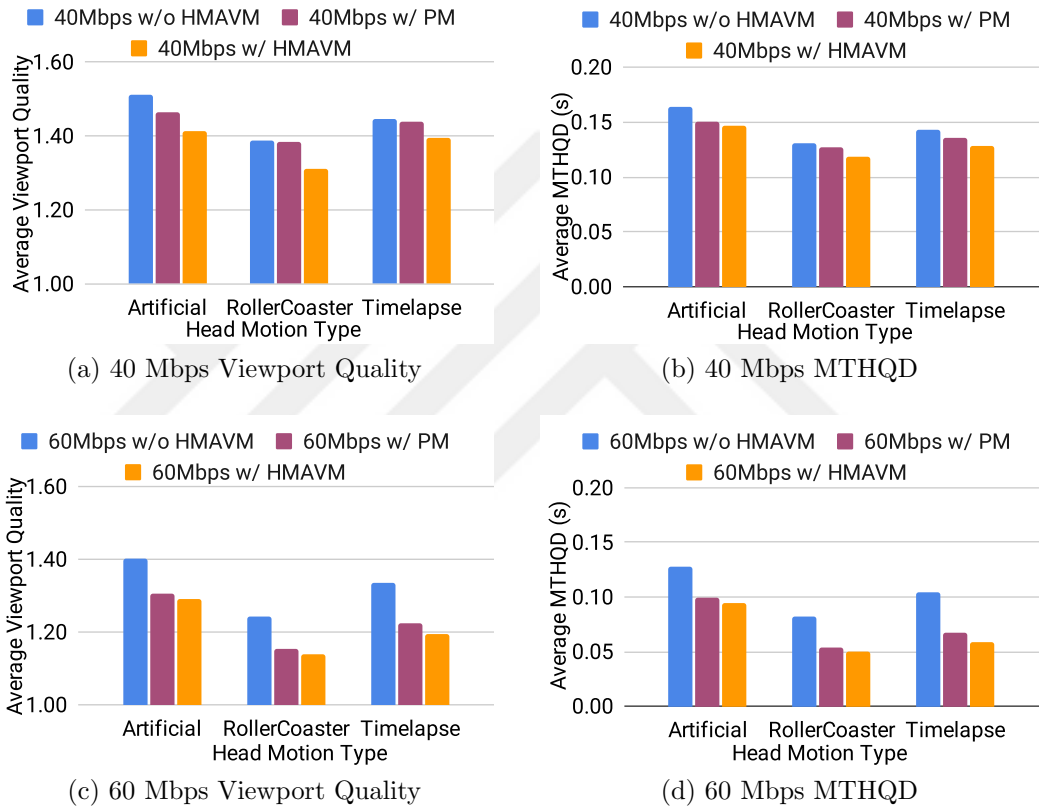


Figure 37: Average viewport quality (left) and MTHQD (right) results for 40 Mbps(top) and 60 Mbps(bottom) with H2.

## CHAPTER VI

### CONCLUSION AND FUTURE WORK

Server and client-side algorithms play a very important role in achieving a better user experience in video streaming. Using objective evaluation metrics discussed in Sections 1.3.1 and 1.3.2, we analyzed four main aspects of video streaming (i) bandwidth prediction accuracy, (ii) utilization of playback speed, (iii) adaptive streaming for content-aware-encoded videos and (iv) head-motion-awareness for 360-degree videos.

In our Size-Aware Rate Adaptation (SARA) algorithm, we have designed an ABR algorithm which adapts to encoding bitrate variation with the transfer of future segment size information and using stable bitrate predictions described in detail. We compared SARA algorithm with dash.js rate-based and dynamic algorithms and observed reductions in rebuffering durations up to 91% compared to the rate adaptive algorithm and up to 87% for the dynamic algorithm. In the SARA algorithm, we simply shared size information from the server side to the client side. In this algorithm we have used a simple approach by using just the size information, a future direction would be also to use the quality value assigned for each segment in addition to the sizes. That quality value might be PSNR, VMAF, or any other metric that can be used to identify the segment quality.

With recent improvements, information shared between server and client systems are standardized with CMCD and CDSO. We showed how these standards can be used to increase the user experience in video streaming. With CMCD we could reduce the average rebuffering duration up to 48% and similarly we could achieve a 33% improvement in rebuffering duration using CDSO.

After making improvements for on-demand streaming, we have also analyzed

the low-latency adaptation problem. Traditional ABR bandwidth prediction methods fail to calculate or predict the bandwidth accurately for live-latency, especially when the content generation is the bottleneck and the streaming session is source limited. Existing approaches are simply measuring the encoding bitrate and most of the well known ABRs are failing to adapt to the changing conditions. So we have designed a new algorithm Low-on-Latency (LoL) and improved it later to be used by industry standard dash.js player. Our improved algorithm Low-on-Latency<sup>+</sup> (LoL<sup>+</sup>) is now part of the dash.js player. We have compared our algorithm with several existing methods, including the winner algorithm of the Twitch Grand Challenge [50] which is named L2A. We observed LoL<sup>+</sup> achieving an average reduction of 61.9% for rebuffering durations and 8.1% improvement in latency compared to the L2A algorithm. After evaluating LoL<sup>+</sup> with different contents we observed that it can still suffer from rebufferings caused by the miscalculated playback speed during the playback sessions. In order to further analyze this issue, we made a proof of concept study by developing CAPSC algorithm, which pays attention to the event density of the content. With CAPSC, we have assigned the playback speed using the event density for a sample scenario and showed the impact of content event density on the playback session could be helpful. We showed that a generic event density aware playback controller with wider subjective testing would make a difference in HTTP adaptive streaming.

As the popularity of interactive streaming increases, we have investigated the bandwidth prediction problem in WebRTC systems. Since the content is being encoded live, the bandwidth prediction was measuring the encoding bitrate in WebRTC systems which is basically the same problem we have solved with LoL<sup>+</sup>. After investigating the same problem for WebRTC we have designed a hybrid algorithm that we call BoB. BoB has a heuristic and learning based controller to solve the bandwidth prediction problem in WebRTC clients to adapt better which allows the client to pick higher encoding rates. We have compared our algorithm with the winner and the runner up algorithm of the bandwidth prediction challenge [78]



sponsored by Microsoft. We have compared BoB with Gemini, HRCC, and pure Heuristic algorithms. The bandwidth accuracy for each is 81.03%, 70.37%, 63.36%, and 32.84% respectively.

Lastly, we have analyzed the 360-degree video viewport dependent streaming (VDS) which is not too much different than conventional video streaming. For 360-degree video, we have the head motion data coming from Head Mounted Display (HMD) devices that help us to point the viewport where the user is looking. Using head motion data, we have created head-motion-aware margins and later we have extended these margins with HMAVM which includes Positive Margins, Negative Margins, and Complementary Margins. We have evaluated our new proposed algorithm with multiple datasets and as a result, in average we have observed a viewport quality improvement of up to 20% and a reduction in MTHQD up to 50%. As a future direction, it may also be possible to use Deep Reinforcement Learning techniques to utilize viewport quality decisions and margin selections in the ABR. Another possible improvement to improve MTHQD in 360-degree streaming with multiple content qualities is the delay introduced by the quality switching delays after the ABR makes the quality switch decision. In our tests, we have used small segment durations to workaround this problem, but it might be possible to switch to a new representation without waiting for the segment boundary which can improve the MTHQD delay.

## Bibliography

- [1] M. Lim, M. N. Akcay, A. Bentaleb, A. C. Begen, and R. Zimmermann, “When They Go High, We Go Low: Low-Latency Live Streaming in dash.js with LoL,” in *ACM MMSys*, 2020 (DOI: 10.1145/3339825.3397043).
- [2] A. Bentaleb, M. Lim, M. N. Akcay, A. C. Begen, and R. Zimmermann, “Common media client data (CMCD): Initial findings,” in *ACM NOSSDAV*, 2021 (DOI: 10.1145/3458306.3461444).
- [3] M. Lim, M. N. Akcay, A. Bentaleb, A. C. Begen, and R. Zimmermann, “The benefits of server hinting when DASHing or HLSing,” in *ACM MHV*, 2022 (DOI: 10.1145/3510450.3517317).
- [4] A. Bentaleb, M. N. Akcay, M. Lim, A. C. Begen, and R. Zimmermann, “Catching the Moment With LoL<sup>+</sup> in Twitch-Like Low-Latency Live Streaming Platforms,” *IEEE Transactions on Multimedia*, vol. 24, pp. 2300–2314, 2022.
- [5] M. M. Hannuksela, Y.-K. Wang, and A. Hourunranta, “An overview of the OMAF standard for 360° video,” in *DCC*, 2019.
- [6] A. C. Begen, M. N. Akcay, A. Bentaleb, and A. Giladi, “Adaptive streaming of content-aware-encoded videos in dash.js,” *SMPTE Motion Imaging Jour.*, vol. 131, no. 4, pp. 30–38, 2022 (DOI: 10.5594/JMI.2022.3160560).
- [7] N. Weil and R. Bouqueau, “Ultra-Low-Latency with CMAF.” Akamai White paper. Online; accessed 10 July 2022.
- [8] O. F. Aladag, D. Ugur, M. N. Akcay, and A. C. Begen, “Content-aware playback speed control for low-latency live streaming of sports,” in *ACM MMSys*, 2021 (DOI: 10.1145/3458305.3478437).
- [9] Cisco, “VNI Complete Forecast Highlights,” *Technical Report*, Cisco, March 2021.
- [10] DASH-IF, “DASH Reference Client.” [Online] Available: <https://reference.dashif.org/dash.js/>. Accessed on June 1, 2021.
- [11] “Coded Representation of Immersive Media—Part 2: Omnidirectional Media Format, ISO/IEC 23090-2:2019,” standard, Information Technology, 2019.
- [12] R. Skupin, Y. Sanchez, C. Hellge, and T. Schierl, “Tile based HEVC video for head mounted displays,” in *IEEE ISM*, 2016.
- [13] W. Zhou, X. Min, H. Li, and Q. Jiang, “A brief survey on adaptive video streaming quality assessment,” *Journal of Visual Communication and Image Representation*, vol. 86, p. 103526, 2022.
- [14] G. Yi, D. Yang, A. Bentaleb, W. Li, Y. Li, K. Zheng, J. Liu, W. T. Ooi, and Y. Cui, “The ACM Multimedia 2019 Live Video Streaming Grand Challenge,” in *ACM Multimedia*, 2019.

- [15] I. D. Curcio, H. Toukoma, and D. Naik, “360-degree video streaming and its subjective quality,” in *SMPTE Annual Tech. Conf. and Exh.*, 2017.
- [16] Z. Li, A. C. Begen, J. Gahm, Y. Shan, B. Osler, and D. Oran, “Streaming video over http with consistent quality,” in *Proceedings of the 5th ACM Multimedia Systems Conference*, MMSys ’14, (New York, NY, USA), p. 248–258, Association for Computing Machinery, 2014.
- [17] A. C. Begen, “Quality-aware HTTP adaptive streaming,” in *IBC*, 2015.
- [18] Y. Qin, S. Hao, K. R. Pattipati, F. Qian, S. Sen, B. Wang, and C. Yue, “Quality-aware strategies for optimizing abr video streaming qoe and reducing data usage,” in *ACM MMSys*, 2019.
- [19] W. Cooper, S. Farrell, and K. Subramanian, “Qbr metadata to improve streaming efficiency and quality,” in *SMPTE 2017 Annual Technical Conference and Exhibition*, pp. 1–9, 2017.
- [20] P. Juluri, V. Tamarapalli, and D. Medhi, “Sara: Segment aware rate adaptation algorithm for dynamic adaptive streaming over http,” in *IEEE International Conference on Communication Workshop (ICCW)*, pp. 1765–1770, 2015.
- [21] Y. Engel, S. Mannor, and R. Meir, “The kernel recursive least-squares algorithm,” *IEEE Transactions on Signal Processing*, vol. 52, no. 8, pp. 2275–2285, 2004.
- [22] P. S. Agachi, M. V. Cristea, A. A. Csavdari, and B. Szilagyi, *2. Model predictive control*, pp. 32–74. De Gruyter, 2016.
- [23] “iproute2.” [Online] Available: <https://wiki.linuxfoundation.org/networking/iproute2>. Accessed on Jan. 21, 2022.
- [24] “CTA-5004: Web Application Video Ecosystem–Common Media Client Data.” [Online] Available: <https://cdn.cta.tech/cta/media/media/resources/standards/pdfs/cta-5004-final.pdf>. Accessed on Feb. 20, 2021.
- [25] NUS-OzU, “CMCD-aware System.” [Online] Available: <https://github.com/NUSstreaming/CMCD-DASH>. Accessed on Feb. 20, 2021.
- [26] “ISO/IEC 23000-19:2020 Information technology – Multimedia application format (MPEG-A) – Part 19: Common media application format (CMAF) for segmented media.” [Online] Available: <https://www.iso.org/standard/79106.html>. Accessed on June 1, 2021.
- [27] H. Nielsen, J. Mogul, L. M. Masinter, R. T. Fielding, J. Gettys, P. J. Leach, and T. Berners-Lee, “Hypertext Transfer Protocol – HTTP/1.1.” RFC 2616, June 1999.

- [28] C. Liu, I. Bouazizi, and M. Gabbouj, “Rate adaptation for adaptive http streaming,” in *Proceedings of the Second Annual ACM Conference on Multimedia Systems*, MMSys ’11, (New York, NY, USA), p. 169–174, Association for Computing Machinery, 2011.
- [29] C. Liu, I. Bouazizi, M. M. Hannuksela, and M. Gabbouj, “Rate adaptation for dynamic adaptive streaming over HTTP in content distribution network,” *Signal Processing: Image Communication*, vol. 27, no. 4, pp. 288–311, 2012. Modern Media Transport – Dynamic Adaptive Streaming over HTTP (DASH).
- [30] Z. Li, X. Zhu, J. Gahm, R. Pan, H. Hu, A. C. Begen, and D. Oran, “Probe and adapt: Rate adaptation for http video streaming at scale,” *IEEE JSAC*, vol. 32, no. 4, pp. 719–733, 2014.
- [31] X. Xie, X. Zhang, S. Kumar, and L. E. Li, “pistream: Physical layer informed adaptive video streaming over lte,” in *Proceedings of the 21st Annual International Conference on Mobile Computing and Networking*, 2015.
- [32] T. Andelin, V. Chetty, D. Harbaugh, S. Warnick, and D. Zappala, “Quality selection for dynamic adaptive streaming over http with scalable video coding,” in *Proceedings of the 3rd Multimedia Systems Conference*, MMSys ’12, (New York, NY, USA), p. 149–154, Association for Computing Machinery, 2012.
- [33] J. Jiang, V. Sekar, and H. Zhang, “Improving fairness, efficiency, and stability in http-based adaptive video streaming with festive,” in *Proceedings of the 8th international conference on Emerging networking experiments and technologies*, pp. 97–108, 2012.
- [34] C. Wang, A. Rizk, and M. Zink, “SQUAD: A Spectrum-based Quality Adaptation for Dynamic Adaptive Streaming over HTTP,” in *ACM MMSys*, 2016.
- [35] T.-Y. Huang, R. Johari, N. McKeown, M. Trunnell, and M. Watson, “A buffer-based approach to rate adaptation: Evidence from a large video streaming service,” in *ACM SIGCOMM*, 2014.
- [36] K. Spiteri, R. Urgaonkar, and R. K. Sitaraman, “Bola: Near-optimal bi-trate adaptation for online videos,” *IEEE/ACM Transactions on Networking*, vol. 28, no. 4, pp. 1698–1711, 2020.
- [37] P. K. Yadav, A. Shafiei, and W. T. Ooi, “Quetra: A queuing theory approach to dash rate adaptation,” in *ACM Multimedia*, 2017.
- [38] J. Jiang, V. Sekar, H. Milner, D. Shepherd, I. Stoica, and H. Zhang, “CFA: A Practical Prediction System for Video QoE Optimization,” in *USENIX NSDI*, 2016.
- [39] A. Bentaleb, C. Timmerer, A. C. Begen, and R. Zimmermann, “Bandwidth Prediction in Low-Latency Chunked Streaming,” in *ACM NOSSDAV*, 2019.

- [40] A. Bentaleb, C. Timmerer, A. C. Begen, and R. Zimmermann, “Performance analysis of ACTE: a bandwidth prediction method for low-latency chunked streaming,” *ACM TOMM*, vol. 16, no. 2s, pp. 1–24, 2020.
- [41] C. Gutterman, B. Fridman, T. Gilliland, Y. Hu, and G. Zussman, “STALLION: Video Adaptation Algorithm for Low-Latency Video Streaming,” in *ACM MMSys*, 2020.
- [42] H. Peng, Y. Zhang, Y. Yang, and J. Yan, “A Hybrid Control Scheme For Adaptive Live Streaming,” in *ACM Multimedia*, 2019.
- [43] P. Houze, E. Mory, G. Texier, and G. Simon, “Applicative-Layer Multipath For Low-Latency Adaptive Live Streaming,” in *IEEE ICC*, 2016.
- [44] T. Kohonen, *Self-Organizing Maps*, vol. 30. Springer Science & Business Media, 2012.
- [45] H. Jin, K. Leung, and M. L. Wong, “An Integrated Self-organizing Map for the Traveling Salesman Problem,” *Advances in Neural Networks and Appl.*, 2001.
- [46] X. Yin, A. Jindal, V. Sekar, and B. Sinopoli, “A Control Theoretic Approach For Dynamic Adaptive Video Streaming Over HTTP,” in *ACM SIGCOMM*, 2015.
- [47] Y. Dogan, D. Birant, and A. Kut, “SOM++: Integration of Self Organizing Map and K-means++ Algorithms,” in *Springer MLDM*, 2013.
- [48] S. Park, S. Seo, C. Jeong, and J. Kim, “The Weights Initialization Methodology of Unsupervised Neural Networks to Improve clustering Stability,” *The Journal of Supercomputing*, pp. 1–17, 2019.
- [49] R. Burkard, M. Dell’Amico, and S. Martello, *Assignment problems: revised reprint*. SIAM, 2012.
- [50] Twitch, “Grand Challenge on Adaptation Algorithms for Near-Second Latency,” in *ACM MMSys*, 2020.
- [51] “Low-on-Latency-plus (LoL<sup>+</sup>).” [Online] <https://github.com/NUStreaming/LoL-plus>. Online; accessed on Sept. 5, 2020.
- [52] J. Van Der Hooft, S. Petrangeli, T. Wauters, R. Huysegems, P. R. Alface, T. Bostoen, and F. De Turck, “HTTP/2-based Adaptive Streaming of HEVC Video Over 4G/LTE Networks,” *IEEE Communications Letters*, vol. 20, no. 11, pp. 2177–2180, 2016.
- [53] K.-Y. Cheng, S.-J. Luo, B.-Y. Chen, and H.-H. Chu, “SmartPlayer: user-centric video fast-forwarding,” in *Conf. Human Factors in Computing Systems*, 2009.
- [54] C.-T. Kao, Y.-T. Liu, and A. Hsu, “Speeda: Adaptive speed-up for lecture videos,” in *ACM Symp. User Interface Software and Technology*, 2014.

- [55] I. Abibouraguimane, K. Hagihara, K. Higuchi, Y. Itoh, Y. Sato, T. Hayashida, and M. Sugimoto, “CoSummary: adaptive fast-forwarding for surgical videos by detecting collaborative scenes using hand regions and gaze positions,” in *Int. Conf. Intelligent User Interfaces*, 2019.
- [56] L. Sun, T. Zong, S. Wang, Y. Liu, and Y. Wang, “Tightrope walking in low-latency live streaming: optimal joint adaptation of video rate and playback speed,” in *ACM MMSys*, 2021.
- [57] V. Jacobson, “Congestion avoidance and control,” *ACM SIGCOMM CCR*, vol. 18, no. 4, pp. 314–329, 1988.
- [58] S. Floyd, T. Henderson, and A. Gurtov, “The newreno modification to TCP’s fast recovery algorithm.” [Online] Available: <https://datatracker.ietf.org/doc/html/rfc3782>, 2004. Accessed on Jan. 21, 2022.
- [59] S. Ha, I. Rhee, and L. Xu, “CUBIC: a new TCP-friendly high-speed TCP variant,” *ACM SIGOPS operating systems review*, vol. 42, no. 5, pp. 64–74, 2008.
- [60] L. S. Brakmo, S. W. O’Malley, and L. L. Peterson, “TCP vegas: New techniques for congestion detection and avoidance,” in *Proceedings of the Conference on Communications Architectures, Protocols and Applications*, pp. 24–35, 1994.
- [61] N. Cardwell, Y. Cheng, C. S. Gunn, S. H. Yeganeh, and V. Jacobson, “Bbr: congestion-based congestion control,” *Communications of the ACM*, vol. 60, no. 2, pp. 58–66, 2017.
- [62] N. Cardwell, Y. Cheng, S. H. Yeganeh, I. Swett, V. Vasiliev, P. Jha, Y. Seung, M. Mathis, and V. Jacobson, “BBR v2 a Model-based Congestion Control.” [Online] Available: <https://datatracker.ietf.org/meeting/104/materials/slides-104-iccr-g-an-update-on-bbr-00>, 2019. Accessed on Jan. 21, 2022.
- [63] K. Winstein and H. Balakrishnan, “TCP ex machina: Computer-generated congestion control,” *ACM SIGCOMM CCR*, vol. 43, no. 4, pp. 123–134, 2013.
- [64] M. Dong, T. Meng, D. Zarchy, E. Arslan, Y. Gilad, B. Godfrey, and M. Schapira, “PCC vivace: Online-learning congestion control,” in *USENIX NSDI*, 2018.
- [65] F. Y. Yan, J. Ma, G. D. Hill, D. Raghavan, R. S. Wahby, P. Levis, and K. Winstein, “Pantheon: the training ground for internet congestion-control research,” in *2018 USENIX Annual Technical Conference (USENIX ATC 18)*, 2018.
- [66] N. Jay, N. Rotman, B. Godfrey, M. Schapira, and A. Tamar, “A deep reinforcement learning perspective on internet congestion control,” in *Int. Conf. Machine Learning*, PMLR, 2019.
- [67] S. Abbasloo, C.-Y. Yen, and H. J. Chao, “Classic meets modern: A pragmatic learning-based congestion control for the internet,” in *ACM SIGCOMM*, 2020.

- [68] X. Zhu, P. Pan, M. Ramalho, and S. Mena, “Network-assisted dynamic adaptation (NADA): A unified congestion control scheme for real-time media.” [Online] Available: <https://datatracker.ietf.org/doc/html/rfc8698>, 2020. Accessed on Jan. 21, 2022.
- [69] I. Johansson and Z. Sarker, “Self-clocked rate adaptation for multimedia.” [Online] Available: <https://datatracker.ietf.org/doc/html/rfc8298>, 2020. Accessed on Jan. 21, 2022.
- [70] M. Polese, F. Chiariotti, E. Bonetto, F. Rigotto, A. Zanella, and M. Zorzi, “A survey on recent advances in transport layer protocols,” *IEEE Communications Surveys & Tutorials*, vol. 21, no. 4, pp. 3584–3608, 2019.
- [71] J. Fang, M. Ellis, B. Li, S. Liu, Y. Hosseinkashi, M. Revow, A. Sadovnikov, Z. Liu, P. Cheng, S. Ashok, *et al.*, “Reinforcement learning for bandwidth estimation and congestion control in real-time communications,” *arXiv preprint arXiv:1912.02222*, 2019.
- [72] Y. Tianrun, W. Hongyu, H. Runyu, Y. Shushu, L. Dingwei, and Z. Jiaqi, “Gemini: An ensemble framework for bandwidth estimation in web real-time communications,” in *ACM MMSys*, 2021.
- [73] “A Google Congestion Control Algorithm for Real-Time Communication.” [Online] Available: <https://datatracker.ietf.org/doc/html/draft-ietf-rmcat-gcc-02>. Accessed on Jan. 21, 2022.
- [74] “OpenNetLab AlphaRTC.” [Online] Available: <https://github.com/OpenNetLab/AlphaRTC>. Accessed on Jan. 21, 2022.
- [75] J. van der Hooft, S. Petrangeli, T. Wauters, R. Huysegems, P. R. Alface, T. Bostoen, and F. De Turck, “HTTP/2-Based Adaptive Streaming of HEVC Video Over 4G/LTE Networks,” *IEEE Communications Letters*, vol. 20, pp. 2177–2180, Nov. 2016.
- [76] H. Riiser, P. Vigmostad, C. Griwodz, and P. Halvorsen, “Commute Path Bandwidth Traces from 3G Networks: Analysis and Applications,” in *ACM MMSys*, 2013.
- [77] L. Mei, R. Hu, H. Cao, Y. Liu, Z. Han, F. Li, and J. Li, “Realtime Mobile Bandwidth Prediction Using LSTM Neural Network,” in *Int. Conf. Passive and Active Network Measurement*, Springer, 2019.
- [78] “Grand Challenge on Bandwidth Estimation for Real-Time Communications.” [Online] Available: [https://2021.acmmmsys.org/rtc\\_challenge.php](https://2021.acmmmsys.org/rtc_challenge.php). Accessed on Jan. 21, 2022.
- [79] A. Zare, A. Aminlou, M. M. Hannuksela, and M. Gabbouj, “HEVC-compliant tile-based streaming of panoramic video for virtual reality applications,” in *ACM Multimedia*, 2016.
- [80] X. Corbillon, G. Simon, A. Devlic, and J. Chakareski, “Viewport-adaptive navigable 360-degree video delivery,” in *IEEE ICC*, 2017.

- [81] M. Hosseini and V. Swaminathan, “Adaptive 360 VR video streaming: Divide and conquer,” in *IEEE ISM*, 2016.
- [82] I. D. Curcio, H. Toukoma, and D. Naik, “Bandwidth reduction of omnidirectional viewport-dependent video streaming via subjective quality assessment,” in *AltMM*, 2017.
- [83] D. Naik, I. D. D. Curcio, and H. Toukoma, “Optimized viewport dependent streaming of stereoscopic omnidirectional video,” in *Packet Video Workshop*, 2018.
- [84] J. Son, D. Jang, and E.-S. Ryu, “Implementing motion-constrained tile and viewport extraction for VR streaming,” in *ACM NOSSDAV*, 2018.
- [85] J. Son, D. Jang, and E.-S. Ryu, “Implementing 360 video tiled streaming system,” in *ACM MMSys*, 2018.
- [86] L. Xie, X. Zhang, and Z. Guo, “CLS: A cross-user learning based system for improving qoe in 360-degree video adaptive streaming,” in *ACM Multimedia*, 2018.
- [87] L. Xie, Z. Xu, Y. Ban, X. Zhang, and Z. Guo, “360ProbDASH: Improving QoE of 360 video streaming using tile-based HTTP adaptive streaming,” in *ACM Multimedia*, 2017.
- [88] E. Kurutepe, M. R. Civanlar, and A. M. Tekalp, “Client-driven selective streaming of multiview video for interactive 3dtv,” *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 17, no. 11, pp. 1558–1565, 2007.
- [89] D. Monakhov, I. D. Curcio, and S. Mate, “On data wastage in viewport-dependent streaming,” in *IEEE MMSP*, 2019.
- [90] J. Fu, X. Chen, Z. Zhang, S. Wu, and Z. Chen, “360SRL: A sequential reinforcement learning approach for ABR tile-based 360 video streaming,” in *IEEE ICME*, 2019.
- [91] X. Jiang, Y.-H. Chiang, Y. Zhao, and Y. Ji, “Plato: Learning-based adaptive streaming of 360-degree videos,” in *IEEE LCN*, 2018.
- [92] A. T. Nasrabadi, A. Samiei, and R. Prakash, “Viewport prediction for 360° videos: A clustering approach,” in *ACM NOSSDAV*, 2020.
- [93] C. Ozcinar, A. De Abreu, and A. Smolic, “Viewport-aware adaptive 360° video streaming using tiles for virtual reality,” in *IEEE ICIP*, 2017.
- [94] F. Qian, B. Han, Q. Xiao, and V. Gopalakrishnan, “Flare: Practical viewport-adaptive 360-degree video streaming for mobile devices,” in *ACM MobiCom*, 2018.
- [95] D. V. Nguyen, H. T. T. Tran, and T. C. Thang, “An evaluation of tile selection methods for viewport-adaptive streaming of 360-degree video,” *ACM Trans. Multimedia Comput. Commun. Appl.*, vol. 16, mar 2020.



- [96] G. Van Brummelen, *Heavenly Mathematics: The Forgotten Art of Spherical Trigonometry*. Princeton University Press, 2013.
- [97] S. Ahsan, A. Hourunranta, I. D. D. Curcio, and E. Aksu, “FriSBE: Adaptive bit rate streaming of immersive tiled video,” in *Packet Video Workshop*, 2020.
- [98] M. Hostetter, D. A. Kranz, C. Seed, C. Terman, and S. Ward, “Curl: a gentle slope language for the web.,” *World Wide Web Journal*, vol. 2, no. 2, pp. 121–134, 1997.
- [99] X. Corbillon, F. De Simone, and G. Simon, “360-degree video head movement dataset,” in *ACM MMSys*, 2017.



# VITA

Mehmet N. Akcay received his B.Sc., in the field of Computer Engineering, from Istanbul Technical University in 2005. He completed his M.Sc. in the same field in Bogazici University in 2008 and he has been working in the industry for more than 15 years. His research interests are HTTP adaptive streaming, low-latency live streaming and 360-degree video streaming.

## Publications

- M. N. Akcay, B. Kara, S. Ahsan, A. C. Begen, I. Curcio, and E. Aksu, “Rate-Adaptive Streaming of 360-Degree Videos with Head-Motion-Aware Viewport Margins,” in *IEEE MIPR*, 2022 (DOI: 10.1109/MIPR54900.2022.00056).
- M. Lim, M. N. Akcay, A. Bentaleb, A. C. Begen, and R. Zimmermann, “The benefits of server hinting when DASHing or HLSing,” in *ACM MHV*, 2022 (DOI: 10.1145/3510450.3517317).
- A. C. Begen, M. N. Akcay, A. Bentaleb, and A. Giladi, “Adaptive streaming of content-aware-encoded videos in dash.js,” *SMPTE Motion Imaging Jour.*, vol. 131, no. 4, pp. 30–38, 2022 (DOI: 10.5594/JMI.2022.3160560).
- M. N. Akcay, B. Kara, S. Ahsan, A. C. Begen, I. Curcio, and E. Aksu, “Head-motion-aware viewport margins for improving user experience in immersive video,” in *ACM Multimedia Asia*, 2021 (DOI: 10.1145/3469877.3490573).
- A. Bentaleb, M. N. Akcay, M. Lim, A. C. Begen, and R. Zimmermann, “Catching the Moment With LoL<sup>+</sup> in Twitch-Like Low-Latency Live Streaming Platforms,” *IEEE Transactions on Multimedia*, vol. 24, pp. 2300–2314, 2022.
- A. Bentaleb, M. Lim, M. N. Akcay, A. C. Begen, and R. Zimmermann, “Common media client data (CMCD): Initial findings,” in *ACM NOSSDAV*, 2021 (DOI: 10.1145/3458306.3461444).
- O. F. Aladag, D. Ugur, M. N. Akcay, and A. C. Begen, “Content-aware playback speed control for low-latency live streaming of sports,” in *ACM MMSys*, 2021 (DOI: 10.1145/3458305.3478437).
- M. Lim, M. N. Akcay, A. Bentaleb, A. C. Begen, and R. Zimmermann, “When They Go High, We Go Low: Low-Latency Live Streaming in dash.js with LoL,” in *ACM MMSys*, 2020 (DOI: 10.1145/3339825.3397043).
- K. Durak, M. N. Akcay, Y. K. Erinc, B. Pekel, and A. C. Begen, “Evaluating the performance of Apple’s low-latency HLS,” in *IEEE MMSP*, 2020 (DOI: 10.1109/MMSP48831.2020.9287117).