DISS. ETH NO. 28749

Understanding, reconstructing and optimising adaptive bitrate video streaming

A thesis submitted to attain the degree of
DOCTOR OF SCIENCES OF ETH ZURICH
(Dr. sc. ETH Zurich)

presented by

MELISSA LICCIARDELLO

Master of Science in Telecommunications Engineering,

University of Bologna

born on 15. January 1992

citizen of Bologna, Italy

accepted on the recommendation of
Prof. Dr. Timothy Roscoe (ETH Zurich), examiner
Prof. Dr. Aruna Balasubramanian (Stony Brook University), co-examiner
Dr. Ankit Singla (Google), co-examiner
2022

Abstract

Video streaming applications account for the majority of Internet traffic nowadays. For this reason, video streaming system optimisation is of utmost importance, with interesting new approaches proposed every few months from both industry and academia. In this dissertation, we identify new research opportunities for understanding, reconstructing, and optimising video streaming applications through network measurements.

Understanding video streaming applications is critical, as little is known about the behaviour of the streaming algorithms deployed across large online streaming platforms. We thus study adaptive bitrate streaming algorithms in use at 10 video platforms with diverse target audiences. We find that deployed algorithms exhibit a wide spectrum of behaviours across different optimization axes, indicating the lack of a consensus one-size-fits-all solution.

Even with a qualitative understanding of video providers' optimisation goals, their implemented adaptation logic is still opaque. This creates obvious hurdles for research on streaming algorithms and their interactions with other network traffic and control loops like that of transport and traffic throttling. To address this gap, in this dissertation we tackle the reconstruction of unknown proprietary video streaming algorithms in a human interpretable fashion through network observation. We find that, out of 10 popular streaming platforms, we can

produce easy-to-understand, and high-accuracy reconstructions for 7 of them.

Based on our analysis of academic and industrial video streaming algorithms, we identify new opportunities for optimizing adaptive bitrate streaming systems using network observations. Specifically, we can tune *offline* video chunking depending on the expected *online* playback behaviour and rate adaptation. Due to video's varying complexity over time, we observe that certain parts are more likely to cause performance impairments during playback with a given rate adaptation algorithm. We propose SEGUE, which uses variable-length video segments, and augment specific segments with additional bitrate tracks. Our network behaviour aware methodology substantially reduces rebuffering and quality fluctuations, while maintaining video quality delivered; SEGUE improves QoE by 9% on average, and by 22% in low-bandwidth conditions.

Zusammenfassung

Videostreaming-Anwendungen machen heutzutage den Grossteil des Datenverkehrs im Internet aus. Deshalb ist die Optimierung von Videostreaming-Systemen von grösster Bedeutung und fast monatlich werden neue interessante Ansätze aus Industrie und Wissenschaft vorgeschlagen. In dieser Dissertation identifizieren wir neue Forschungsmöglichkeiten für das Verständnis, Rekonstruktion und Optimierung von Videostreaming-Anwendungen durch Netzwerkmessungen.

Videostreaming-Anwendungen zu verstehen ist entscheidend, da wenig über das Verhalten der Streaming-Algorithmen bekannt ist, welche in den grossen Online-Streaming-Plattformen verwendet werden. Wir untersuchen daher adaptive Bitraten-Streaming-Algorithmen von 10 Videoplattformen mit unterschiedlichen Zielgruppen. Wir stellen fest, dass die eingesetzten Algorithmen ein breites Spektrum an Verhaltensweisen über verschiedene Optimierungsachsen zeigen, was darauf hindeutet, dass es keinen Konsens über eine Einheitslösung gibt.

Selbst mit einem qualitativen Verständnis der Optimierungsziele von Videoanbietern ist ihre implementierte Anpassungslogik immer noch undurchsichtig. Dies hindert die Erforschung von Streaming-Algorithmen und deren Interaktionen mit anderen Netzwerkverkehrs- und Kontrollschleifen wie der Flusskontrolle des Transportprotokolls oder der Traffic-drosselung. Um diese Lücke zu schliessen, wird in

dieser Dissertation die Rekonstruktion von unbekannten proprietären Videostreaming-Algorithmen in einer für den Menschen interpretierbaren Weise durch Netzwerkbeobachtung vorgestellt. Wir stellen fest, dass wir von 10 populären Streaming-Plattformen für 7 leicht verständliche und hochpräzise Rekonstruktionen erstellen können.

Auf der Grundlage unserer Analyse von akademischen und industriellen Video-Streaming-Algorithmen, identifizieren wir neue Möglichkeiten zur Optimierung von adaptiven Bitraten-Streaming-Systemen anhand von Netzwerkbeobachtungen. Insbesondere können wir *offline* das Video-Chunking in Abhängigkeit des erwarteten *Online*-Wiedergabeverhalten und der Bitratenanpassung abstimmen. Aufgrund der im Laufe der Zeit variierenden Komplexität von Videos beobachten wir, dass bestimmte Teile eher zu Leistungseinbussen bei der Wiedergabe mit einem bestimmten Algorithmus neigen. Wir schlagen SEGUE vor, welches Videosegmente mit variabler Länge verwendet, und ergänzen bestimmte Segmente mit zusätzlichen Bitratenspuren. Unsere Methodik, die das Netzwerkverhalten berücksichtigt, reduziert das Rebuffering und die Qualitätsschwankungen bei gleichzeitiger Beibehaltung der gelieferten Videoqualität; SEGUE verbessert die `QoE` im Durchschnitt um 9%, und um 22% unter Bedingungen mit geringer Bandbreite.

Acknowledgments

Well, this is going to be long.

Thank you, Ankit. Thanks for giving me the huge opportunity to work at ETH. I still cannot wrap my head around the huge trust that you put in me. Thank you for being such a supportive and encouraging PhD advisor. I am, and I will always be, so proud of having been advised by you.

Thank you, Mothy. Thanks for "adopting" me at the end of my PhD journey, thanks for supporting me and making me always feel welcome. Thanks for teaching me that, sometimes, in systems as in life, we just have to *embrace the mess*.

Thank you, Aruna. Thanks for accepting being part of my committee. Thanks for all the precious advices and wonderful conversations of the past year. Knowing that I could always "knock to your *Zoom* door" helped me so much throughout the end of this chapter of my life.

Thank you, Simonetta. Thank you, Nadia. Thank you, Jena. Thanks for being so patient with me and my allergy for bureaucracy. Thanks for always helping me throughout my internationally acknowledged disorganisation and never making me feel bad about it. I know that you will miss my tidy desk :)

Thanks to Christopher Schroers and all the Disney folks for giving me the incredible opportunity to work with them at Disney Research.

Thank you all for teaching me so much, and making me realize at the ripe age of 30 that I'm definitely better as a researcher than as a Disney Princess (sigh).

Thanks to Debopam, Vojislav, Max, Michael, Anastasiia and Johannes. You have been much more than colleagues, you have been (and still are) my wonderful family.

Thanks to Isma, Giulia, Claudia, Eva and Silvia, my forever best friends. You, more than any other in my life, teach me everyday how distance is just a number.

Thank you, Luki. If I would have to list all the reasons I am grateful for having you in my life, I would have to double the length of this thesis, so I'll try to make it short: thanks for being that person in my life that brings out of the best of me everyday. I love you.

Last, and more importantly, thank you Maya. Being the mom of such a wonderful human as you are is what makes my eyes shine everyday. Parenting while doing a PhD program has been a struggle sometimes, and never in a million year I would have imagined that my small toddler would have been my number one supporter. I love you, topolina.

Contents

1 Introduction **13**

1.0.1 Related publications 16

2 Background **19**

2.1 VOD streaming: an architectural overview 19

2.2 Offline: Video preparation 21

2.2.1 Video encoding 21

2.2.2 Encoding modes 23

2.2.3 Quality metrics 25

2.3 Online: Video delivery 30

2.3.1 DASH 31

2.3.2 Quality of Experience 32

2.3.3 Adaptive bitrate algorithms 34

3 Understanding video streaming algorithms **39**

3.1 Introduction 39

3.2 Related Work 41

3.3 Methodology 43

3.3.1 Experimental setup 44

3.3.2 The proxy's impact on measurements 45

3.3.3 Metrics of interest 46

3.3.4 Measurement coverage 48

3.4 Measurement results 51
3.5 Limitations and future work 58
3.6 Conclusion 59

4 Reconstructing video streaming algorithms 61
4.1 Introduction 61
4.2 Related work 65
4.3 Data preparation 67
4.4 Rule-set based inference 68
4.4.1 Motivating example 69
4.4.2 Decision trees and rules 70
4.4.3 Feature engineering 72
4.5 Implementation 75
4.6 Evaluation 78
4.6.1 Experimental methodology 78
4.6.2 Results 82
4.7 The utility of interpretability 90
4.7.1 Examining two reconstructions 92
4.8 Limitations & future work 94
4.9 Conclusion 95

5 Optimising video streaming systems with SEGUE 97
5.1 Introduction 97
5.2 Background and related work 100
5.2.1 Video streaming 101 100
5.2.2 Related work 102
5.3 New opportunities in streaming 104
5.3.1 What levers can we tune? 106
5.3.2 The need of variability 110
5.4 SEGUE design 110
5.4.1 Segmentation 111
5.4.2 Augmentation 114
5.5 Implementation 116
5.5.1 Offline video chunking 116
5.5.2 Online playback and rate adaptation 119

5.6 Evaluation methodology 120
5.7 Results 123
5.7.1 SEGUE's segmentation 123
5.7.2 SEGUE's augmentation 126
5.7.3 The impact of the adaptation algorithm . . . 127
5.7.4 The impact of the video 131
5.7.5 Performance comparison with related works . 133
5.7.6 SEGUE's computational cost 137
5.7.7 *dash.js* player implementation 138
5.7.8 Summary of results 141
5.8 Discussion and Future Work 142
5.9 Conclusion 146

6 Conclusions and suggested future works 147

1 Introduction

What do we use the Internet for?

In recent years, the answer to this question is non-trivial. A plethora of Internet applications became *embedded* in our daily life. At the same time, not all Internet applications are equally complex and bandwidth hungry. Among all the possible services that the Internet offers us, streaming video over the Internet has been, and still is, one of the most popular and demanding applications. A recent report from Sandvine [San22] states that, in the first half of 2021, 53.72% of the overall Internet traffic was video.

Video streaming is an umbrella term that includes multiple different applications, each of which poses different challenges and optimisation opportunities under a system design perspective. For example, Video-On-Demand (`VOD`) streaming systems require different encoding and delivery strategies compared to live video streaming and conferencing. Similarly, rising Virtual Reality video streaming applications [Tec22] pose additional challenges compared to standard 2D streaming.

In this dissertation, we focus on understanding and optimising 2D VOD streaming services. As [San22] reports, this is still the most common video streaming service scenario, as the overall Internet capacity usage is still dominated by YouTube and Netflix, which account 16.37% and 10.61% of the downstream traffic respectively.

Video on Demand systems are usually split into two pipelines, one *offline*, that involves mostly encoding operations, and one *online*, which includes, for example, adaptive bitrate streaming algorithms.

Offline, the video is usually segmented into equal-length segments (usually 4 to 6 seconds long), and each segment is encoded at multiple quality levels.

During online playback, an adaptive bitrate algorithm (ABR) chooses which quality to fetch the next segment, depending on the client's estimated bandwidth, playback buffer occupancy, and features of upcoming video segments. ABR control loops are usually implemented client side. This architecture is commonly developed according to the Dynamic Adaptive Streaming over HTTP (DASH) standard.

The goal of such adaptation is to improve the client's quality of experience (QoE) by delivering the highest quality video, without pauses (referred usually as *rebuffering events*) and infrequent quality switching. These metrics deeply affect user experience and, as a consequence, streaming companies revenues. For example, a recent study of Limelight [Aka19] suggests that, if experiencing rebuffering, 28% of the users abandon the streaming session.

During the past years, several high quality research proposals described different strategies to optimise different parts of such pipelines. Such trends cover various aspects of video streaming systems, ranging from bitrate ladders optimisation ([Aar+], [Net18a]), to ABR algorithms implementation ([Qin+18; MNA17b; Mil+15]) and transport layer ([Nat+19]) design.

In this dissertation, we tackle the problem from a different perspective. Rather than targeting the optimisation of a specific subsystem of VOD pipelines, we show how network measurements and behaviour can be effectively utilised to *understand*, *reconstruct* and *optimise* video streaming services.

Understanding video streaming is of pivotal importance, as little is known about the behaviour of the streaming algorithms deployed across large online streaming platforms. In-depth knowledge of major providers deployed solutions can allow researchers into better tuning their objective function depending on the type of service and users. In Chapter 3 we study adaptive bitrate streaming algorithms in use at 10 video platforms with diverse target audiences. To do so, we rely on network observation: we collect traces of each video player's response to controlled variations in network bandwidth, and examine the algorithmic behaviuor. We find that deployed algorithms exhibit a wide spectrum of behaviours across different axes, indicating the lack of a single one-size-fits-all solution. We also find evidence that most deployed algorithms are tuned towards stable behavior rather than fast adaptation to bandwidth variations, some are tuned towards a visual perception metric rather than a bitrate-based metric, and many leave a surprisingly large amount of the available bandwidth unused.

Even with the knowledge of providers' optimisation goals, it can be hard for academia to test innovative solutions against the already deployed ones. In this context, *reconstruction* of (proprietary and unknown) video streaming algorithms can be a valuable resource to fill such a gap. Also, instead of opaque reconstruction through, e.g., neural networks, we seek reconstructions that are easily understandable and open to inspection by domain experts. Such reconstruction, if successful, would also shed light on the risk of competitors copying painstakingly engineered algorithmic work simply by interacting with popular services. We describe our reconstruction approach Chapter 4. Such approach makes extensive use of logs of player and network state and observed player actions across a variety of network traces and videos. The goal is to learn decision trees using streaming-specific engineered features. We find that, of 10 popular streaming platforms, we can produce easy-to-understand, and high-accuracy reconstructions for 7 using concise decision trees with no more than 20 rules. In addition, we explain the limitations of our approach as applied to the other 3.

Based on our analysis of streaming provider behaviour, we then show how we can *optimise* video streaming service with the joint consideration of offline video chunking and online rate adaptation. We observe that, due to a video's complexity varying over time, certain parts of the videos are more likely to cause performance impairments during playback with a particular rate adaptation algorithm. To address such an issue, we propose SEGUE, a novel system that we discuss in Chapter 5. SEGUE carefully uses variable-length video segments, and augments specific segments with additional bitrate tracks. The key novelty of our approach is in making such decisions based on the video's time-varying complexity and the expected rate adaptation behavior over time. We propose and implement several methods for such adaptation-aware chunking. Our results show that SEGUE substantially reduces rebuffering and quality fluctuations, while maintaining video quality delivered; SEGUE improves QoE by 9% on average, and by 22% in low-bandwidth conditions.

To summarise, this dissertation provides 3 main contributions:

- We show how network measurements and player behaviour can offer precious hints on providers optimisation goals.
- We present a methodology for reconstructing unknown adaptive bitrate streaming algorithms in an interpretable fashion via network and player state observation.
- We develop and discuss in details SEGUE, a systems that optimise the offline part of `VOD` streaming pipeline depending on the expected online network behavior.

1.0.1 Related publications

This dissertation is based on the following list of published papers and thesis.

[Wes18] Dimitri Wessels. “Quantifying and Explaining Unfairness in Online Video Streaming”. Bachelor’s Thesis. Department of Computer Science, ETH Zurich, 2018.

[LGS20] Melissa Licciardello, Maximilian Grüner, and Ankit Singla. “Understanding video streaming algorithms in the wild”. In: *PAM*. 2020.

[Grü19] Maximilian Grüner. “Human-interpretable auto-inference of networked algorithms”. Master’s Thesis. Department of Computer Science, ETH Zurich, 2019.

[GLS20a] Maximilian Grüner, Melissa Licciardello, and Ankit Singla. “Reconstructing proprietary video streaming algorithms”. In: *2020 USENIX Annual Technical Conference (USENIX ATC 20)*. July 2020.

[Roh21] Fabian Rohr. “CAVA on Segue and Segue in DASH: Verifying Simulated ABR Behaviour on the DASH-IF Reference Player”. Master’s Thesis. Department of Computer Science, ETH Zurich, 2021.

[Lic+22a] Melissa Licciardello, Lukas Humbel, Fabian Rohr, Maximilian Grüner, and Ankit Singla. “Prepare Your Video for Streaming with Segue”. In: *Journal of Systems Research* 2.1 (July 2022).

In particular, Chapter 3 was originally inspired by the bachelor thesis [Wes18] and then extended with the measurement work published in [LGS20]. I am joint first author with Maximilian Grüner. The measurement system was originally implemented by me, and Maximilian extended it to support multiple services as a part of his master thesis, that I co-supervised with Dr. Ankit Singla.

Chapter 4 is based on the master thesis [Grü19] that lead to the publication [GLS20a]. This work was performed in collaboration with Maximilian Grüner, whose master thesis I served as the primary supervisor for. Besides guiding the development of the approach we took, I also implemented the basic infrastructure for collecting measurement data for the learning-based study, as well as experiments with reconstructed models in the DASH player. I will be always grateful of having had the honour of supervising such a brilliant student like Maximilian. Writing this paper with him was one of the happiest moments I spent during my PhD.

Chapter 5 is based on the publication [Lic+22a], and uses part of the implementation discussed in the master thesis [Roh21]. I would like to thank, from the bottom of my heart, my partner and colleague Lukas Humbel for the infinite and meaningful discussion over the months that helped finalise this work. I would also like to thank Fabian Rohr, who trusted me to supervise his master thesis [Roh21], one of which main core points is the dash.js implementation of SEGUE. Thanks, again, to my friend and paper fellow Maximilian Grüner, whose inputs have been fundamental. Last but not the least, thanks to Dr. Ankit Singla, that supported me throughout the full SEGUE's journey and gave me full trust throughout my PhD.

2
Background

In this chapter we will introduce the main concepts behind the dissertation. In Section 2.1 VOD adaptive streaming architecture is described. In Section 2.2 we will focus on the description of the offline part of the pipeline, while in Section 2.3 we will discuss the aspects that involves the video online delivery.

2.1 VOD streaming: an architectural overview

VOD adaptive streaming systems are usually split into two pipelines, one offline and one online (Fig. 2.1). During the offline stage, the *original*, ideally uncompressed, video content is rescaled and encoded at different resolution-bitrate pairs and segmented into chunks. Chunks are usually of equal length, and their length is usually between 4 to

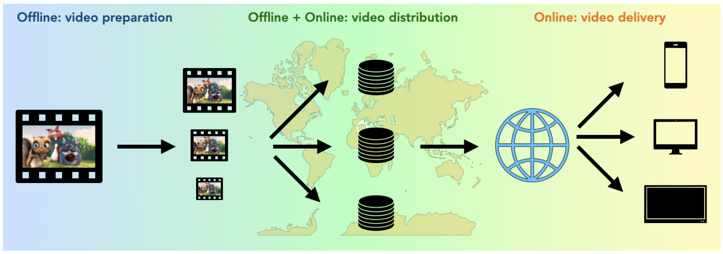

Offline: video preparation
Offline + Online: video distribution
Online: video delivery

Figure 2.1: VOD streaming architecture involve multiple steps. Video preparation usually happens offline. In this stage, the content is re-encoded at multiple bitrate-resolution pairs and partitioned into few seconds long segments. The content then is distributed and replicated in Content Delivery Networks (CDN).CDN's optimisation strategies can involve both offline and online content distribution algorithms. These control loops are out of the scope of this dissertation. Online, users' then fetch video segments from the closest CDN. An ABR algorithm decides the quality to fetch the next segment at.

6 seconds. The degree of complexity and optimisation of the offline stage highly depends on the video streaming providers use case.

Later, the video content is replicated and distributed across the Content Delivery Networks (CDNs), with the twofold goal of failure resiliency and of pushing the content near the users to reduce communication delays.

During online playback, the user web player requests via HTTP each video segment. An adaptive bitrate algorithm (ABR), that usually runs on the client side, decides the quality of each video segment depending on network fluctuations and web player state. The goal of such an adaptation is to download the highest video quality as possible without incurring abrupt playback interruptions (also called rebuffering events) and frequent quality switches.

In the upcoming sections of this dissertation we will focus in greater details on the offline (Section 2.2) and online (Section 2.3) part of such a pipeline.

2.2 Offline: Video preparation

While this dissertation mostly focuses on the online aspects of VOD adaptive streaming, it is still useful to introduce some basic concepts of video preprocessing. In Section 2.2.1 we will briefly discuss the concept of video encoding, while in Section 2.2.2 we will describe the most relevant encoding modes and parameters used in the streaming context. In Section 2.2.3 we will then introduce to the reader the most relevant metrics to measure video quality.

2.2.1 Video encoding

Video encoding is defined as the process of converting a digital video from one format into another. Digital videos, in fact, exist into different formats, that differ, for example, in *codecs* and *containers*.

A *video codec* is a software that *compresses* (and *decompresses*) the bitstream in order to decrease the required storage space and to speed up the transmission. A codec can be either lossless (i.e. the original bitstream can be perfectly recovered from the compressed one) or lossy (i.e. the bitstream recovered from the compressed one suffers from distortion with respect to the original one). Due to the better compression ratio, lossy codecs are more suitable for VOD streaming use cases. Some example are H.264 [FFMb], H.265 [FFMc] and AV1 [FFMa]. In this dissertation we will mostly work with H.264, a codec that was published in 2004 but it is by far the most widely used video format for the compression and distribution of video content (used by 91% of the video industry as of September 2019) [Bit19b].

The reason why H.264 is still so popular despite newer standards with better encoding efficiency is because of the increased computational complexity of H.265 and AV1, that leads to substantially higher encoding and decoding time [Sim21]. H.264 has also broad hardware support.

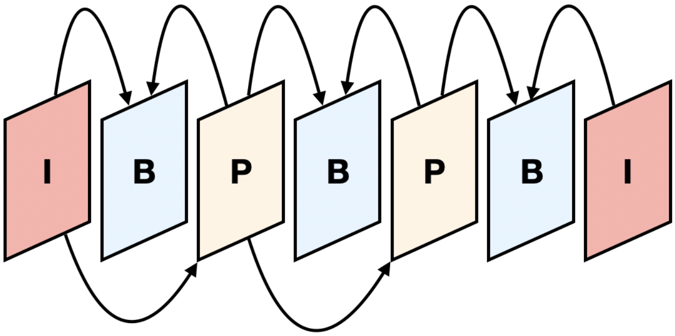


Figure 2.2: Diagram of the prediction relationship between I-Frame, P-Frame and B-Frame.

H.264 uses a differential compression technique: The current frame is reconstructed based on one (or multiple) previous frames or differences between those frames in time (Fig. 2.2). The standard provides three different types of frame:

- **I-Frame**: Referred also as **key-frames**, the frame is *intra* predicted, in other words the frame can be reconstructed independently on the others.
- **P-Frame**: The frame is created based on the information about changes between subsequent P or I frames (*inter* predicted).
- **B-Frame**: The frame is coded using two reference frames, one before and one after the current one in the video sequence (*inter* predicted).

A *video container* is a wrapper that stores the required metadata and that can embed multiple bitstreams, for example video and audio, into a single file. In this dissertation we will mostly work with the MP4 container, but other notable examples, like MOV, TS or OGG, exist and are widely used.

2.2.2 Encoding modes

Video encoding is a complex process that involves a plethora of parameters. In this section we will discuss the most relevant ones in the context of VOD adaptive streaming.

A fundamental difference in the pipeline for video on demand adaptive streaming is whether the video is segmented into chunks *before* or *after* the encoding stage. Both system choices are viable and present different advantages.

Segmenting the original video before the encoding process and encoding each segment separately allows better fine tuning of encoding parameters depending on the specific (short) video sequence. Netflix dynamic optimiser [Net18a] is an example of such design.

Encoding the video as a whole and then splitting it into segments allows the encoder to have larger look ahead and then better optimise certain parameters (*e.g.,* the average target bitrate of the video stream). This is the most common approach, and, as a consequence, is the one that is described and used in the dissertation.

In video coding, a *group of picture* (GOP) is a collection of successive frames within a coded video bitstream, and specifies the order in which intra-predicted and inter-predicted frames are arranged. A GOP always starts with an I-Frame, and then includes the subsequent P and B frames. GOPs can be encoded into two different modes: *open* GOPs and *closed* GOPs. In open GOPs B and P frames can use reference frames in other GOPs for redundant blocks. Conversely, in closed GOPs, B and P frame can reference frames only inside their GOP.

In adaptive bitrate streaming, each chunk needs to be independent from the others, given that the player is allowed (and encouraged!) to change bitrate track. For this reason, if the splitting into chunks happens *after* the encoding process, the video should be encoded with *closed* GOPs.

One of the most important parts of the encoding pipeline for video streaming delivery systems is rate control. By rate control we mean the control loop that decides how many bits to spend for a given frame (and coding blocks in a frame), in order to find a balance between the allocated resources and the frame distortion. This tradeoff is fundamental for streaming applications, as video segments need to be streamed across a capacity-constrained network.

There are several modes of rate control, that we describe in the following paragraphs. First of all, it is important to stress the difference between Constant Bitrate encoding (CBR) and Variable Bitrate encoding (VBR). While CBR allocates, within a certain frames' window, the same amount of resources independently of the content, VBR distributes the bits depending on how easy or hard it is to compress certain parts of the video. In video streaming applications, usually VBR is preferred over CBR, as the first ensures a better resource efficiency (at the cost of less predictability of bitrate fluctuations during the adaptive bitrate process).

A type of VBR encoding is Constant Quantisation Parameter (CQP). The quantisation parameters controls the compression level of each coding unit. In H.264 quantisation parameters vary from 0 to 51, where higher quantisation value means higher level of compression. This VBR mode is usually not used for adaptive bitrate streaming, as the bitrate can largely vary depending on the complexity of the scene. However, Netflix uses this rate control mode [Net18a] for their per-shot encoding pipeline.

Constant Rate Factor (CRF) encoding takes as an input a value from 0 to 51 in H.264 and ensures a certain constant quality throughout the video frames. Similarly to CQP, the bitrate can vary substantially throughout the video, and it is usually not used for video streaming applications.

1-Pass Average Bitrate (`1-Pass ABR`) encoding mode takes as an input an average target bitrate. The encoder then tries to reach it, but given the limited knowledge of the future frames it usually strives to optimally distribute the bitrate budget across the frames.

On the other hand, 2-Pass Average Bitrate (`2-Pass ABR`) scans the whole video before starting the encoding process. Due to the multiple pass, it is slower than `1-Pass ABR`, but it ensures a better quality under certain bitrate constraints. This is usually the preferred choice for adaptive bitrate streaming, and it is the encoding technique utilised throughout this dissertation.

For `ABR` modes, a maximum bitrate and a minimum bitrate given a certain buffer size (i.e. the interval at which the bitrate is measured) can also be passed as a parameter. The value of buffer size sets how much variability is tolerated in the bitrate.

The target bitrate is a fundamental parameter for video streaming applications. Deciding which target bitrate given a specific video sequence is a widely explored area of research. Given a certain resolution, the target bitrate can be specified *statically*, i.e. given a conventional value, or it can be set according to a *rate distortion optimisation* process. Rate distortion optimisation means trying to find an optimal bitrate that reach a certain satisfactory video quality, trading off the bitrate, and usually highly depends on the video (or video portion) content and complexity. Several approaches construct rate distortion curves and brute force the selection of the bitrate given certain tradeoff criteria [Net18a] [Aar+]. This process can be extremely slow and resource consuming, and it is usually utilised by streaming providers that do not deal with user uploaded content.

In order to give an idea of the performance differences among different rate control methodologies, Fig. 2.3 plots the rate distortion curves of Blender Sintel Trailer [Ble10] encoded at 1920×1080 resolution with different target bitrates using the default H.264 library of *ffmpeg* (*libx264*).

2.2.3 Quality metrics

One of the main objectives of video streaming control loops is to download the video at the *highest quality possible*. While the bitrate of the videos correlates to the (perceived) video quality, the amount

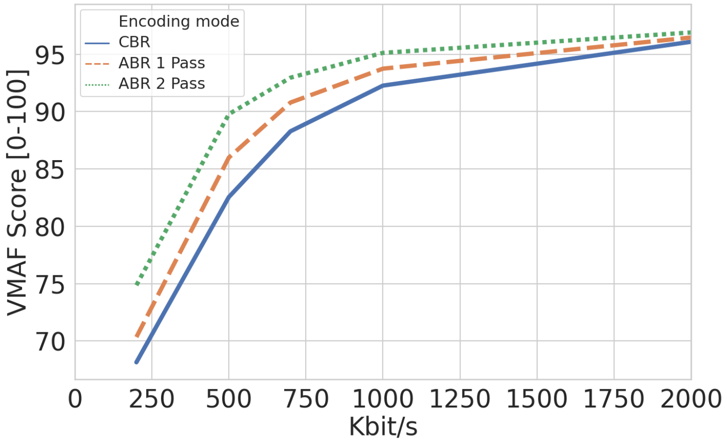


Figure 2.3: Blender Sintel Trailer [Ble10] rate distortion curves varying the target bitrate from 200 Kbit/s to 2 Mbit/s with *libx264* library for `CBR`, `1-Pass ABR` and `2-Pass ABR` rate control modes. The quality metric under test is Netflix VMAF [Nat+19; Li+16], described in section 2.2.3. Both `ABR` rate control modes have been configuted to use, as maximum rate, two times the target rate. `2-Pass ABR` is able to reach an higher perceptual quality for the same target bitrate with respect to both `1-Pass ABR` and `CBR`, at the cost of higher encoding time (+35% in average for the depicted benchmark).

of distortion perceived cannot be easily quantified by this value alone. There are several different ways to measure the quality of a video. They can be mainly classified as *objective* and *subjective*. While the first class takes into account mostly mathematical formulation of distortion, the latter tries to model subjective user experience. The most common objective quality metric is Peak Signal to Noise Ratio (`PSNR`). `PSNR` is defined via the mean squared error (`MSE`). Given a noise-free $m \times n$ monochrome image I and its distorted version K, MSE is defined as:

$$\text{MSE} = \frac{1}{m \cdot n} \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} [I(i,j) - K(i,j)]^2$$

Then the PSNR is defined, in dB:

$$\text{PSNR} = 10 \cdot log_{10}(\frac{MAX_I^2}{MSE})$$

where MAX_I is the upper bound of the possible pixel values.

While widely used to assess video quality, PSNR shows some limitations in correlating with visual perception, as represented in Fig. 2.4.

Another common video quality metric is structural similarity index (SSIM). SSIM is a metric for predicting the *perceived quality* of a video, and it quantifies the similarity of frames. SSIM can be calculated on various windows of images and it is based on three main components: *luminance* (l), *contrast* (c) and *structure* (s). The individual formulas of these components are, given an equal size window of $N \times N$ pixels:

$$l(x, y) = \frac{2 \cdot \mu_x \mu_y + c_1}{\mu_x^2 + \mu_y^2 + c_1}$$

$$c(x, y) = \frac{2\sigma_x \sigma_y + c_2}{\sigma_x^2 + \sigma_y^2 + c_2}$$

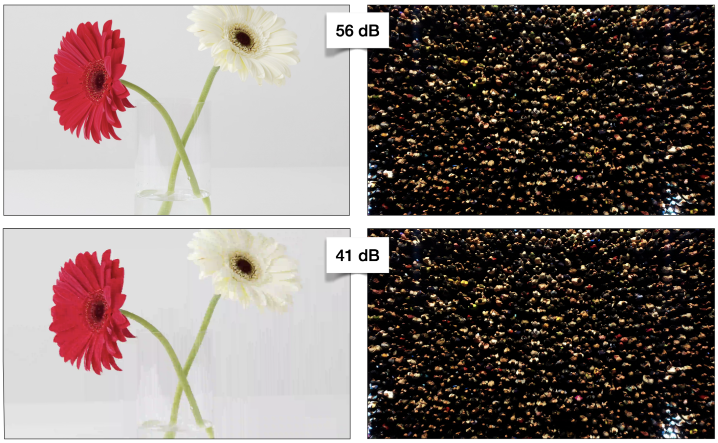


Figure 2.4: Two different video contents have been encoded at different CRF values. The content on the left represent a video of a vase of flower with a static white background, while the video on the right records a slowly moving crowd. We encoded these video at various CRF values and we then extracted the first frame of each video at two equal values of PSNR, one high (56dB, top) and one low (41dB, botton). The drop in quality from 56dB to 41dB is much more noticeable in the video content on the left, while it is barely noticeable on the content on the right.

$$s(x, y) = \frac{\sigma_{xy} + c_3}{\sigma_x \sigma_y + c_3}$$

where:

- μ_x is the average of x
- μ_y is the average of y

- σ_x is the variance of x
- σ_y is the variance of y
- σ_{xy} is the covariance of x and y
- $c_1 = (k_1 \cdot L)^2$, $c_2 = (k_2 \cdot L)^2$
- $c_3 = \frac{c_2}{2}$
- L is the dynamic range of the pixel values (usually $2^{\#bitspp} - 1$)
- $k_1 = 0.01$ and $k_2 = 0.03$

The `SSIM` weights then these components as:

$$\mathrm{SSIM}(x, y) = [l(x, y)^{\lambda} \cdot c(x, y)^{\beta} \cdot s(x, y)^{\gamma}]$$

Setting the exponents λ, β and γ to 1 leads to the formulation:

$$\mathrm{SSIM}(x, y) = \frac{(2 \cdot \mu_x \cdot \mu_y + c_1) \cdot (2 \cdot \sigma_{xy} + c_2)}{(\mu_x^2 + \mu_y^2 + c_2) \cdot (\sigma_x^2 + \sigma_y^2 + c_2)}$$

One of the most widely used perceptual quality metric to measure video distortion is the Emmy Awards winning Video Multi-Method Assessment Fusion (`VMAF`) [Nat+19; Li+16]. `VMAF` is an objective full-reference video quality metric developed by Netflix and University of Southern California. Given its huge popularity, `VMAF` will be predominantly used throughout this dissertation.

`VMAF` is based on a machine learning model. For training purposes, a dataset of user observations is used. Specifically, users were asked to rate distorted video sequence in a living room-like environment on a scale to 1 (very annoying) to 5 (not noticeable). The scores then were combined to generate Differential Mean Opinion Score (`DMOS`) on a scale from 0 to 100. `VMAF` predicts the video distortion per frame compared to the reference using the following features:

- Visual Information Fidelity (`VIF`): accounts the information fidelity loss at four different spatial scales.
- Detail Loss Metric (`DLM`): measures details losses and noises that can affect viewer attention.
- Mean Co-Located Pixel Difference (`MCPD`): measures the temporal difference between frames on the luminance component.

The features are then fed into a Support Vector Machine (`SVM`) regressor. `VMAF` models are intrinsically bounded on the `DMOS` retrieval methodology, i.e. the viewing conditions. In this dissertation we are going to use mainly three different versions: the standard model, the `VMAF 4K` model and the `VMAF mobile` model.

The standard `VMAF` model operates under the assumption that the viewer sits in front of a 1080p TV with the viewing distance of 3 times the screen height. This differentiate from the `VMAF 4K` model, where the subjective quality of video is predicted based on the assumption that the content is displayed on a 4K TV and viewed from a distance of 1.5 times the screen height. For training the phone model, the viewer where instead instructed to position the 1080p phone screen at a distance in which they felt comfortable. Compared to both the standard and 4K model, the phone model has been shown to be less sensitive to the differences in term of delivered video quality between 720p and 1080p.

2.3 Online: Video delivery

In this section, we will introduce the reader to the main concepts of online video delivery. In Section 2.3.1 we will briefly describe the Dynamic Adaptive Streaming over HTTP (DASH) standard. In Section 2.3.2 we then describe which metrics are usually identified as important for video streaming delivery. Finally, in Section 2.3.3, we describe briefly the adaptive bitrate streaming algorithms utilised throughout this thesis.

2.3.1 DASH

MPEG-DASH is an application level standard for video streaming. DASH is similar to Apple HLS, with some major differences [Clo19]:

- DASH is codec agnostic, while HLS only supports H.264 and H.265.
- HLS is only supported by Apple devices.
- Before 2016 HLS only supported a segment length of 10 seconds, while DASH allowed segments length variability. Today HLS default segments length is 6 seconds, but it can be adjusted.
- MPEG-DASH is an international standard, while HLS has been developed by Apple and has never been published, although it is widely supported.

At a high level, the major steps of DASH architecture are the encoding and the segmentation (discussed in section Section 2.2), the distribution and replication of the content into CDNs, and online delivery. To communicate to the video players all the information concerning a specific video content, DASH architecture relies on Media Presentation Description (MPD) files. Usually MPDs are delivered and standardised in an XML file, but custom implementations of DASH architecture can rely on different formats, like JSON.

In the standardised DASH XML format, video representation usually consists of one or more *periods*, portions of media with a given start time and a duration. The division of a specific video content into periods is used, for example, for advertisement insertion or changes in the codec configurations.

Within a single period, one or more adaptation sets are described. We could have, for example, a single adaptation set for the video content, and multiple adaptation sets for different languages.

The adaptation set is then divided into different representations. This different representation indicates the bitrate tracks at which a specific content, being it audio or video, is made available. For the video representation, some basic information (like resolution and average bitrate) are presented.

Each representation is then subdivided into media segments. Media segments can be represented in various ways, but in general they all point to the actual URL of the segment to be fetched. They can provide additional information that are useful to the adaptation logic, like segment duration and bitrate values.

When a video client starts the playback of a video, the video MPD is immediately downloaded and the video information is made available to the adaptive bitrate streaming algorithm, that will decide which representation of a specific segment to download depending on the network variations and player state.

One of the reference implementations of a DASH client is the dash.js player [DAS12]. This implementation is provided by the DASH Industry Forum (DASH-IF) with a BSD-3 license, a permissive license that allows extensions and modifications.

2.3.2 Quality of Experience

The main goal of adaptive streaming is to improve the users *so-called* perceived Quality of Experience (QoE). While the concept of QoE is context and user dependent, some common optimisation metrics can be identified and mathematically formulated:

- **Quality gain**: let V be a video stream session composed by a sequence of N video chunks. Let R_b be the set of available resolutions and bitrate pairs of each level of V. Let $q(\cdot) : R_b \rightarrow R_+$ be a non-decreasing function mapping each segment representation to a user perceived level of satisfaction (e.g.

`VMAF` score). Let $R^*_{b,i}$ be the downloaded quality level for the i-th segment. The quality gain of V can be defined as:

$$\frac{1}{N} \cdot \alpha \cdot \sum_{i=1}^{N} q(R^*_{b,i})$$

- **Quality switching penalty**: The non-smoothness penalty targets the magnitude of the quality variation between one chunk to another. This impairment can be formulated as follow:

$$\frac{1}{N-1} \cdot \beta \cdot \sum_{i=1}^{N-1} |q(R^*_{b,i+1}) - q(R^*_{b,i})$$

- **Rebuffer penality**: A rebuffer event happens whenever the video chunk that has to be played has not been downloaded yet. Let K be the number of rebuffer events during the video stream session V. Let $\Delta t_{stall,k}$ be the duration in time of the k-th playback stall. The rebuffer penalty is then defined as:

$$\mu \cdot \sum_{k=1}^{K} \Delta t_{stall,k}$$

- **Startup delay penalty**: Let T_s be the time needed by the video player to start the playback of the video stream session. The startup delay penalty can be then formulated as:

$$\gamma \cdot T_s$$

A common weighted linear formulation of the user QoE [Mil+15] is then defined as:

$$QoE^N = \frac{1}{N} \cdot \alpha \cdot \sum_{i=1}^{N} q(R^*_{b,i}) - \frac{1}{N-1} \cdot \beta \cdot \sum_{i=1}^{N-1} |q(R^*_{b,i+1})$$

$$-q(R^*_{b,i})| - \mu \cdot \sum_{k=1}^{K} \Delta t_{stall,k} - \gamma \cdot T_s]$$

The weighting parameters α, β, μ and γ define the importance of a component with respect of the other. This QoE formulation does not depend on users study, and for such reason it cannot be used as a subjective users satisfaction assessment. Nevertheless, it usually has a twofold goal:

- As the metrics previously listed are usually in tradeoff, it provides a quantitative assessment on how a specific adaptation logic balances the different optimisation goals.
- During optimisation time, by carefully tuning the weighting parameters, it can be used as the objective function in order to maximise (or minimise) certain optimisation metrics.

In this work, this formulation will be extensively used, both for evaluation and for optimisation. The different weighting parameters will be set depending on the specific problem.

2.3.3 Adaptive bitrate algorithms

Adaptive bitrate algorithms (ABR) are an integral and fundamental part of DASH architecture. The ABR main task is to decide which quality to fetch the next segment at depending on the client's fluctuating network capacity and player state. The goal of such adaptation is

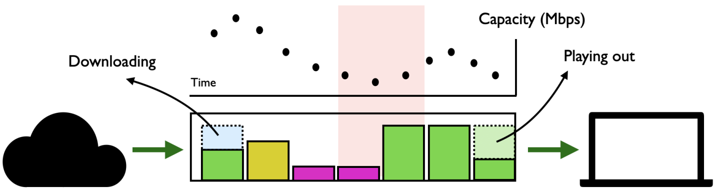


Figure 2.5: The main concept behind rate adaptation algorithm is to download the best quality possible depending on varying network conditions. The red shaded area of the pictures highlight a very simple adaptation approach: when the capacity of the network decreases, a lower quality level is picked.

to maximise users' perceived QoE. A simple diagram of ABR main concept is depicted in Fig 2.5.

Different ABRs can use different parameters to estimate the best choice in terms of quality to download. Some of them are designed to base their decision solely on network bandwidth, i.e. they estimate the current network capacity based on the download history and pick the next chunk quality accordingly. Other approaches can also take into account the player buffer behaviour, i.e. the number of seconds of the video that have been downloaded but not yet played. Other parameters that the ABR might account for are, for example, the future segments' properties (i.e. their perceptual quality value and their bitrate). In the following subsections we will describe a small set of ABRs that are used in this work.

2.3.3.1 Rate Based approaches

Rather than a specific ABR, rate based approaches (RB) are a class of ABR implementations. The decision of the next quality to donwload is taken depending only on the estimate of the network bandwidth. How to estimate the network bandwidth given the past download samples is fundamental design parameter. A common and straightforward way to estimate the bandwidth is a weighted mean over a certain number of samples, where most recent samples are given more importance. The implementation of a rate based approach that is used in this dissertation is the one described by Bitmovin [Aya+18]. In this implementation, given a certain bandwidth estimate, the adaptation algorithm chooses the track with the highest bitrate that is not larger than the estimate. During the startup phase, a default (low) quality is downloaded for the first 10 seconds of playback. This conservative startup design choice is implemented in order to safely build up the buffer and to collect bandwidth samples.

2.3.3.2 Buffer based approaches

Buffer based approaches (BB) base their decision mainly on the player buffer occupancy. Buffer based, as rate based, can be identified as a class of ABR algorithms. The buffered amount of video, i.e. the amount of video that has been downloaded but not yet played, can indicate how much safety is present before incurring in a rebuffering event.

An example of buffer based approach is the one described in [Hua+14]. In this implementation, the buffer size b is mapped to a track level, and two parameters, the reservoir r and the cushion c indicate the boundaries of this mapping. Specifically, if $b < r$, the lowest track is used, and if $b > r + c$ the highest track is used. The intermediate tracks are then mapped linearly in the range [r, r+c] (Fig 2.6).

To prevent frequent track changes, the discrete boundaries of the tracks act as thresholds, so that the track is only changed if it passes the threshold of the next higher or next lower bitrate level, introducing an hysteresis to the track selection.

Another example of buffer based algorithm is the Buffer Occupancy based Lyapunov Algorithm (BOLA)[SUS16]. However, BOLA takes a more sophisticated approach than the previous one, using Lyapunov optimization techniques to achieve a near-optimal QoE.

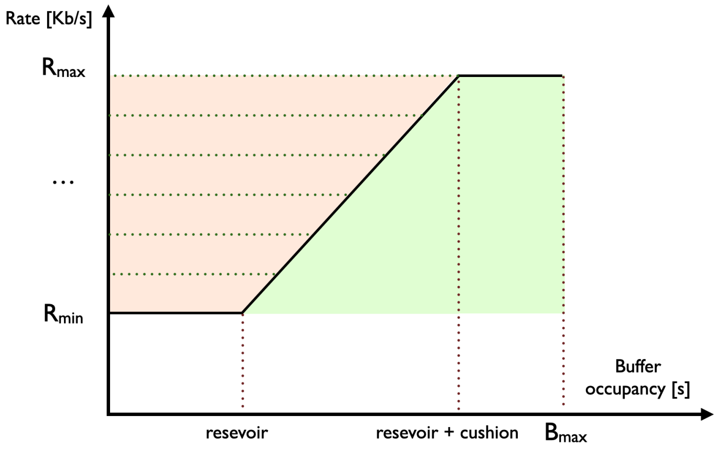


Figure 2.6: Graphical representation of a buffer based approach described in [Hua+14]. The red shaded area of the 2D plane identifies the risky buffer health-rate mapping, in which rebuffering events are more likely to happen. Conversely, the green shaded area identifies the safe area of the mapping, in which rebuffering events are unlikely to happen, at the cost of being too conservative with the chosen quality.

2.3.3.3 MPC

Model predictive control (MPC), described in [Mil+15], solves the optimisation problem with a control-theoretic approach. It uses the bandwidth estimate, current buffer size, and features of upcoming segments to plan a sequence of requests based on the expected reward. The reward is usually formulated in the linear form described in Section 2.3.2. Specifically, all the possible sequence of downloads are simulated within a certain horizon, and the best performing first option is picked. A version of it, Robust MPC, makes use of a more conservative bandwidth estimate in order to perform safer choices, and it is the version of MPC widely used in this dissertation.

2.3.3.4 Pensieve

Pensieve [MNA17b] uses the A3C reinforcement learning model in order to learn the best performing choices in terms of track selection. It does so by capturing the environment's state in terms of buffer occupancy, past bandwidth samples, and future video chunks properties. The reward is usually expressed in the form described in Section 2.3.2.

In this Chapter we introduced to the reader the basic concepts behind VOD streaming system. Specifically, in Section 2.2, we detailed the main variables involved into video preparation, while in Section 2.3 we described DASH architecture and the concept of ABR optimisation algorithms. In Chapter 3 and Chapter 4 we will discuss in detail how network measurements and active manipulation can be effectively utilised to better understand and reconstruct ABR behaviours. Finally, in Chapter 5, we will show how to optimise the offline part of VOD streaming systems depending on the expected flow of the online one.

3 Understanding video streaming algorithms

In this Chapter, we evaluate and compare the behavior of 10 different large video platforms' ABRs. In particular, by manipulating the network bandwidth and carefully analysing the players' logs, we focus on understanding which optimisation goals these providers are targeting.

3.1 Introduction

Video streaming now forms more than 50% of downstream Internet traffic [San22]. Thus, methods of delivering video streams that provide the best user experience despite variability in network conditions are an area of great industry relevance and academic interest. As explained in Section 2.3, the problem is to provide a client with the highest possible video quality, while minimizing pauses in the video stream. There are other factors to consider, of course, such as limiting the number of distracting resolution changes. These considerations are typically rolled into one quality-of-experience score. Streaming services then use adaptive bitrate algorithms, which attempt to maximize QoE by dynamically deciding at what resolution to fetch video segments, as network conditions fluctuate.

While high-quality academic work proposing novel ABR is plentiful, the literature is much more limited (Section 3.2) in its analysis of widely deployed ABRs, their target QoE metrics, and how they compare to recent research proposals. This chapter is designed to precisely address this gap. Understanding how video platforms serving content to large user populations operate their ABR is crucial to framing future research on this important topic. For instance, we would like to know if there is a consensus across video platforms on how ABR should behave, or whether different target populations, content niches, and metrics of interest lead to substantially different ABR behavior. We would also like to understand whether ABR research is optimising for the same metrics as deployed platforms, which are presumably tuned based on operator experience with real users and their measured engagement.

Towards addressing these questions, we present a study of ABR behavior across 10 video streaming platforms (Table 3.1) chosen for coverage across their diverse target populations: some of the largest ones in terms of overall market share, some regional ones, and some specialized to particular applications like game streaming (not live, archived). Our methodology is simple: we throttle download bandwidth at the client in a time-variant fashion based on throughput traces used in ABR research, and monitor the behavior of streams from different streaming platforms by analyzing jointly their browser-generated HTTP Archive (HAR) files and properties exposed by the video players themselves. By doing so, we show how network measurements can be use to *understand* video streaming, which is indeed the first goal of this dissertation. For robust measurements, we collect data for several videos on each platform, with our analysis based on 6 days of continuous online streaming. Our main findings are as follows:

1. Deployed ABRs exhibit a wide spectrum of behaviors in terms of how much buffer they seek to maintain in their stable state, how closely they try to match changing bandwidth vs. operating more smoothly, how they approach stable behavior after stream initialization, and how well they use available network bandwidth. There is thus not a consensus approach in wide deployment.

2. Several deployed ABRs perform better on a QoE metric based on visual perception rather than just video bitrate. This lends support to the goals of recent work [Qin+18], indicating that at least some of the industry is already optimizing towards such metrics rather than the bitrate-focused formulations in most prior ABR research.

3. Most deployed ABRs eschew fast changes in response to bandwidth variations, and thus exhibiting stable behavior. In contrast, research ABRs follow bandwidth changes more closely. It is unclear whether this is due to (a) a mismatch in target metrics used in research and industrial ABR; or (b) industrial ABR being sub-optimal.

4. Several deployed ABRs leave substantial available bandwidth unused. For instance YouTube uses less than 60% of the network's available bandwidth on average across our test traces. Similar to the above, it is unclear whether this is due to ABR sub-optimality, or a conscious effort to decrease bandwidth costs.

3.2 Related Work

There is a flurry of academic ABR proposals [Akh+18; Sun+16; SUS16; Mil+15; MNA17b; WRZ16; Qin+17; JSZ14; De +13; Li+14;

SSS18; Qin+18], but only limited study of the large number of deployed video streaming platforms catering to varied video types and audiences.

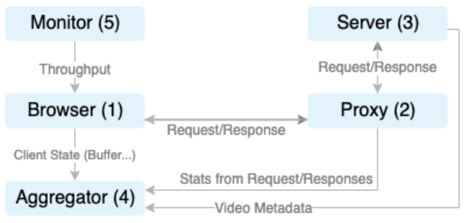
YouTube itself is relatively well studied, with several analyses of various aspects of its behavior [Mon+17; Año+18; Wam+16], including video encoding, startup behavior, bandwidth variations at fixed quality, a test similar to our reactivity analysis, variation of segment lengths, and redownloads to replace already fetched segments. There is also an end-end analysis of Yahoo's video streaming platform using data from the provider [Gha+16].

Several comparisons and analysis of academic ABR algorithms [Yan+20; TMR16; Sto+17] have also been published, including within each of the several new proposals mentioned above. In particular, [Sto+17] compares three reference ABR implementations, showing that the configuration of various parameters has a substantial impact on their performance.

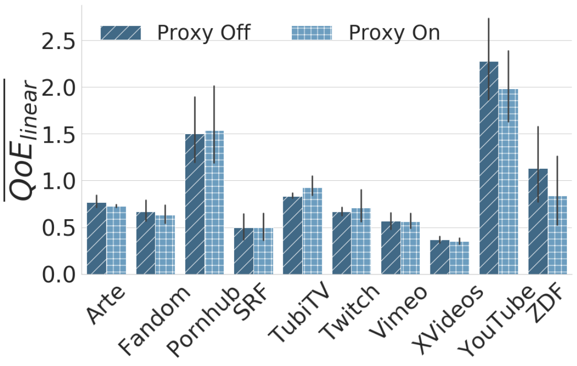
Facebook published [Mao+19] their test of Pensieve [MNA17b] in their video platform, reporting small improvements (average video quality improvement of 1.6% and average reduction of 0.4% in rebuffers) compared to their deployed approach.

However, a broader comparative study that examines a large number of diverse, popular streaming platforms has thus far been missing. Note also that unlike ABR comparisons in academic work and head-to-head comparisons of methods in Facebook's study, QoE comparisons across platforms are not necessarily meaningful, given the differences in their content encoding, content type, and audiences. Thus, in contrast to prior work, we define a set of metrics that broadly characterize ABR behavior and compare the observed behavior of a large, diverse set of streaming providers on these metrics. Where relevant, we also contrast the behavior of these deployed ABRs with research proposals. To the best of our knowledge the work [LGS20], upon which this Chapter is based, is the only work to compare a large set of deployed ABRs and discuss how their behavior differs from academic work in this direction.

3.3 Methodology



(a) Experimental setup



(b) Proxy impact

Figure 3.1: (a) Player behaviour is influenced through bandwidth throttling, and is recorded from multiple sources. (b) The proxy has little impact on player behavior as measured in terms of average linear QoE ($\overline{QoE_{linear}}$); the whiskers are the 95% confidence interval.

To understand a target platform's ABR, we must collect traces of its behavior, including the video player's state (in terms of selected video quality and buffer occupancy) across controlled network conditions and different videos.

3.3.1 Experimental setup

Fig. 3.1a shows our architecture for collecting traces about player behaviour. Our Python3 implementation (available at [GL]) uses the Selenium browser automation framework [Hug] to interact with online services. For academic `ABR` algorithms, trace collection is simpler, and uses offline simulation, as suggested in [MNA17b].

While playing a video, we throttle the throughput at the client **(1)** using `tc` (Traffic control, a Linux tool).[1] The state of the client browser (*e.g.,* current buffer occupancy) is captured by the Monitor (**5**) every *a* seconds. All requests sent from the client (**1**) to the server (**3**) are logged by a local proxy (**2**). Beyond the final browser state, the proxy allows us to log video player activity such as chunks that are requested but not played. We also obtain metadata about the video from the server (*e.g.,* at what bitrate each video quality is encoded). Metadata is obtained through offline analysis by downloading the video at all different qualities. All information gathered from the three sources — the proxy, the browser and the server — is aggregated **(4)**.

Certain players replace chunks previously downloaded at low quality with high quality ones ("redownloading") in case there is later more bandwidth and no immediate rebuffer risk. Using the proxy's view of requests and responses and the video metadata, we can map every chunk downloaded to a play-range within the video, and use this mapping to identify which chunks / how many bytes were redownloaded.

How do we add a platform to our measurements? Most video platforms (all except YouTube, for which we use [Ami]) use chunk-based streaming. To evaluate such platforms, we use developer tools

[1] At the bandwidth levels seen in our traces, bottlenecks are at our client — our university's connectivity to large services is otherwise high-bandwidth, consistently resulting in the highest-quality playback available on each service.

in Chrome to understand how the player obtains the download links for the chunks. Typically, a .m3u8 [PM15] file downloaded by the player contains the locations for all chunks at all qualities. This allows us to write code that fetches all chunks for the test videos at all qualities, such that we can use these videos in our offline simulation analysis of the academic `Robust MPC` approach (which implementation is described in section Section 2.3.3).[2] Having all chunks available also enables calculation of their visual perceived quality (`VMAF` [Li+16]). We also need to map each chunk to its bitrate level and time in the video stream, by understanding how video content is named in the platform (*e.g.,* through “itags” in YouTube).

For online experiments through the browser, we need to instrument the platform’s video player. We do this by automating the selection of the HTML5 video player element, and having our browser automation framework use this to start the video player and put it in full screen mode. We can then access the current buffer occupancy and current playback time using standard HTML5 attributes. We use a proxy to log the remaining statistics (*e.g.,* resolution played/fetched) because relying on the player alone would have required painstaking code injection specialized to each provider.

YouTube does not follow such chunked behavior (as past work has noted [Mon+17]). It can request arbitrary byte ranges of video from the server. We use an already available tool [Ami] to download the videos, and then learn the mapping from the byte ranges to play time from the downloaded videos.

3.3.2 The proxy’s impact on measurements

Some of our measurements (*e.g.,* redownloads) use an on-path proxy, so we verify that this does not have a meaningful impact by comparing metrics that can be evaluated without the proxy. For this, we use traces with constant bandwidth $b \in [0.5, 0.8, 1.2, 2.5]$

[2] To avoid the unintended use of our scripts for downloading copyright-protected content, we refrain from publishing code for this part of our pipeline.

Mbps, repeating each experiment 5 times for the same video. For our comparison, we calculate `QoE` using the linear function from `MPC` [Mil+15] with and without the proxy. For every video-network trace combination, we calculate the mean `QoE` and show the mean across these, together with its 95% confidence interval with whiskers in Fig. 3.1b.

As the results show, for most platforms the proxy has a minimal impact: across providers, the average difference in `QoE` with and without the proxy is 7%. For YouTube and ZDF, the differences are larger, but still within the confidence bounds: for these providers, there are large variations across experiments even without the proxy, indicating differing behaviour in very similar conditions in general.

3.3.3 Metrics of interest

Different video platforms serve very different types of content, and target different geographies with varied client connectivity characteristics. It is thus not particularly informative to compare metrics like bitrate-based `QoE` across platforms. For instance, given the different bitrate encodings for different types of content, bitrate-`QoE` is not comparable across platforms. We thus focus on comparisons in terms of the following behavioral and algorithm design aspects.

Initialization behavior: We quantify how much *wait time* a video platform typically incurs for streams to start playback, and how much *buffer* (in seconds of playback) it builds before starting. We use traces with a fixed bandwidth of 3 Mbps until player's HTML5 interactions are available, thus always downloading items like the player itself at a fixed bandwidth. This is done to avoid failure at startup: some platforms cause errors if network conditions are harsh from the beginning. After this, we throttle using only the high-bandwidth traces from the Oboe [Akh+18] data set, which have a mean throughput of 2.7 Mbps. We start timing from when the first chunk starts downloading (per the HAR files; the player HTML5 interactions may become available earlier or later).

Convergence: During startup, an ABR may have little information about the client's network conditions. How do different ABRs approach stable behavior starting from this lack of information? Stablility in this sense refers to fewer bitrate switches. Thus, to assess convergence characteristics, we quantify the bitrate changes (in Mbps per second) across playback, *i.e.,* a single switch from 3 Mbps to 4 Mbps bitrate over a total playback of 5-seconds amounts to 0.2 Mbps/sec on this metric. We chose not to compare the raw *number* of switches/sec — one switch at YouTube is very different from one switch at TubiTV, due to the differing discreteness of their bitrate ladders.

Risk-tolerance: ABRs can hedge against rebuffer events by building a larger buffer, thus insulating them from bandwidth drops. Thus, how much *buffer* (in seconds of video) an ABR builds during its stable operation is indicative of its risk tolerance.

Reactivity: ABRs must react to changes in network bandwidth. However, reacting too quickly to bandwidth changes can result in frequent switching of video quality, and cause unstable behavior when network capacity is highly variable. To quantify reactivity of an ABR, we use synthetic traces with just one bandwidth change after convergence, and measure the evolution of *bitrate difference* in the video playback after the change over time (with the number of following chunk downloads used as a proxy for time).

Bandwidth usage: ABR must necessarily make conservative decisions on video quality: future network bandwidth is uncertain, so fetching chunks at precisely the estimated network bandwidth would (a) not allow building up a playback buffer even if the estimate were accurate; and (b) cause rebuffers when bandwidth is overestimated. Thus, ABR can only use some fraction of the available bandwidth. We quantify this behavior in terms of the fraction of *bytes played to optimally downloadable*, with "optimally downloadable" reflecting the minimum of (*a posteriori* known) network capacity and the bytes needed for highest quality streaming.

For better bandwidth use and to improve QoE, some ABRs are known to redownload and replace already downloaded chunks in the buffer with higher quality chunks. We quantify this as the fraction of *bytes played to bytes downloaded*. Fractions <1 reflect some chunks not being played due to their replacement with higher quality chunks.

QoE goal: Academic ABR work has largely used a QoE metric that linearly combines a reward for high bitrate with penalties for rebuffers and quality switches [Mil+15; MNA17b]. This linear formulation has been presented in Section 2.3.2. More recent work has suggested formulations of QoE that reward perceptual video quality rather than just bitrate [Qin+18]. One such metric of perceptual quality, VMAF [Li+16], combines several traditional indicators of video quality. While it is difficult, if not impossible, to determine what precise metric each platform's ABR optimizes for, we can evaluate coarsely whether this optimization is geared towards bitrate or VMAF-like metrics by examining what video chunks an ABR tries to fetch at high quality: do chunks with higher VMAF get fetched at a higher quality level? To assess this, we sort chunks by VMAF (computed using [Li+16]) and quantify for the top n% of chunks, their (average) playback quality level compared to the (average) quality level of all chunks, $\overline{Q_{top-n\%}} - \overline{Q_{all}}$. A large difference implies a preference for high-VMAF chunks.

3.3.4 Measurement coverage

We evaluate multiple videos on each of 10 platforms across a large set of network traces.

Target platforms: Table 3.1 lists the platforms we analyze (with their Alexa popularity rank, as of January 2020). While by no means exhaustive, these were chosen to cover a range of content types and a few different geographies. Note that Netflix, Amazon Prime Video, and Hulu were excluded because their terms of service prohibit automated experiments or/and reverse-engineering [Net; Hul; Ama]. For Twitch, which offers both live streams and video-on-demand of archived live streams, we only study the latter, as live streaming is a substantially different problem, and a poor fit with the rest of our chosen platforms.

Provider	Description	Alexa rank	# Resolutions
Arte	Cultural	270, France	4.0 ± 0.0
Fandom	Pop-culture	91, Global	5.0 ± 0.0
SRF	Public Service	45, Switzerland	5.7 ± 0.48
TubiTV	Movies/Series	1330, USA	3.0 ± 0.0
Twitch	Gaming	39, Global	5.9 ± 0.32
Vimeo	Artistic content	188, Global	4.2 ± 0.92
YouTube	Broad coverage	2, Global	6.5 ± 1.08
ZDF	Public Service	47, Germany	5.3 ± 0.48
Pornhub	Pornographic	46, Global	4.0 ± 0.0
XVideos	Pornographic	67, Global	4.4 ± 0.52

Table 3.1: We test a diverse set of large video platforms.

Different platforms encode content at varied resolutions and number of resolutions, ranging from just 3 quality levels for TubiTV to 6.5 on YouTube (on average across our test videos; YouTube has different numbers of resolutions on different videos.)

When comparing the behavior of deployed ABRs with academic ones, we test the latter in the offline environment made available by the Pensieve authors [MNA17b]. For each tested video on each platform, we pre-download all its chunks at all available qualities. We then simulate playback using the same network traces up until the same point offline for academic ABRs as we do for the deployed ones. We primarily rely on `Robust MPC` [Mil+15] (referred to throughout as `MPC`) as a stand-in for a recent, high-quality academic ABR approach. While even newer proposals are available, they either use data-dependent learning techniques [MNA17b; Akh+18] that are unnecessary for our purpose of gaining intuition, or do not have available, easy-to-use code.

Videos: The type of content can have substantial bearing on streaming performance, *e.g.,* videos with highly variable encoding can be challenging for ABR. We thus used a set of 10 videos on each platform. Where a popularity measure was available, we used the most popular videos; otherwise, we handpicked a sample of different types of videos. Videos from each platform are encoded in broadly similar bitrate ranges, with most differences lying at higher qualities, *e.g.*, some content being available in 4K.

It would, of course, be attractive to upload the same video content to several platforms (at least ones that host user-generated content) to remove the impact of videos in the cross-platform comparisons. However, different platforms use their own encoding pipelines, making it unclear whether this approach has much advantage over ours, using just popular videos across platforms.

Network traces: Our experiments use synthetic and real-world traces from 3 datasets in past work [Akh+18; Rii+13a; Fed]. Unfortunately, a full cross-product of platform-video-trace would be prohibitively expensive — the FCC traces [Fed] alone would require 4 years of streaming time. To sidestep this, we rank traces by their throughput variability and pick traces with the highest and lowest variability together with some randomly sampled ones.

Our final network trace collection consists of the 5 least stable, 5 most stable, and 5 random traces from the Belgium trace collection [Hoo+16], and 10 in each of those categories from the Norway [Rii+13a], the Oboe [Akh+18] and the FCC datasets[3]. We also use 15 constant bandwidth traces covering the range from 0.3 to 15 Mbps uniformly. Lastly we add 10 step traces: after 60 seconds of streaming we suddenly increase/drop the bandwidth from/to 1 Mbps to/from 5 values covering the space from 1.5 to 10 Mbps uniformly.

In total, we use 130 traces with throughput (average over time for each trace) ranging from 0.09 to 41.43 Mbps, with an average of 6.13 Mbps across traces. Note that we make no claim of our set of traces being representative; rather our goal is to test a *variety* of traces to obtain insight into various ABR behaviors. If a trace does not cover the whole experiment we loop over it.

For quantifying reactivity, we only use the synthetic traces mentioned above, with a single upward step change in bandwidth. For

[3]Specifically, the stable collection from September 2017 [Fed].

quantifying startup delay, we use traces with a bandwidth of around 3 Mbps as noted in Section 3.3.3.

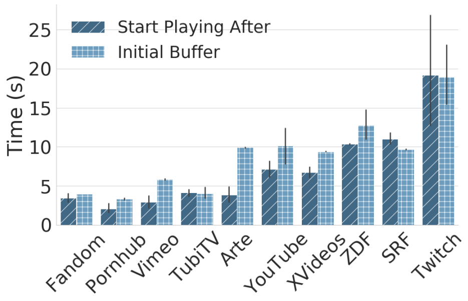
Ethics: We are careful to not generate excessive traffic or large bursts to any platform, measuring at any time, only one stream per service, typically at a low throttled rate.

3.4 Measurement results

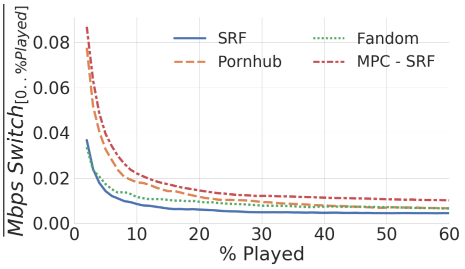
Overall, we see diverse behavior on each tested metric across platforms. We attempt to include results across all platforms where possible, but for certain plots, for sake of clarity, we choose a subset of platforms that exhibits a range of interesting behaviors.

Initialization behavior, Fig. 3.2a: We find that most platforms' `ABR` simply waits for one chunk download to finish before beginning playback. This is reflected in the buffer occupancy at playback. Some players like ZDF and SRF use a larger chunk size (10 seconds), which is why they pre-load more seconds of buffer.

As one might expect, building a larger buffer before playback starts generally incurs a higher start time. Twitch stands out in this regard, as it downloads nearly 20 seconds of buffer before start. Some players, whilst downloading the same number of buffer seconds as others, do so at much higher resolution – *e.g.,* SRF downloads its first 10 seconds with 6× as many pixels as Arte. This is reflected in the disparity between their start times, despite both populating the buffer with 10 seconds of playback. More broadly, all such "discrepancies" are difficult to explain because startup is hard to untangle from other network activity, *e.g.,* some players already start downloading video chunks while the player itself is still downloading, thus complicating our notion of timing. (We start timing from the point the first chunk starts downloading. For most platforms, this provides a leveling standard that excludes variation from other downloads on their Web interface. It also helps reduce latency impacts that are mainly infrastructure driven, as well as effects of our browser automation framework.)



(a) Initialization behavior



(b) Convergence

Figure 3.2: (a) Initialization: most providers start playback after one chunk is downloaded. (b) Convergence is measured in terms of changes in bitrate switching, *i.e.,* the (absolute) sum of bitrate differentials across all switches from the start, divided by the thus-far playback duration. As expected, switching is more frequent during startup, but the degree of switching varies across providers both in startup and later.

Convergence, Fig. 3.2b: As expected, during startup and early playback, every player attempts to find a stable streaming state. This results in many track switches followed by much smoother behavior with more limited switching. Nevertheless, there are large differences across players, *e.g.,* Pornhub switches more than twice as much as Fandom and SRF in the beginning. In stable state, Fandom switches substantially more than SRF. We also evaluated the academic (Robust) `MPC` algorithm [Mil+15] on the same network traces and over the SRF videos. The `MPC` algorithm would use more than twice as much switching both in startup and later, compared to SRF's deployed `ABR`. Consequently, SRF scores lower than `MPC` on the default linear `QoE` model used in `MPC`. However, this does not necessarily imply that SRF's design is sub-optimal; it could also be optimizing for a different metric that values stability more.

For clarity, we only picked a few platforms as exemplars of behavior towards convergence instead of including all 10 tested platforms. The behavior is broadly similar with more switching early on, but the precise stabilization differs across platforms.

Risk-tolerance, Fig. 3.3: We observe widely different buffering behavior across the players we tested. Of course, every player uses early playback to download lower quality chunks and accumulate buffer, but some, like YouTube, settle towards as much as 80 seconds of buffer, while others like Fandom operate with a much smaller buffer of around 20 seconds. Testing `MPC`'s algorithm on the same traces across the YouTube videos reveals that it falls towards the lower end, stabilizing at 20 seconds of buffer.

Note that for approaches that allow redownloads (including YouTube), larger buffers are a reasonable choice: any chunks that were downloaded at low quality can later be replaced. This is likely to be a more robust strategy in the face of high bandwidth variability. However, for approaches that do not use redownloads, a larger buffer implies that all its content must be played out at whatever quality it was downloaded at, thus limiting the possibilities to benefit from opportunistic behavior if bandwidth later improves. Thus operating with a smaller buffer of higher-quality chunks may be preferable to filling it with lower-quality chunks. In the absence of redownloads,

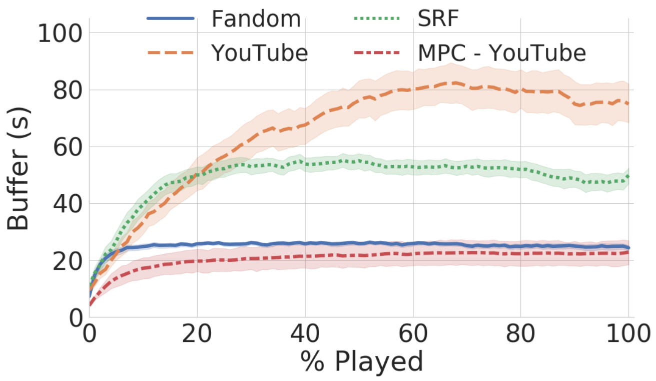


Figure 3.3: Risk-tolerance: YouTube operates with nearly 4× the buffer for Fandom. The shaded regions show the 95% confidence interval around the mean.

there is thus a tradeoff: a larger buffer provides greater insurance against bandwidth drops, but reduces playback quality. At the same time, redownloads are themselves a compromise: *if* better bitrate decisions could be made to begin with, redownloads amount to inefficient bandwidth use.

Reactivity, Fig. 3.4: We find that most deployed ABRs are cautious in reacting to bandwidth changes. This is best illustrated through comparisons between deployed and academic ABRs. Fig. 3.4(right) shows such a comparison between TubiTV and MPC evaluated on the same traces and videos. After the bandwidth increases (at x-axis=0 in the plot), TubiTV waits for tens of chunk downloads before it substantially ramps up bitrate. In contrast, MPC starts switching to higher bitrates within a few chunk downloads. (The large variations around the average arise from the varied sizes of the step-increases in the used network traces and variations in the tested videos.)

While we have not yet evaluated a large number of mobile ABR

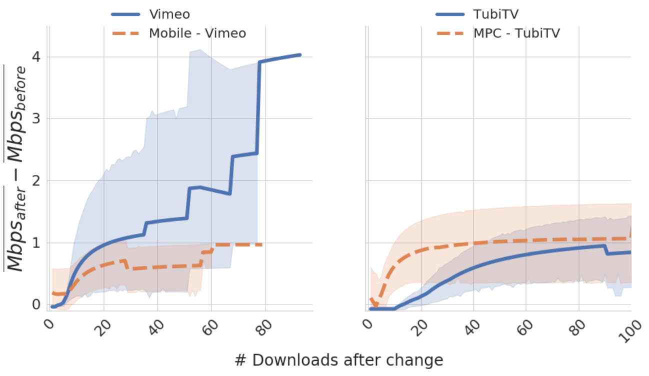


Figure 3.4: We measure reactivity in terms of bitrate evolution after a bandwidth increase, *i.e.,* difference in average playback bitrate after and before the bandwidth change over time (in terms of chunk downloads). The plots show the reactivity differences between: (left) mobile and desktop versions of Vimeo; and (right) TubiTV and `MPC`.

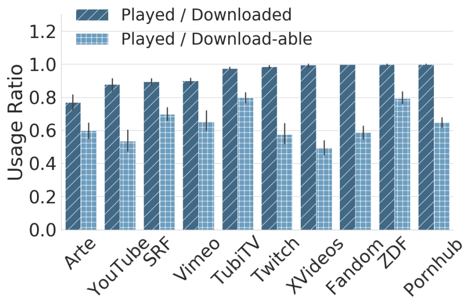
implementations (see Section 3.5), we were able to experiment with Vimeo's mobile and desktop versions, shown in Fig. 3.4(left). They exhibit similar ramp-up behavior in terms of how many downloads it takes before Vimeo reacts, but show very different degrees of bitrate change. The desktop version increases bitrate in several steps after the bandwidth increase, while the mobile one settles at a modest increase. This is along expected lines, as the mobile player, targeting the smaller screen, often does not use the higher-quality content at all.

A comparison between TubiTV and Vimeo (desktop) across the two plots is also interesting: Vimeo ramps up faster than TubiTV. (`MPC` ramps us even faster on the Vimeo videos.) One potential reason is the difference in encoding — TubiTV serves each video in only 3 resolutions, compared to Vimeo's 4-5. This implies that over the same network traces, TubiTV must necessarily see a larger change in bandwidth to be able to jump from one bitrate to the next, given its larger differential in bitrate levels.

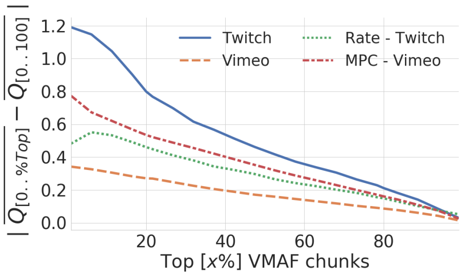
Bandwidth usage, Fig. 3.5a: Different platforms use bandwidth very differently. Arte discards a surprisingly large 23% of its downloaded bytes in its efforts to replace already downloaded low-quality chunks with high-quality ones. Some platforms, including YouTube, SRF, and Vimeo, show milder redownload behavior, while several others, including XVideos, Fanrom, Pornhub, and ZDF, do not use redownloads at all.

ZDF and TubiTV are able to use 80% of the network's available bytes for fetching (actually played) video chunks, while all others use the network much less effectively. While the uncertainty in future bandwidth and the desire to maintain stable streaming without many quality switches *necessitates* some bandwidth inefficiencies, we were surprised by how large these inefficiencies are. In particular, XVideos, YouTube, Twitch, and Fandom all use less than 60% of the network's available capacity on average across our trace-video pairs[4]. This low usage is particularly surprising for YouTube, which uses several strategies — variable chunk lengths (as opposed to fixed-size chunks in other providers), larger number of available video resolutions, and redownloads — that allow finer-grained decision making, and thus should support more effective bandwidth use. Given these advanced features in their ABR design, it is more likely that their optimization goals differ from academic ABR work than their algorithm simply being poorly designed. While we cannot concretely ascertain their optimization objectives, one could speculate that given the large global demands YouTube faces while operating (largely) as a free, ad-based service, a profit maximizing strategy may comprise providing good-enough QoE with a limited expense on downstream bandwidth.

[4]Note that these inefficiencies cannot be blamed on transport / TCP alone, as on the same traces, other players are able to use 80% of the available capacity. We also carefully account for non-video data to ensure we are not simply ignoring non-chunk data in these calculations. For instance, audio data is separately delivered for Vimeo and YouTube, but is accounted for appropriately in our bandwidth use analysis.



(a) Bandwidth usage



(b) QoE goal

Figure 3.5: (a) Bandwidth usage: many players use surprisingly little of the available network bandwidth (Played / Download-able) despite the potential to improve quality with more bandwidth, *e.g.,* XVideos uses only 50% of it; and some players, like Arte, spend a large fraction of their used bandwidth on redownloads. (b) `QoE` goal: we measure how much a player prefers high-`VMAF` chunks by quantifying the average quality-level difference between all chunks and only the top-x% of chunks by `VMAF` (*i.e.,* $\overline{Q_{[0...\%Top]}}$). Some players, like Twitch, show a large preference for high-`VMAF` chunks.

QoE goal, Fig. 3.5b: We find that some providers fetch high-VMAF chunks at higher quality than the average chunk. In particular, Twitch fetches the chunks in the top 20th percentile by VMAF at a mean quality level 0.79 higher than an average chunk. If instead of Twitch's ABR, we used a VMAF-unaware, simple, rate-based ABR[5] that uses an estimate of throughput to decide on video quality, this difference in quality level between high-VMAF and the average chunk would reduce to 0.46.

Note that given the correlation between higher quality and higher VMAF, high-VMAF chunks are more likely to be fetched at high quality; what is interesting is the degree to which different players prefer them. Vimeo, for instance, shows a much smaller difference of 0.27 between the quality level of chunks in the top 20th percentile and an average chunk. If MPC's ABR were used to fetch chunks from Vimeo, this difference increases to 0.534, because MPC is willing to make more quality switches than Vimeo.

Our results thus indicate diversity in optimization objectives in terms of bandwidth use and QoE targets across deployed video platforms. It is at least plausible that academic ABRs produce different behavior over the same traces not because they are much more efficient, but rather the optimization considerations are different. While algorithms like MPC are flexible enough to be used for a variety of optimization objectives, it is unclear how performance would compare across a suitably modified MPC (or other state-of-the-art ABR) when evaluated on operator objectives.

3.5 Limitations and future work

Our first broad examination of a diverse set of widely deployed ABRs reveals several interesting insights about their behavior, but also

[5]This ABR estimates throughput, T, as the mean of the last 5 throughput measurements. For its next download, it then picks the highest quality level with a bitrate $\leq T$. It thus downloads the largest chunk for which the estimated download time does not exceed the playback time.

raises several questions we have not yet addressed:

1. Does ABR behavior for the same platform vary by geography and client network? Such customization is plausible — there are likely large differences in network characteristics that a provider could use in heuristics, especially for startup behavior, where little else may be known about the client's network bandwidth and its stability. However, addressing this question would require running bandwidth-expensive experiments from a large set of globally distributed vantage points.
2. How big are the differences between mobile and desktop versions of ABR across platforms? Unfortunately, while the browser provides several universal abstractions through which to perform monitoring on the desktop, most platforms use their own mobile apps, greatly increasing the per-platform effort for analysis.
3. If we assume that the largest providers like YouTube and Twitch are optimizing ABR well, based on their experience with large populations of users, can we infer what their optimization objective is? While there are hints in our work that these providers are not necessarily optimizing for the same objective as academic ABR, we are not yet able to make more concrete assertions of this type.
4. Does latency have a substantial impact on ABR? ABR is largely a bandwidth-dependent application, but startup behavior could potentially be tied to latency as well. We have thus far not evaluated latency-dependence.

3.6 Conclusion

In this Chapter, we conducted a broad comparison of adaptive bitrate video streaming algorithms deployed in the wild across 10 large video platforms offering varied content targeted at different audiences.

Specifically, we analysed different providers' ABRs behavior, identifying various key metrics, such as stability and reactivity to bandwidth changes.

In the following Chapter we will take a step forward in the analysis of propertary video streaming algorithms. While this Chapter focused on a quantitative analysis of ABRs' behaviours, Chapter 4 will focus on the reconstruction of such algorithms in an human-interpretable way.

4 Reconstructing video streaming algorithms

In this Chapter, we take a further step in the analysis of propertary video streaming algorithms. In particular, we produce human-interpretable reconstructions of the 10 video streaming platforms's ABRs analysed in Chapter 3.

4.1 Introduction

The problem of maximizing QoE in video streaming is clearly defined, intellectually interesting, and practically valuable. Thus, numerous ABR algorithms have been suggested in recent work to tackle it, *e.g.,* Oboe [Akh+18] and MPC [Yin+15]. As previously discussed, little is known about the proprietary algorithms actually deployed in widely used video streaming services such as YouTube, TwitchTV and

Netflix.[1] While Chapter 3 focused on the qualitative understanding of different deployed approaches of ABR, in this chapter, we attempt to address this gap by exploring whether it might be possible to learn such algorithms by controlled observation of video streams.

Our goal is to produce ABR controllers that: (a) mimic the observed behavior of ABR logic deployed in target online video services across a wide range of network conditions and videos; and (b) are open to easy manual inspection and understanding. Note that the latter precludes the direct use of blackbox machine learning techniques like neural networks.

We are motivated by three factors. First, this effort helps understand the risk of competitors copying painstakingly-engineered algorithmic work simply by interacting with popular, public-facing front-ends. Second, being able to reconstruct widely deployed algorithms would allow head-to-head comparisons between newly proposed research ABRs and industrial ABRs, something lacking in the literature thus far. Third, given that video is the majority of Internet traffic, this traffic being controlled by unknown proprietary algorithms implies that we do not understand the behavior of most Internet traffic. This makes it difficult to reason about how different services share the network, and interact with other control loops such as congestion control and traffic shaping.

The above use cases help sharpen the goals for our reconstruction effort. Simplifying our task is the fact that instead of exact algorithm recovery, we need functional equivalence of a reconstruction with its target algorithm over a large, varied set of inputs – Note that the same set of outcomes could be arrived at by two substantially different algorithms, making exact recovery of a particular algorithm impossible. However, our use cases also impose a difficult additional requirement: our reconstructions must be human-interpretable, allowing not only

[1] Researchers at Netflix published, in 2014, work on this problem [Hua+14], including tests on their commercial deployment. Per our conversations with them, their current deployment incorporates some features of this published work, but they are unwilling to share more details, including the differences from this published approach.

the mimicking of observed behavior, but also manual inspection and understanding. A competitor seeking to copy the ABR logic of an online service needs interpretability to be able to modify it as necessary for their use.[2] They would also like to ensure the robustness of the obtained logic, something that is difficult with blackbox learning — prior work has shown corner cases with undesirable behavior in blackbox learning methods applied to networking [Kaz+19]. Likewise, in terms of comparisons between industrial and academic ABRs, we would not only like to observe the performance differences empirically, but also understand where these differences stem from. Lastly, reasoning about interactions with other network control loops and competing services also requires having a richer understanding of the control logic under study than blackbox learning can provide.

Algorithmic reconstruction of this type is an ambitious goal, with the current tools available for general-purpose program synthesis still being fairly limited. However, there are two reasons for optimism if we can suitably narrow our scope: (a) the core of ABR algorithms involves a small set of inputs and has a limited decision space; and (b) it is easy to collect large amounts of curated data for analysis.

Our approach automatically generates concise, human-interpretable rule-sets that implement ABR by learning from an existing target ABR algorithm. These rule-sets map the client and network environment, video features, and state over the connection, to a video quality decision for the next video chunk. To obtain generalizable, succinct, and interpretable pseudocode in a reconstruction, we find that it is insufficient to directly use sophisticated techniques from imitation learning [BPS18; RGB11]. As we shall show later, such methods can either mimic the behavior of a target accurately with a large set of complex rules, or, when limited to a small set of rules, lose accuracy. Our approach sidesteps this tradeoff by embedding suitable domain knowledge in the learning mechanism: framing intuitive primitives familiar to domain experts, and making them available to

[2]Our work enables an understanding of whether this risk *exists*: “Can a competitor reconstruct an ABR in a meaningfully beneficial, robust way?” We leave the question of how this risk may be tackled to followup work.

the learning mechanism, results in rule-sets that are accurate, concise, and meaningful.

We use our approach to obtain concise reconstructions that can successfully mimic the decision-making of several target academic and industry ABR algorithms, achieving high agreement with their decisions and similar video QoE behavior. Of the 10 online streaming services we evaluate across, our reconstruction achieves behavior similar to its target for 7 services. In each case, we produce a concise decision-tree with 20 or fewer short rules, using primitives that are intuitive and easy to understand. We also explain the reasons for failure for the remaining 3 services.

In this chapter we make the following contributions:

- We describe an approach for deriving accurate *and* concise rule sets for ABR, using a corpus of decision outcomes over network traces and videos. Our approach handles the complex output space corresponding to diverse video encodings, as well as noise in the data.
- We apply our method to the reconstruction of algorithms deployed in 10 popular streaming services. For 7 services, we successfully achieve high agreement with their decisions and closely similar streaming behavior.
- The rule sets we obtain are concise, with 20 or fewer rules in each case. Our code also generates a loose natural language translation, which we used extensively in understanding problems and improving performance.
- We also expose a likely fundamental compromise necessary for interpretable and effective learning: the time-consuming encoding of domain knowledge.
- Our code and reconstructed ABRs are open-source [GLS20b].

Beyond the above results, our ambitious effort raises several exciting areas for future exploration, such as: (1) on the tradeoffs between the effort invested in embedding domain knowledge, and the quality of the inferred pseudocode; (2) to what extent such domain knowledge may itself be learnt from a corpus of hand-designed algorithms broadly from the networking domain; (3) applying our approach to other networking problems, like congestion control, and newer problems where we have more limited experience, such as multipath transport; (4) and how online service providers may obscure their logic against reconstruction, if so desired.

4.2 Related work

Numerous high-quality ABR proposals have appeared just within the past few years [Yeo+18; Akh+18; Qin+18; EA19; SSS18], but relatively little is known about widely deployed industrial ABR algorithms.

There is a large body of work on reconstructing unknown algorithms. One may approach this using code analysis, like Ayad et al.'s analysis of Javascript code for some online video services [Ibr+18]. However, some targets can be too large and obfuscated for such analysis – YouTube, for instance, comprises 80,000+ lines of obfuscated Javascript. We used JS NICE [SL18], the state-of-the-art in Javascript deobfuscation, but even coupled with a step-through debugger and with help from the authors of JS NICE, this provided little insight – ultimately, manually examining such a large piece of code with meaningless variable names to reconstruct its functionality seems futile. It also has the downside of potentially requiring substantial rework for even small changes in the target. Even more fundamentally, the code may not be available at the client at all, with decision-making residing on the server side.

Several prior efforts have used manual experimentation and analysis for dissecting the behavior of a variety of online services [Xu+13; ABD11; Ibr+18; DM10; Mon+17; LS10; LGS20]. For instance, Mondal et al. [Mon+17] used network traces to experimentally study the behavior under changing network conditions, and then manually draw coarse inferences, such as that YouTube's requested segment length varies with network conditions. An earlier effort on inferring Skype's adjustment of its sending rate [LS10], was based on the researchers making experimental observations, then manually hypothesizing a control law, and finally tuning its parameters to fit the data. Our own parallel measurement study [LGS20], described in Chapter 3, experimentally examined the behavior of several deployed ABR algorithms in terms of metrics like stability of playback and convergence time after bandwidth changes. In concurrent work, Xu et al. [XSM20] propose a method for inferring the quality of video chunks downloaded within encrypted streams, and apply it to experimentally study the streaming outcomes in response to different traffic throttling schemes. In contrast to all the above efforts, our goal here is to *automatically generate logic that mirrors a target ABR algorithm's behavior* by observing the target ABR's actions in response to variations in the environment and inputs.

There are also efforts in networking to inspect the internals of learning-based networked systems. This work is not directly applicable to our goal of reconstructing arbitrary ABRs, which are most likely non-ML, and more importantly, are not available to us. However, one could first train a blackbox-ML algorithm to mimic any reconstruction target, and then use such tools. Recent work on inspecting [DCK19] or verifying [Kaz+19] systems built using ML has examined Pensieve [MNA17b]. The authors frame hypotheses/questions about the system's behavior, and then evaluate them. However, this (a) requires knowing what hypotheses to examine, and (b) does not yield a reconstruction. Among efforts in this vein, the most closely related are the concurrent TranSys [Men+19b] and PiTree [Men+19a] studies. PiTree focuses on converting ABR algorithms to decision trees, and TranSys broadens this approach to NN-based strategies in networking. Both are networking applications of a broader paradigm in ML, which we discuss next.

Beyond networking efforts, *imitation learning* is a rich discipline in its own right. Most work in this direction uses (uninterpretable) neural networks [Wan+19; BK19; HE16], but recent work has also developed model-free approaches to approximate the learned neural network via, *e.g.,* a decision tree [RGB11; BPS18]. As we show later in Section 4.6.2, directly using this approach (like TranSys and PiTree) does not meet both of our accuracy and interpretability goals simultaneously, instead requiring the sacrifice of one or the other. While complex decision trees, with a large number of rules with many literals, can robustly imitate a target algorithm, they are difficult, if not impossible, for even domain experts to understand and work with. On the other hand, restricting the complexity of the generated trees results in a loss of imitation accuracy and robustness. While the expressiveness and compactness of these approaches can be improved by employing genetic algorithms to craft features for use therein [GZN07], this often leads to both overfitting, and complex, non-intuitive features.

Lastly, program synthesis is a rich and growing field. While we use one particular strategy for ABR reconstruction, there are other tools we plan to examine in future work. The most promising perhaps is recent work combining learning with code templates [Ver+18], where the core idea is to modify templates to minimize the distance from a target learning algorithm.An alternative "deductive synthesis" approach, as employed in Refazer [Rol+17], could also be fruitful.

To the best of our knowledge, our work [GLS20a], upon which this chapter is based, is the first to attempt an interpretable reconstruction of unknown deployed ABRs.

4.3 Data preparation

We extend a trace collection harness that we built for the measurement study described in Chapter 3, where we used manual analysis to comment on the behavior of deployed ABR algorithms across 10 streaming platforms [LGS20].

We launch a video stream on a target service, and according to an input network trace, shape the throughput at the client using Linux `tc`. We record the current client buffer occupancy, the video chunk qualities played out, video metadata, etc. The client buffer occupancy is directly measured through the instrumentation of the HTML5 player element. If the HTML5 player element were not available, we could instead use the captured HTTP chunk requests (locally at the client, making encryption irrelevant) to reconstruct the buffer occupancy — this strategy may be of use for future work exploring mobile ABR implementations. This alternative can be less accurate though, as "redownloading" (*e.g.,* to replace already downloaded low-quality chunks in the client player buffer by higher-quality ones) introduces an ambiguity into which chunk is actually played.

For each platform, by appropriate tailoring of HTTP requests, we also fetch all chunks for the test videos at all qualities, such that we can use these videos in an offline simulation, allowing the learned ABR to make choices different from those in our logs, as well as to enable us to study the behavior of academic ABRs. Ultimately, we obtain the following measurements:

- $\mathbb{C}_t$: segment size (Mb) downloaded for request t
- $\mathbb{R}_t$: segment bitrate (Mbps) for request t
- $\mathbb{V}_t$: segment VMAF[3] for request t
- $\mathbb{D}_t$: download time for request t
- $\mathbb{Q}_t$: quality level requested in request t
- $\mathbb{S}_t$: segment length (seconds) downloaded for request t
- $\mathbb{P}_t$: Percent of the video played at request t
- $\mathbb{B}_t$: buffer size (seconds) when requesting t
- $\mathbb{RB}_t$: rebuffer time (seconds) when requesting t
- $\mathbb{C}^i_{t+n}$: segment size of quality i for n^{th} chunk after t
- $\mathbb{R}^i_{t+n}$: segment bitrate of quality i for n^{th} chunk after t
- $\mathbb{V}^i_{t+n}$: segment VMAF of quality i for n^{th} chunk after t

4.4 Rule-set based inference

We shall first consider a motivating example for why rule-sets are a simple and potent representation for our type of target algorithms,

[3]VMAF is a video perceptual quality metric [Li+16], described in 2.2.3.

$$\begin{aligned}
\text{quality } 0 &\rightarrow \mathbb{T}_{N-1} \leq 4.99 \\
\text{quality } 1 &\rightarrow \mathbb{T}_{N-1} > 4.99 \;\&\; \mathbb{T}_{N-1} \leq 6.97 \\
\text{quality } 2 &\rightarrow \mathbb{T}_{N-1} > 6.97
\end{aligned}$$

Figure 4.1: Minimal rule-set for a reservoir-based algorithm, which uses only the last chunk's throughput estimate to pick a quality level.

and then present our recipe for constructing succinct rule-sets that capture the target algorithm's behavior.

4.4.1 Motivating example

Let us examine a simple throughput-based ABR algorithm, similar to that described in prior work [Hua+14]. It uses only the throughput estimate for the last video chunk fetched, $\mathbb{T}_{N-1}$, and two thresholds: *reservoir* and *cushion*. If $\mathbb{T}_{N-1}$ < *reservoir*, the lowest quality is served. If $\mathbb{T}_{N-1}$ > *reservoir* + *cushion*, the highest quality is served. For other values of $\mathbb{T}_{N-1}$, quality is linearly interpolated between these levels.

This algorithm, regardless of its specific instantiation with particular values of the thresholds, can be easily expressed as a set of rules. For a simple instantiation with only 3 quality levels, and both *reservoir* and *cushion* set to 4 Mbps, this rule-set is shown in Fig. 4.1.[4] The rule-set is inferred (which is why the rules contain imprecise values like 6.97) by the process we shall describe shortly.

We caution the reader against concluding from this small motivating example that only simple, stateless, "templates with parameters / thresholds" type of algorithms can be expressed in this way. Rule

[4]Readers may expect the rule-set in Fig. 4.1 to mean that $reservoir = 5$ and $cushion = 2$. The discrepancy stems from the discreteness of the interpolation: for some $\mathbb{T}_{N-1} > reservoir$ *i.e.,* $\mathbb{T}_{N-1} \in [4, 5]$, quality 0 will be chosen.

sets are capable of capturing complex stateful behavior, as long as primitives encoding this state are made available for their use.

4.4.2 Decision trees and rules

We first learn binary decision trees [L+84] that encode conditions for specific outputs, *e.g.,* the video quality levels requested. Further, in such a decision tree, each path from the root to a leaf can be naturally interpreted as a descriptive rule capturing the conditions for the outcome at the leaf to be reached.

Consider a single-threshold decision: "If throughput < 5 Mbps, pick low quality; otherwise, pick high quality.". This can be captured in a 3-node tree with the conditional at its root, and the two outcomes as leaves. In this case, the rule-lengths, *i.e.,* the path lengths from the root to the leaves, are 1; and so is the number of "splits" in the tree.

Fig. 4.2 shows a more complex target decision plane with two inputs (along the x and y dimensions), where there are still only two outcomes (labels), but the data samples that map to these labels are separated by more complex logic. If we constrain the decision tree that approximates this decision plane to use rules of only length one, we can use only one line separating the labels, as shown in the top-left smaller figure. Allowing more and more complex (longer) rules, allows a tighter representation of the target decision plane. Of course, using too many rules risks overfitting, especially under noisy data that is typical in networking. Fortunately, our goal to obtain concise rule sets aligns well with that of avoiding overfitting and preserving generalization.

Framing the output space: A key design consideration in using decision trees is the framing of the output decision space. Suppose we frame decision outcomes in terms of the video quality level that the client should fetch the next video chunk at. *If* all the videos being served were encoded with the same quality levels, both in terms of number and their bitrates, *e.g.,* 6 video qualities at bitrates of {200, 450, 750, 1200, 2350, 4300} Kbps, this would be easy: there are 6 *a priori* known outcomes that we can train decision trees for.

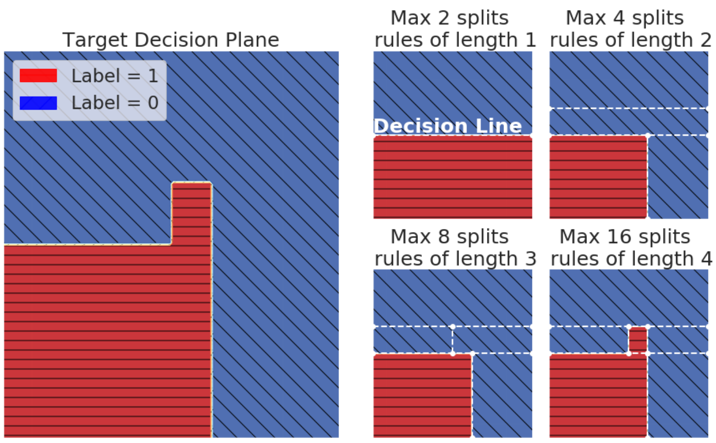


Figure 4.2: The big image is the target decision plane, with 2 labels. On the right are its approximations with decision trees of different rule-lengths, going from 1 to 4.

However, this is clearly overly restrictive: in practice, ABR algorithms must tackle a variety of videos encoded at different bitrates. The set of different bitrates at which a video is encoded in is referred to as its "bitrate ladder". Providers like Netflix even use per-video customization of bitrate ladders, varying the number and separation of bitrate levels [Aar+]. This diversity in the output space is a challenge for learning approaches: what should we present as the potential output decision space? It is noteworthy that Pensieve [MNA17b] does not fully tackle this challenge, instead restricting the video bitrate levels to a small pre-defined set.

To overcome this issue, we formulate the decision process in terms of video quality being upgraded or downgraded relative to the current video quality. With one decision, a maximum of n quality shifts are permitted in either direction, with n being a tunable parameter. Of course, this prevents us from capturing some algorithms, where larger quality changes (than n) may be enforced in a single step. However, this is atypical, as video streaming typically targets smooth output. Even if some algorithm involves such decision making, our approach only differs from such a target's decisions transiently. This small compromise allows us to generalize to arbitrarily diverse video bitrate ladders.

4.4.3 Feature engineering

Applying textbook decision-tree inference, with the above framing, one can already infer simple algorithms. However, as we shall see, appropriate customization based on domain expertise is crucial to obtain concise and generalizable rules.

Consider, for instance, a target algorithm that uses the mean throughput across the last 2 video chunks fetched. Naively learnt rules will then contain complex conditionals across both $\mathbb{T}_{N-1}$ and $\mathbb{T}_{N-2}$. Fig. 4.3 shows this for rules of increasing length, up to 20. The target decision plane uses the mean, $\frac{\mathbb{T}_{N-1}+\mathbb{T}_{N-2}}{2}$ to decide between three video qualities. Rules of length 2 and 5 yield poor accuracy, necessitating much longer (complex) rules.

Of course, if we knew that the building block for throughput estimation is the mean, we could simplify such rules substantially by expressing them in terms of the mean. Thus, we can consider adding common estimators, based on our domain knowledge, to the feature-set available to our learning pipeline. Then, instead of only learning across primitive literals (like each individual chunk's throughput), more compact representation across these non-primitive literals becomes possible. We thus explore three classes of such non-primitive features that are intuitive and likely commonplace in ABR, and even more broadly, other networking algorithms.

Throughput estimators: Clearly, having the most accurate estimate of network throughput is advantageous in deciding on video quality. As such, throughput estimators are potentially useful building blocks. We consider two types of estimators. The first type is only

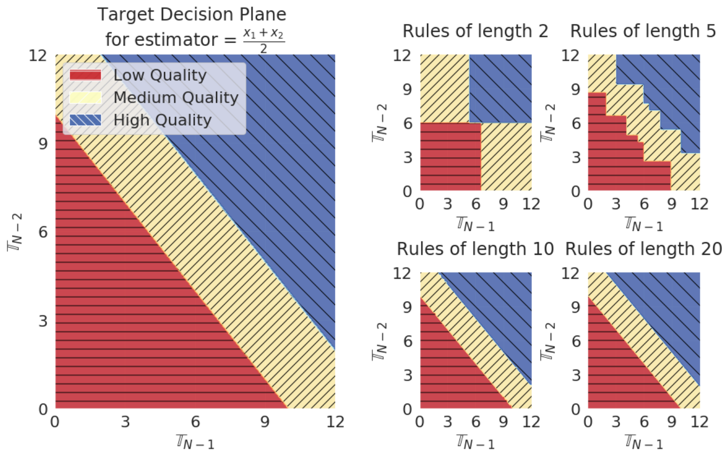


Figure 4.3: Here, the target decision plane (big, left) is governed by the mean of $\mathbb{T}_{N-1}$ and $\mathbb{T}_{N-2}$. The smaller figures show that we need long rules to approximate this if we are restricted to using individual literals ($\mathbb{T}_{N-1}$ and $\mathbb{T}_{N-2}$) in our rules.

parametrized by the number of past throughput measurements it aggregates using a mean, median, harmonic mean, etc., while the second type involves additional tunable parameters, such as the weight decrease, α, in an exponential weighted moving average (`EWMA`), which sets the relative weight of a new measurement compared to old measurements.

Encoding these estimators with a range of different parameter choices gives us a large set of features ranging from nearly stateless (*e.g.,* using only the last chunk's throughput) to those with long-term state (*e.g.,* a moving average). In addition to throughput, we also construct features capturing the *variation* across recent throughput measurements as it characterizes the stability of the network.

Comparisons: Decisions often depend on not just thresholding of

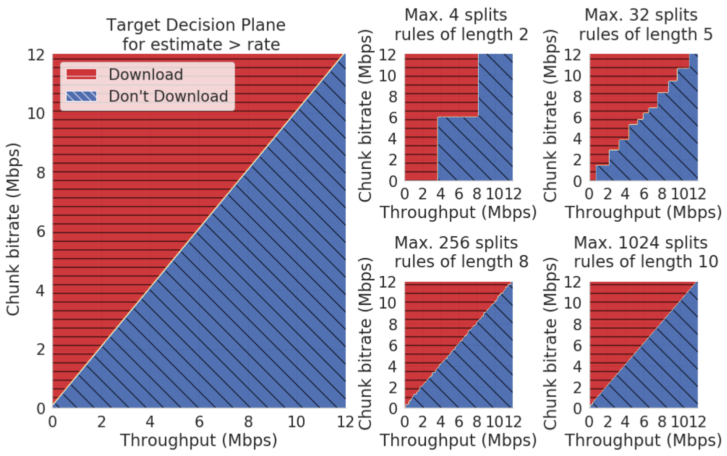


Figure 4.4: A decision plane comparing the throughput estimate to the chunk bitrate is difficult to capture without long rules if rules can only be framed in terms of the individual literals.

certain primitive or non-primitive features, but also on comparisons among features. For instance, generalizing even a simple rate-based algorithm to work for arbitrary videos encoded with different bitrate ladders requires a comparison: is the throughput larger than a particular step in the bitrate ladder? Unfortunately, while decision trees can capture such comparisons using only primitive features, they require a large number of rules to do so. This is shown in Fig. 4.4, where the decision trees with rules of length 2 and 5 do not accurately represent a simple comparison-based target decision plane.

Thus, we must encode potential comparisons of this type as non-primitive features. These can also be parameterized in a similar manner to the throughput estimators discussed above, *e.g.,* by what *factor* should one feature exceed another.

Planning ahead: ABR, like many other control tasks, is not only about making one decision in isolation, but also considering its longer-term influence. For instance, it might be interesting to estimate the max, min, or mean rebuffering one should expect given the decision to download a chunk at a certain quality, assuming the throughput stays the same. We design features along these lines, such as `QoE` in `MPC` [Yin+15].

More features? Over time more features can be added to enhance our approach without having to reason about their mutual interactions, as would be the case with incorporating new ideas into human-engineered routines. One could also extend this approach by adding automatically engineered features [GZN07]. However, maintaining interpretability would require limiting the complexity of auto-generated features.

4.5 Implementation

We implement the rule inference pipeline in Python3. For the decision tree, we use the standard implementation provided by the scikit-learn library [Ped+11]. If not otherwise mentioned we use a maximum of 20 rules and limit one-step changes in quality to upgrading or downgrading by at most 2 quality levels. The 20-rule limit is somewhat arbitrarily chosen as a quantitative threshold for interpretability, but we also find that for our approach, more rules do not improve performance substantively in *most* cases. This threshold is essentially a hyperparameter that could be tuned by practitioners based on where they seek to operate in the tradeoff space involving interpretability, avoiding overfitting, and performance.

Baselines: To put our results in context, we compare them against three neural network approaches, both as-is (blackbox approaches, always represented by a recursive neural network with GRU cells [Cho+14]), and translated to decision trees. The first blackbox approach is the simplest, attempting to directly copy the decisions in a supervised learning setting. The other two use more sophisticated imitation learning methods [Wan+19; HE16].

For translating the blackbox approaches into decision trees, we test two state-of-the-art methods [RGB11; BPS18]. One of these [BPS18] needs a reward function. In the vein of other imitation learning approaches, we use a clustering algorithm to assign a similarity reward to every sample. In our implementation we use an isolation forest [LTZ08] implemented in scikit-learn [Ped+11] with the standard parameters as our clustering approach. At every training step, we sample 3000 samples (as this gave the best results) according to the cluster weighting. We also tried changing the weighting function to a more agnostic divergence measure as the proposed decision by the blackbox approach might not always be what the original algorithm had in mind. This makes the sampling approach more robust.

For each reconstruction, when we compare results to our approach, we use the best blackbox approach and the best tree-translation. Thus, we invested substantial effort in implementing sophisticated baselines from the ML literature.

We also test whether our approach benefits from learning from the blackbox, instead of directly from the data. We find that this yields only minor improvements for 2 of our 10 reconstruction targets. We also explore learning in two passes, where in the first pass, we learn a tree over engineered features, and use a classifier to label its decisions in terms of their similarity to decisions made by the reconstruction target. In the second pass, we re-learn across weighted data samples, such that samples corresponding to more similar decisions are weighted higher. This approach also results in only minor improvements for one of our ten reconstruction targets.

Feature engineering: We instantiate our features (Section 4.4.3) with appropriate parameter ranges as below. 'Action' refers to quality decisions, such as maintaining quality, or increasing or decreasing it by up to n quality levels. The 'any' operator instantiates all options for a parameter or primitive.

1. Standard deviation, mean, harmonic mean, `EWMA`, and q^{th} percentile over the last $\boldsymbol{n}$ chunks, with $n \in \{1 \ldots 10\}$. Additionally, for `EWMA`, $\boldsymbol{\alpha} \in \{0.15, 0.35, 0.55, 0.75, 0.95\}$.

2. For q^{th} percentile, $\boldsymbol{q} \in \{15, 35, 55, 75, 95\}$.

3. Reward $\mathcal{R}$ achievable by planning ahead 3 steps for **any** action with **any** throughput estimate. The 'any' operators here imply that we have numerous reward features, each of which combines one of the many available throughput estimators (from 1. and 2. above) with one of the possible actions. As the reward function, we use the linear QoE function introduced by Yin et al. [Yin+15], which converts bitrate, buffering and bitrate change per chunk downloaded into a score. Note that this is not necessarily what any of our reconstruction targets is optimizing for – each provider may have their own reward goals. We use this feature simply as a compact representation of QoE components.

4. Fetch time for **any** action, **any** throughput estimate.

5. Bitrate gained weighted by the buffer filling ratio for **any** action, **any** throughput estimate. Intuitively, this captures the gain in bitrate relative to its cost, *i.e.,* how much the buffer is drained by an action if throughput stays the same.

6. VMAF gained weighted by the buffer filling ratio for **any** action, **any** throughput estimate. Same as above, but with VMAF.

Ultimately we make $\sim$ 1300 features available to the learner. Note the multiplicative effect of the **any** operator above.

Throughout, we use a standard training, validation, and testing methodology. The test set contains two videos combined at random with 60 traces randomly sampled from the overall set; these 60 traces are *neither* in the training nor in the validation set. We only discuss results over the test set.

Automated Feature Engineering: As a comparison and future outlook on the possibility of automated feature engineering, which has shown promise in other applications [GZN07], we also coarsely implement an automated feature generator. This generator recombines the raw features in an iterative fashion so that the most used features "survive" and get recombined and the others "die" out. We use the library gplearn [Ste17] with basic mathematical operators as usable functions. We limit the iterations to $s \in [50, 100, 150]$ seconds to avoid overfitting.

4.6 Evaluation

We summarize below the experiments we conducted as well as their top-line conclusions:

1. How well can we reconstruct target algorithms? We can mimic the decision-making of 7 of 10 targets to a high degree, and obtain high similarity scores.
2. What influence does domain knowledge have? Certain engineered features are crucial to obtain rules that generalize beyond training data, are concise, and achieve similar QoE as the target algorithms.
3. How interpretable and generalizable are the output rule sets? We find that we can easily spot flaws in the learned algorithm and propose ways to adapt it. Further, trees with only 20 leaves suffice in most cases.
4. How do deployed ABRs compare to academic ones? We find that academic ABRs generally outperform industrial ones, with the caveat that our evaluation uses metrics from academic work. Interestingly, we observe that one provider's algorithm shows behavior closely matching the well known BOLA algorithm, indicating potentially that this provider uses BOLA or a derivative of it.

4.6.1 Experimental methodology

Target platforms: We use the same 10 popular streaming platforms we used in our measurement study of deployed ABRs [LGS20]. While certainly not exhaustive, this is a diverse set of platforms, including some of the largest, such as Twitch and YouTube; some regionally focused, such as Arte, SRF, and ZDF; and some serving specific content verticals, such as Vimeo (artistic content), TubiTV (movies and TV), and Pornhub and XVideos. We exclude Netflix, Hulu, and Amazon Prime because their terms of service prohibit robotic interaction with their services [LGS20].

Different platforms encode content at varied resolutions and number of resolutions, ranging from 3 quality levels for TubiTV to 6.5 on YouTube (on average across our test videos; YouTube has available resolutions for different videos.) For Twitch, which offers both live streams and video-on-demand of archived live streams, we only study the latter, as live streaming is a substantially different problem, and a poor fit with the rest of our chosen platforms. For several of the providers we study, there are multiple implementations, such as for desktop browsers, mobile browsers, or mobile apps; we only attempt reconstruction for the desktop versions.

We also evaluate our ability to emulate well-known academic approaches for `ABR`. We use the `Robust MPC` (henceforth, just `MPC` throughout) and Multi-Video Pensieve (henceforth, `NN` , because it uses a neural network approach) implementation provided by the authors of the Pensieve paper [MNA17a]. We train and test these approaches on the Twitch video data set. To speed up our experiments, for `MPC`, we use a lookahead of 3 chunks instead of 5, finding that this did not make a large difference in performance.

Videos: The type of content can have a substantial bearing on streaming performance, *e.g.,* videos with highly variable encoding can be challenging for `ABR`. We thus used a set of 10 videos on each platform. Where a popularity measure was available, we used the most popular videos; otherwise, we handpicked a sample of different types of videos. Videos from each platform are encoded in broadly similar bitrate ranges, with most differences lying at higher qualities, *e.g.,* some content being available in 4K.

Network traces: Our experiments use synthetic and real-world traces from 3 datasets in past work [Akh+18; Rii+13a; Fed]. Unfortunately, a full cross-product of platform-video-trace would be prohibitively expensive — the FCC traces [Fed] alone would require 4 years of streaming time. To sidestep this while still testing a diversity of traces, we rank traces by their throughput variability, and pick traces with the highest and lowest variability, together with some randomly sampled ones.

Our final network trace collection consists of the 5 least stable, 5 most stable, and 20 random traces from the Belgium trace collection [Hoo+16]; and 10 most/least stable ones plus 25 random traces from each of the Norway [Rii+13a], the Oboe [Akh+18] and the FCC datasets.[5] We also use 15 constant bandwidth traces covering the range from 0.3 to 15 Mbps uniformly. Lastly we add 10 step traces: after 60 seconds of streaming we suddenly increase/drop the bandwidth from/to 1 Mbps to/from 5 values covering the space from 1.5 to 10 Mbps uniformly. If a trace does not cover the whole experiment, we loop over it.

In total, we use 190 traces with throughput (average over time for each trace) ranging from 0.09 to 41.43 Mbps, with an average of 6.13 Mbps across traces. Note that we make no claim of our set of traces being representative; rather our goal is to test a *variety* of traces.

Evaluation metrics: For training our approach and evaluating its accuracy in a manner standard in learning literature, we use two metrics: one measures *agreement*, and another the *similarity* of sets of decisions. We train towards maximizing the harmonic mean of these. Additionally, for our ABR-specific use-case, we evaluate the video quality of experience [Nat+19].

Agreement, F_1 score: For each output decision, we compute the precision and recall of the inferred algorithm against its target. The F_1 score is the harmonic mean of these. $F_1 \in [0, 1]$, with 1.0 being optimal. We compute an average over the F_1 scores across the labels in an unweighted fashion.

What is high/low agreement? If we were not interested in interpretability, we could obtain a procedure that mimics any target algorithm by using blackbox learning. We can think of the agreement

[5]Specifically, the stable collection from September 2017 [Fed].

such a blackbox approach achieves with its target as a baseline free of our conciseness constraint. On the other end of the spectrum, if the inferred rules do not achieve substantially higher agreement with the target than a generic ‘reasonable’ algorithm, then they are useless: this implies any reasonable algorithm would make at least that many decisions similar to the target. We use the simple rate-based approach as the concrete stand-in for this generic reasonable algorithm.

Similarity: As we cannot assume anything about how each provider designs their algorithm, we must use an agnostic approach in evaluating whether the experience under our reconstruction and the actual ABR is the same. Thus, we choose, as is typical in imitation learning, to learn whether the experience of two streaming sessions is “similar”. Similarity measures whether a set of samples (our reconstruction’s decisions) is likely to be from a given distribution (the actual ABR’s decisions). To classify whether a particular decision looks like it has been taken by the actual ABR or by our reconstruction, we choose an isolation forest [LTZ08].

Each of these two metrics is insufficient on its own. High agreement is useful, but making a few important decisions differently can substantially change a video stream’s behavior. Thus the need for similarity. However, there’s a benign solution to achieving high similarity: most commercial providers tend to keep the quality stable, so, by just keeping the same quality one can get high similarity. Conversely, agreement solves this problem: to get high agreement, we must match a large fraction of *each* decision type, matching only the “keep quality” decisions will result in poor agreement because of low matches on quality changes.

QoE: Agreement and similarity can be thought of as “micro-benchmarks” – these are standard measures in imitation learning, and are useful both for training our approach, and evaluating its learning accuracy. But ultimately, we also want to know: “How different is the user experience from an ABR versus from our reconstruction of it?”. We thus directly compare how well different components of a visual QoE metric used in earlier work [Nat+19] match up between a target and its reconstruction. As we show below, agreement and similarity correlate well with QoE: when a reconstruction achieves high agreement and similarity, it typically also performs like the target algorithm in terms of different QoE components.

Finally, we also comment on the role of domain knowledge in achieving good results with our approach, and the interpretability of our reconstructions.

4.6.2 Results

Agreement and similarity, Fig. 4.5: We compare the agreement and similarity achieved by our rule-set approach against the (best) blackbox approach and the simple rate baselines across all 10 online providers.We also include MPC and Pensieve (NN) evaluated on the Twitch videos.

The rule-sets achieve nearly the same or better agreement than the blackbox approach achieves for a reconstruction target in each case – in the worst case (NN), the rule-set's agreement score is 8% lower. Note that in many cases, we achieve higher agreement than even the blackbox approach. This is due to the imitation learning approaches trying to achieve higher similarity in terms of behavior rather than matching each individual quality decision.

The rule-sets also achieve high similarity in most cases, in the worst case (Twitch), achieving a ≈20% lower similarity score than the best blackbox approach, and in the mean, 5% higher. In contrast, the rate-based approach achieves not only very low agreement, but also very poor similarity.

Some readers may interpret the "low" absolute numbers in Fig. 4.5, *e.g.,* $F_1 \sim 50\%$, as a negative result. However, note that F_1 differences often don't cause appreciable video session quality differences, *e.g.,* if an ABR switches two quality levels in one step, and its reconstruction switches them in two successive steps, the F_1 score is lowered *twice*, but the video stream behavior changes negligibly. Also, rare labels (*e.g.,* increase quality by three levels) contribute equally to F_1 as

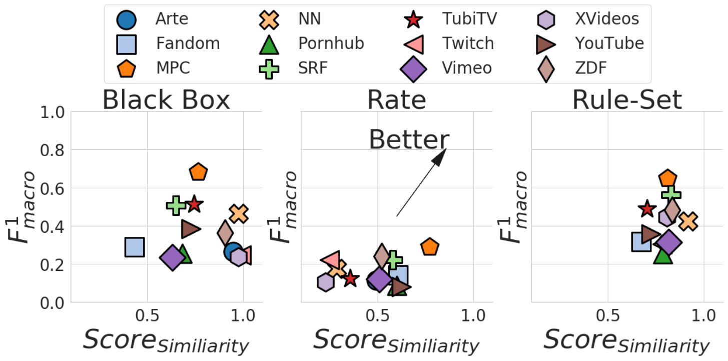


Figure 4.5: The generated rule-sets are never worse by more than 8% and 20% than the blackbox approach on agreement and similarity respectively. In contrast, the rate-based approach achieves extremely poor results.

common ones (*e.g.*, retain quality), so a few errors on rare labels have out-sized effect.

Video session quality metrics, Fig. 4.6: We compare metrics used to quantify video playback quality — `VMAF` [Li+16], `VMAF` switches, and rebuffers – as seen in the actual algorithm (hatched points in the plot) and its rule-set reconstruction (solid points) across the same set of `ABR`s as in Fig. 4.5. For 9 of 12 targets, we achieve a very good match: the mean `VMAF` (x-axis in Fig. 4.6) for these 9 reconstructions is within 6% of the target `ABR`'s on average; the maximum `VMAF` difference is 12%. These good matches include Twitch, SRF, Arte, ZDF, TubiTV, XVideos, Vimeo, `MPC`, and Pensieve (`NN`). On the other hand, for the other 3, YouTube, PornHub, and Fandom, there are large discrepancies, with quality metrics being very different for the reconstruction compared to the target. That our reconstruction does not yield good results on these targets is also supported by exactly these `ABR`s being in the low-agreement-low-similarity part of

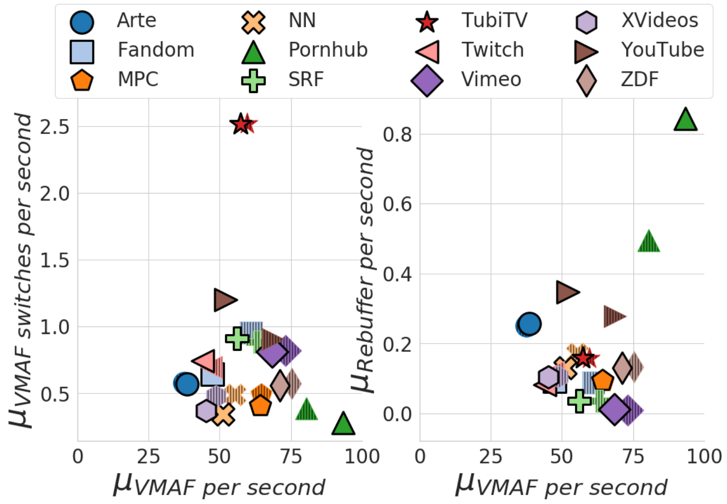


Figure 4.6: For all but 3 of the 12 targets, the reconstruction matches the target algorithm very closely. For YouTube, Fandom, and PornHub, there is a substantial difference in performance; these are the same 3 providers in the bottom-left of Fig. 4.5, for which we achieve the lowest agreement and similarity scores as well.

Fig. 4.5(bottom-left in the rightmost plot). We further investigated these 3 negative results:

1. YouTube, in addition to making quality decisions, varies its segment length and can also redownload low-quality chunks to replace them with high-quality ones [Mon+17]. Ultimately, learning approaches will not frame new decision spaces, only logic for arriving at the asked-for decisions – in essence, YouTube is solving a different problem than we expected. This is a fundamental gap for efforts like ours: if the *decision space* is not appropriately encoded, outcomes will be sub-optimal. We could add the relevant primi-

tives to achieve better results, but we resist such modifications that use the benefit of hindsight.

2. In a similar vein, we find that PornHub often switches to a progressive download, disabling video quality adaptation altogether. Our approach ends up overfitting to the progressive behaviour as we see few switches. If we exclude data where adaptation is disabled, we're able to match PornHub to within 4%, 0%, and 5% difference in terms of mean `VMAF`, rebuffering, and switching respectively.

3. For Fandom, we find that the issue is the limited complexity of our tree. A rule-set with a somewhat higher complexity (31 rules) performs substantially better, diverging from the target algorithm by 5%, 11%, and 22% in terms of mean `VMAF`, rebuffering, and switching respectively. Note that rebuffering and switching, being infrequent events are extremely difficult to *always* match, so a somewhat larger difference there is expected. As noted earlier, the rule-count is a hyperparameter that may need tuning for certain providers.

Role of domain knowledge, Fig. 4.8, 4.7: We already discussed in Section 4.4 why the use of domain knowledge is critical for interpretation: without simple primitives like moving averages, rules are framed in terms of various basic literals, resulting in complex and incomprehensible rules. Besides interpretation, we find that there is also substantial impact on agreement from adding useful domain knowledge.

We used our modified version of the `DASH` player to evaluate how the different trees emulating `MPC` generalize to other videos. We selected a mixed subset of 60 traces, that both include challenging and stable throughput measure and generated the distribution across them of linear `QoE` used in the `MPC` work [Nat+19]. Results are normalized to the mean `QoE` for the ground-truth `MPC` implementation across the same video-trace set.

Fig. 4.7 shows how the rule-set reacts to additional building blocks being available for reconstruction in the form of engineered features.

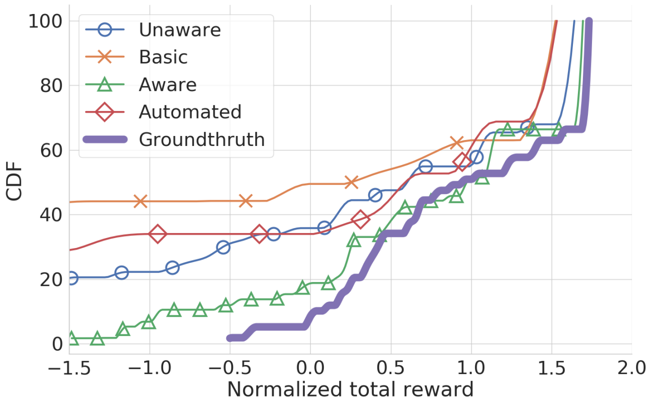


Figure 4.7: Domain knowledge helps the rule-set (Bitrate-`QoE`).

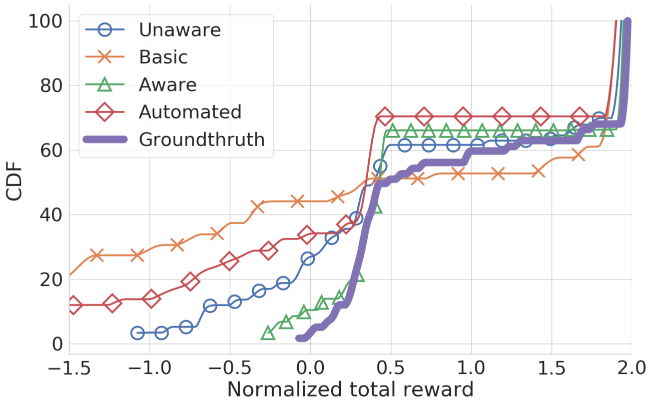


Figure 4.8: Domain knowledge helps the rule-set (`VMAF-QoE`).

The ‘Basic’ rule-set is framed directly on the data features listed in Section 4.3, without any feature engineering. The ‘Unaware’ and ‘Aware’ approaches use several engineered features, as described in Section 4.4. The difference between them stems from the ‘Unaware’ approach only using engineered features related to buffer filling and draining in addition to the primitive data features. The ‘Aware’ approach with the benefit of all engineered features matches `MPC` the closest. ‘Aware’ improves average `QoE` over ‘Unaware’ by ~5×. Thus, encoding domain knowledge helps not only with conciseness, but also performance and generalization. Also of note is the ‘Automated’ approach, which starts with the ‘Basic’ features, but can recombine them in the fashion described in Section 4.5. While promising for future exploration, it presently performs worse than manually engineering features, and does not produce features that are meaningful to humans.

Fig. 4.8 repeats the above experiment, but for a `VMAF`-based `QoE` function. The results are similar to those for bitrate `-QoE`. The average `QoE` of the ‘Aware’ reconstruction is within 10% of that of the target `MPC` algorithm, the median being within 2%. This is especially significant because we did not engineer any explicit features similar to this `QoE` function.

Interpretability: Across our efforts on reconstruction, the generated rule sets are concise, with no more than 20 rules. We realized early that being able to read and understand the generated trees would make debugging and improvements easier, and thus wrote a small, simple utility to translate the predicates in trees loosely into natural language. Fig. 4.9 shows an illustration of a tree generated for SRF. This version is hand-drawn for aesthetic reasons, but there is no fundamental reason it could not be auto-generated. Due to space constraints, this version is a compressed tree which was allowed to have at most 10 leaves instead of 20. We extensively examined and used our natural-language trees ourselves throughout this work, as we describe in a few instances in Section 4.7.

We also understand, to some extent, *why* small rule-sets suffice:

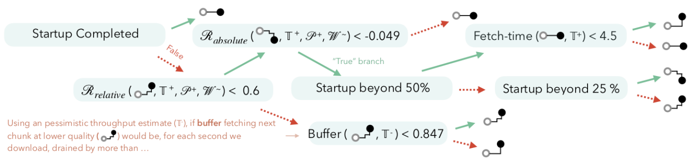

Startup Completed
False
$\mathscr{R}_{absolute}$ (, $\mathbb{T}^+$, $\mathscr{P}^+$, $\mathscr{W}^-$) < -0.049
Fetch-time (, $\mathbb{T}^+$) < 4.5
"True" branch
$\mathscr{R}_{relative}$ (, $\mathbb{T}^+$, $\mathscr{P}^+$, $\mathscr{W}^-$) < 0.6
Startup beyond 50%
Startup beyond 25 %
Using an pessimistic throughput estimate ($\mathbb{T}^-$), if buffer fetching next chunk at lower quality ( ) would be, for each second we download, drained by more than …
Buffer (, $\mathbb{T}^-$) < 0.847

Figure 4.9: The core of the decision tree generated by learning from the SRF example data. The green (solid) and red (dashed) arrows are the "True" and "False" evaluations of the predicates at the nodes. The video quality of the last fetched chunk is denoted by ∘ and the next chunk's by ● . Potential decisions and decision outcomes are coded in terms of the relationships between these qualities: *e.g.,* ∘● denotes that the next chunk is requested at one higher video quality level. The predicates are in terms of expected buffer size after a certain potential decision, based on throughput estimates (*e.g.,* $\mathbb{T}^+$ is an aggressive/optimistic estimator); or on a reward ($\mathcal{R}_{relative}$) calculated relative to the other obtainable rewards involving throughput, rebuffering penalty ($\mathscr{P}$), the lookahead horizon over which these are estimated ($\mathscr{W}$), etc.

(a) a single rule has implications capturing more than is plainly visible, *e.g.,* if the buffer is running too low, the best recourse is to lower quality, and not much new will be learnt from a long planning horizon; and (b) the domain-specific primitives are a dense encoding of useful knowledge. We caution readers against generalizing these observations to imply that small rule-sets will necessarily suffice for other problems where learning is effective — our exploration and results are limited to ABR. That small rule-sets would suffice for many ABRs, is also supported by prior work [DCK19] showing, for instance, that the neural network ABR approach, Pensieve, largely depends on only 3 features.

Comparing academic and industry ABRs, Fig. 4.10: For targets we

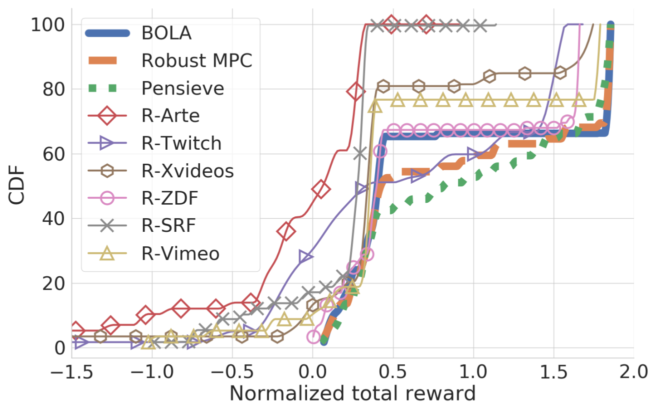


Figure 4.10: Reconstructing commercial ABR algorithms allows us to uniformly compare them to both other commercial and academic ones under the same test conditions.

can reconstruct well, having a reconstruction enables us to compare them to each other and academic ABRs in the same test harness. This is non-trivial without a reconstruction, as each video platform has a very different streaming setup in terms of encoding, server architecture, and player design. For instance, if one platform uses more encoding levels than another, then the same ABR algorithm can make more fine-grained decisions on this platform than on the one with coarser encoding. Therefore the same algorithm on the same video would perform differently across video platforms, making it difficult to compare ABRs across providers without removing such confounding factors in a common test harness.

To this end, we extend the DASH architecture [SSS18] with implementations of the (rule-set) reconstructions for the 6 targets we are able to match most closely. The same setup has ground truth implementations for BOLA [SUS16], MPC [Yin+15], and Pensieve [MNA17b]. We evaluate VMAF-QoE [Nat+19] using the Envivio video typically used in academic work, and normalize the results to the mean QoE for Pensieve.

As the results in Fig. 4.10 show, Pensieve and MPC generally outperform the deployed ABRs' reconstructions, although, for a subset of traces, R-Twitch achieves the same or better performance as MPC. This is perhaps not unsurprising: we are evaluating all providers with QoE functions used in academic literature, while the providers may, in fact, be optimizing for a different goal. Amongst the providers, R-Arte's ABR achieves the worst performance on this QoE metric.

But perhaps most striking is the distribution-wide extremely close match between R-ZDF and BOLA – except for a few outliers at the tails, for most video-trace pairs, their performance is virtually identical. Thus, it is likely that ZDF is using BOLA, or a derivative of it.

4.7 The utility of interpretability

Human insight can be crucial to robust solutions that account for gaps and unanticipated changes in the data that drives the behavior of learned control procedures. We discuss several ways in which preserving the ability of expert designers to understand the decision procedure helps.

Tracing the input-output mapping: With concise decision trees, human experts can easily trace the decision process used for particular inputs or sets of inputs. For any input conditions, a path can be traced to a leaf (decision output), and for any output, how it was arrived at can be understood as well. Such tracing can allow easy debugging — "Why were bad outcomes seen for these traces?". This also opens the door to more rigorous analyses of the outcomes for sets of inputs, based on methods like symbolic execution [Bal+18].

Identifying potential issues: Experts can often identify overfitting and other problems *a priori* if they understand the procedure, as is the case with the concise decision trees we produce. Our experience itself revealed three such instances:

(1) One feature we encoded for use in our decision trees was a prospective reward from fetching the next chunks at different bitrates. This worked for most videos, giving good average performance. However, for some videos with much higher/lower average bitrate than most other videos, results were inferior. This is due to the reward function using absolute bitrates, and thus not being meaningful across videos. Defining reward in relative terms, *i.e.,* normalized to the maximum possible reward, addresses this issue. A blackbox method, by hiding from human experts the logic used in the rules, makes such improvements more challenging.

(2) We noticed that even after training across the sizable network traces used in past work, our rule sets largely depended on optimistic estimators for throughput, unlikely to work well in more challenging environments, *e.g.,* new geographies a video service expands to where connectivity is more limited and variable. To force more conservative behavior, we can either add such traces to the training data, or restrict the learning approach to use only conservative throughput estimators leading to more stable behavior. Another possibility is to add new features to detect situations where conservative or optimistic behavior would be appropriate. Note that while *given enough appropriate data* blackbox solutions would also potentially overcome such problems, this requires noticing the problem in the first place. Also, such data may not always be available: *e.g.,* if the video service performs poorly in a geography, users may self-select themselves out by dropping the service, thus further skewing the data.

(3) Early in our experiments, we observed a peculiar learned rule that translates to "Never fetch the lowest quality after 45 video chunks." This stemmed from overfitting due to training on one video with 49 chunks (on which most other academic `ABR` work is also evaluated), where even over a sizable set of traces, typically a large enough buffer was built such that the lowest quality was ruled out for the last few chunks. While this particular artifact would be unlikely to

arise in a large provider's pipeline given enough diverse training data, similar problems may occur and go undetected in blackbox methods, especially when the underlying data changes, *e.g.,* if a short-form video service introduces longer videos.

Across these examples, blackboxes can hide problems that might otherwise have been obvious to human experts. Prior work [Kaz+19] has found problems of this type, *e.g.,* Pensieve, even if conditions are highly favourable, does not always download the last chunk of a video at the highest quality.

Finally, when such problems do present themselves, the recourse with blackboxes, depending on the problem's nature, can involve blindly tinkering with the inputs to the blackbox approach until the outcomes improve, or adding extraneous safeguards for each discovered problem.

4.7.1 Examining two reconstructions

We next give a view of two reconstructions of different complexity: SRF (simplified, same as in Fig. 4.9) and Twitch.

Simplified SRF: The output tree reveals an intuitive structure and highlights obvious flaws as we discuss below. (These are only present in the simplification, and not SRF's full tree.)

Fig. 4.9's caption explains how to read the tree. First it checks the point in playback, concluding that it is in a startup phase *if* playtime is below a threshold. In the startup phase, it checks *if* the possible gain in Reward is large enough to warrant the leveling up by two levels. This is only done *if* we deplete the buffer by not too much when doing so; etc. Of course, behind these loose statements are concrete, parametrized features which describe what the particular throughput estimator is, what reward function is used, etc.

An interesting characteristic of the simplified-SRF tree is that there are no quality changes beyond the startup phase. This is clearly unsuitable in practice, and would be an obvious red flag to any domain expert examining this tree. The full tree does, in fact, use adaptation after startup too, although it is infrequent. We have verified this behavior experimentally as well, where SRF makes frequent quality switches during startup, and much fewer later.

Twitch: Having examined a small simplified tree, we discuss the (full) reconstruction for a more complex target, Twitch.

Twitch's tree visually reveals 3 "branches" of activity, which we call panic, cautious, and upbeat. The panic mode is entered when the throughput is very low. Here the tree is most likely to downgrade the quality by two levels to try to build up buffer, regardless of current buffer occupancy. An example trace captured online shows such behavior at roughly 100 s in playback in Fig. 4.11.

The cautious mode is entered at mediocre connection quality and, unlike the panic mode, examines the buffer level. In this mode, the most likely action is to either keep the quality or, if buffer-level is low, downgrade it. Downgrading can also be induced by high throughput variance, which indicates uncertain networking conditions.

If throughput is above mediocre, the tree enters the upbeat mode. Here the most common action is upgrading the quality, or if we approach higher quality levels (and therefore, longer download times even with good network conditions), the decision to upgrade is weighted against the buffer drain it would incur, and the current buffer occupancy.

Unlike several other providers, Twitch's reconstruction reveals a willingness to switch qualities often. This is in line with our experimental observation that Twitch and `MPC` make similar number of switches in the same conditions, while other providers switch much less frequently compared to `MPC`. Based on this analysis, if a switching-averse provider wanted to adopt Twitch's approach by reconstructing it, they would have to suitably tweak it to reduce switching.

To summarize, with interpretability, we can catch problems before they occur, reason about generalization and behavior across sets of operating conditions instead of just point testing, and methodically discover and fix encountered problems.

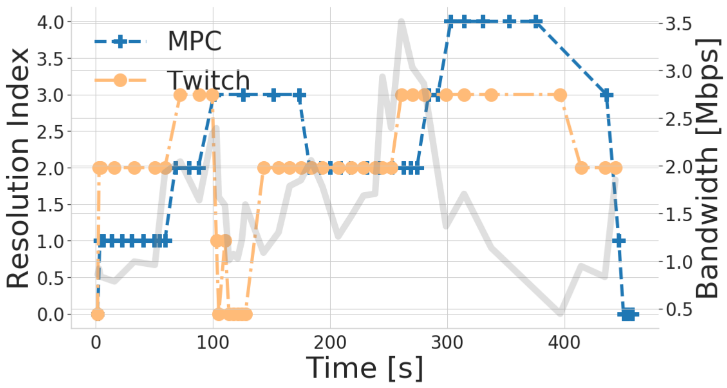


Figure 4.11: Twitch shows only marginally more reluctance towards switching when compared to `MPC`

4.8 Limitations & future work

Over the course of this work, unsurprisingly, we uncovered several shortcomings of our approach, which offer interesting avenues for future exploration:

- Accurate and concise trees require intuitive primitives, *e.g.*, moving averages, which must be manually encoded (Section 4.4). Perhaps such primitives can be automatically captured from a corpus of available hand-designed networked algorithms. But this is likely a challenging task.
- We explored a limited set of features, some across only a small part of their possible parameter ranges, *e.g.*, only 5 discrete values for the α parameter in moving averages. A potentially highly effective avenue of improvement lies in tuning the features using a black box optimizer, *e.g.*, a Gaussian Process Optimizer [OGR09], to suggest useful values.
- We can only train for an appropriately specified decision space,

as is clear from the failure of our approach for YouTube (Section 4.6.2). We can expand the decision space with the benefit of manually-drawn observations from experiments, but automatically discovering it seems difficult.

- We do not expect our approach to always be able to match a target algorithm. However, failures of our approach also help: they often flag “atypical” ABR designs for manual analysis, like for YouTube and Pornhub, and could help uncover unknown (proprietary) insights.
- We used an intuitive but *subjective* definition of “interpretable”: trees with under 20 leaves on domain-specific literals. Our own experience with understanding the results was positive, but we hope feedback from other researchers will help sharpen the interpretability goal for future work.
- For providers that customize their ABR for different regions and sets of clients, we can only reconstruct the behavior we observe from our test clients. For future work, this opens an interesting opportunity: observing differently-tuned versions of the same ABR, it may be possible to achieve higher-quality reconstructions, which also identify the parameters whose tuning varies across regions.

4.9 Conclusion

In this Chapter we took a first step towards an ambitious goal: reconstructing unknown proprietary streaming algorithms in a human-interpretable manner. While promising, our results also expose what is likely a fundamental limitation — we need to encode and make available suitable domain knowledge to the learning approach.

So far, this dissertation focused on understanding and reconstructing propertary video streaming algorithms deployed by large streaming providers. Our analysis and algorithm synthesis extensively used network and player traces.

Based on what we learned with our explorations, the next Chapter tackles how to optimise the offline part of VOD pipeline depending on the expected interaction between ABRs algorithms and network flow.

5 Optimising video streaming systems with Segue

In this Chapter we propose Segue, a system that, based on the online video streaming session flow, optimises the offline part of `VOD` pipeline. Specifically Segue identifies the parts of the video stream that are more likely to suffer from streaming-related impairments, and tunes video chunking accordingly.

5.1 Introduction

While Chapter 3 and Chapter 4 focused on the understanding and reconstruction of propertary video streaming algorithms, in this Chapter we will exploit the knowledge gained during our past analysis in order to optimise users' experience. Specifically, we will show how offline video preparation can be tuned to follow the *expected flow* of the network and player state, in order to maximise the average delivered QoE.

The traditional problem framing in past literature is that of using a provider's given QoE function to construct an ABR algorithm that will result in high QoE, as much as possible, when operating online across a range of network conditions and video content.

However, this framing leaves out the offline, provider-side phase of ABR: video chunking. We use the term "chunking" to refer to cutting a video into segments, and determining what set of bitrates each segment will be encoded in. Most prior works and deployed streaming platforms use constant-length segmentation, typically 4-6 seconds, and the same number of bitrate tracks across each video segment within one video. While some prior work (Section 5.2.2) has explored relaxing these constraints, we posit that there are new and unexplored opportunities at the intersection of the offline and online phases of ABR streaming. Specifically, offline chunking can be tuned based on the expected playback behavior of a video given a provider's online adaptation algorithm.

Video complexity varies over time. Indeed, we observe that some parts of a video are more likely to suffer from performance impairments such as lower quality, rebuffering, and frequent bitrate track switches during playback. Even two similarly complex segments of a video may differ in their vulnerability to performance impairments due to their surrounding context, *e.g.,* a complex segment preceded by many low-complexity segments differs from one preceded by many high-complexity ones. Finally, using different rate adaptation algorithms for the same video and network conditions can also change the same segment's vulnerability to impairments.

While prior work has explored time and space variability on a per segment granularity, to the best of out knowledge there is no prior work that accounts for playback context dependence and adaptation algorithm dependence in tuning video chunking. With SEGUE, we account for these factors in exploring the tuning of chunking along two axes: (a) segmentation, *i.e.,* deciding the lengths and boundaries of video segments that a client can fetch; and (b) augmentation, *i.e.*, adding additional bitrate tracks for a small fraction of segments to the provider's current bitrate track design, such that more bitrate options are available to online adaptation for these segments.

SEGUE uses a simulation-based method, exploring and evaluating segmentations and augmentations of a video across a broad set of network traces. In such simulations, the provider's ABR adaptation algorithm is used to make decisions, and their QoE function is used rank the candidate chunkings. We compare SEGUE's chunking performance to several heuristics drawn from intuition, revealing how the inability of the latter to account for context and adaptation leads to significantly worse performance than our proposal. While some of these heuristics still improve over constant-length segmentation and a constant set of bitrate tracks, the improvements are smaller than SEGUE's.

We implement SEGUE atop an unmodified H.264 video encoding pipeline. Our implementation uses *ffmpeg* with the *libx264* library. We modify the reference DASH player implementation [SSS18] to support SEGUE, but like past work [MNA17b], we use this implementation only to demonstrate the high fidelity of a simulator that is orders of magnitude faster. We then use the simulator to extensively evaluate the performance with SEGUE's chunkings across a diverse set of videos and network traces, and four adaptation algorithms from past work.

We show that especially in low-bandwidth conditions, SEGUE yields large improvements in QoE, 9% on average across traces and videos (Section 5.7.8).

To summarize, this Chapter makes the following contributions:

- We propose to optimize offline chunking, considering both the playback context and the adaptation algorithm.
- In this framework, we explore various methods for *segmentation* of a video into variable-length segments.
- We explore the *augmentation* of parts of a video with additional bitrate tracks to help adaptation make finer-grained decisions and

improve QoE.

- We evaluate SEGUE extensively, showing how its improvements depend on video content and adaptation algorithms. We also comment on the provider-side costs of SEGUE.

Perhaps equally valuable to SEGUE's optimization methods and results, are the questions it sets up for future work, on how we might co-design the offline and online phases on ABR streaming. For the benefit of future research along this path, we release SEGUE's implementation, together with our high-fidelity simulator [Lic+22b].

5.2 Background and related work

5.2.1 Video streaming 101

Adaptive bitrate video streaming comprises two pieces, video encoding, which runs offline at the content provider, and video adaptation, which runs online, typically at the client. Together, these optimize for improving quality of experience for clients, while limiting resource usage for the provider.

Encoding: Offline, a video is encoded into multiple tracks, each of a different *bitrate*. The bitrate describes compression, *i.e.,* the bits per second used to encode the content. Different tracks are described by their average bitrate, with substantial variation around this average due to variable bitrate encoding; this allows complex scenes to benefit from a higher than average bitrate, while reducing bitrate for simple scenes. The bitrates of different tracks are chosen for different target viewing *resolutions*. If a bitrate targeted at a lower resolution is delivered to a higher-resolution client viewing screen, the video can be scaled up, with some "pixelation". Video may be encoded such that for the same target resolution, multiple tracks with different bitrates are available.

Typically, each track is broken into fixed-length *segments*. For continuity when playback switches from one track to another, this segmentation must meet two constraints: (**C1**) segment boundaries of different tracks must be aligned in terms of content; and (**C2**) each segment starts with a *key frame*, *i.e.,* one encoded without reference to previous frames.

Adaptation: Online, an adaptive bitrate adaptation algorithm decides on which video bitrate-track to use. If the network provides consistently high bandwidth, the highest-bitrate track that makes a perceivable difference for a particular screen size can be used; otherwise, as bandwidth varies over time, lower-bitrate tracks may be used dynamically. A client-side buffer is used to absorb some bandwidth variability by storing video segments for future playback, but large or persistent bandwidth changes require shifting to a lower-bitrate track; otherwise, the playback buffer will empty out, and the client will see a pause or *rebuffer*.

Client QoE: Quality of experience metrics assess viewer satisfaction with video streaming. The relevant metrics include:

- The sum of bitrates across segments played;
- The sum of pause or rebuffer times; and
- The sum of bitrate differences from track switching.

Typically, a weighted sum of these metrics is used as a `QoE` function (carefully described in 2.3.2), with the weights drawn from past measurement work [Yin+15]. In line with newer work driven by industry shifts, instead of just bitrate, we use Netflix's `VMAF` score for perceptual quality [Nat+19; Li+16], which uses a learning model to assign a score to a segment's playback at a certain bitrate in line with what a human would rate its quality as on a certain screen size. The raw video has `VMAF`, $\mathcal{V}_{\text{raw}} = 100$, with different bitrates leading to `VMAF` scores from 0 to 100. We use the `VMAF` mobile, HDTV, and 4K models.

If $\mathcal{V}_i$ is the `VMAF` of the i^{th} segment at the track used for it, and R_i is the rebuffering time incurred during the i^{th} segment being played, then for a video with N segments in all, `QoE` is calculated as:

$$QoE = \lambda \sum_{i=0}^{N} \mathcal{V}_i - \beta \sum_{0}^{N} R_i - \gamma \sum_{1}^{N} |\mathcal{V}_i - \mathcal{V}_{i-1}|. \qquad (5.1)$$

λ, β, and γ are weights reflecting the value of each metric.

Provider resource usage: While client `QoE` is the determinant for many provider decisions, providers also want to contain infrastructure costs. Encoding video is compute intensive, and storing the segments for a large number of different tracks consumes storage at distributed caches. Thus, providers also attempt to limit the complexity of the encoding pipeline, and the number of tracks per video.

5.2.2 Related work

Adaptive bitrate video streaming is a broad research area; we only discuss the work closest to SEGUE's ideas.

Industry efforts: Netflix's per-scene encoding [Net18a] exploits the relative homogeneity of content comprising one scene within a video, to refine encoding decisions. The outcome of this process is *still* a fixed number of tracks per video. In a subsequent blog [Net18b] Netflix lays out how to merge shots into streamable segments, using a strategy that is very close to SEGUE's *Time* heuristic, explained in Section 5.4.

The innovation is rather in what specific bitrate is being used at each encoding point. Per-scene encoding, by selecting appropriate bitrates for each scene on each track, could potentially increase bitrate variations across scene boundaries, and thus provide *more* opportunities for SEGUE's methods. Absent an available implementation of Netflix's ideas, we have not quantified the impact of this yet.

Measurement work on YouTube [Mon+17] observed variable-length segments, with some evidence that during adaptation, YouTube uses shorter segments in response to bandwidth fluctuations. Unfortunately, no details are publicly known on how these are encoded or used. For instance: is each video coded with multiple equal-length segmentations? If so, how are the lengths decided, and how many different lengths are encoded? Alternatively, if segment lengths are indeed non-uniform like SEGUE, with only some parts of a video available in shorter segments, how are these parts chosen? Even if YouTube *is* pursuing a SEGUE-like method, that would only underscore the value in an *open* investigation of these ideas.

Segmentation: Prior work [VH13; IWG16; ZS18] has explored aligning scene boundaries and segmentation to improve coding efficiency by grouping homogeneous content together. As noted above for per-scene encoding, SEGUE's ideas are orthogonal to this, and address grouping which scene fragments into segments will result in beneficial rate adaptation behavior. Other work has attempted to optimize the (fixed) size of segments with the goal of improving transport [LBG11] or HTTP protocol efficiency [LBE16]. SEGUE's constraints like avoiding "too small" segments also address some of these problems, but its primary objectives and methods are very different: producing a variable-length segmentation that results in desirable rate adaptation behavior and high QoE.

The closest prior work [Sch+20] simply uses video group of picuters (GOP) as segments. This approach is prone to performance pitfalls, as discussed in Section 5.7.5.1. Another prior effort [Hoo+18] suggests segmenting the same video multiple times with different (fixed) segment lengths, to allow the client greater flexibility during adaptation. SEGUE's approach naturally inherits this property when it is desirable, without the expense of multiple redundant segmentations, as discussed in Section 5.3.2. SEGUE also allows more flexibility by not being restricted to a small set of fixed-length segmentations, allowing natural keyframe boundaries to determine segment length.

Augmentation: The closest prior works [Qin+19; Rai+17] pursue the opposite of SEGUE's strategy, *i.e.,* removing redundant segments to reduce storage costs or to improve bandwidth utilization. For instance, in Fig. 5.2, few segments across different tracks encode a near-identical perceptual quality; one could keep only the lower bitrate version, removing the higher bitrate ones, without much performance impact.

In contrast, SEGUE's optimization for improvements in QoE requires a completely different methodology, where accounting for playback context and rate adaptation algorithm is useful, as we show later. In Section 5.7.5.2 we compare the performance of [Qin+19; Rai+17] to SEGUE's approach, and we elaborate on how we could merge them, in order to both account for playback context and optimize for bandwidth utilization and storage.

Other work: Salsify [Fou+18] closely integrates the video codec and congestion control for real-time video like in video conferencing. Instead of fixed discrete encoding schemes and fixed frame-rates, Salsify sets each frame's quality and the time it is sent out, based on transport protocol signals. SEGUE is designed for video-on-demand, where live encoding per client is unnecessary and would be prohibitively expensive.

5.3 New opportunities in streaming

We draw out SEGUE's motivating observations using a running example of a video encoded using H.264 with variable bitrate encoding. The video is encoded into constant-length segments of 5 seconds each, across multiple bitrate tracks. (The details of the encoding are left to Section 5.5.) Using two rate adaptation algorithms, we evaluate the streaming behavior aggregated across a large set of traces. To avoid the effects of startup behavior, we show results starting at the 25th segment, *i.e.,* 125 seconds into playback.

Fig. 5.1(a) shows the variability of the video bitrate across segments for 3 tracks. As expected, variations for different tracks are in close alignment. Segment S37, S35, and S30 are the most complex, with the encoder using the highest bitrates, while S29, S33, and S39 are the simplest segments.

We partition our traces into bandwidth buckets, each bucket containing traces with average-over-time bandwidth in a certain range: 0.5-1 Mbps, 1-1.5 Mbps, 1.5-2 Mbps, etc. Details concerning the utilized

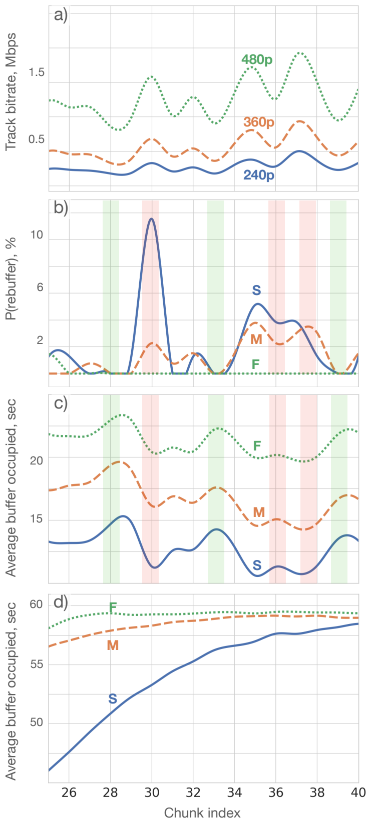


Figure 5.1: *Fig. (a) shows the variability in bitrate for three different resolutions (480p, 360p and 240p). Fig. (b) shows the observed probability of rebuffering for the segments plotted in Fig. (a) for a buffer-based algorithm for three different traces buckets (blue line: Slow, orange line: Medium, green line: Fast). Fig. (c) shows the correspondent average buffer occupancy. We highlight that higher observed rebuffering probability correspond to drops into the average buffer occupancy (red shaded in the picture). Conversely, drops into the observed rebuffering probability correspond to higher average buffer occupancy. Fig. (d) plots the average buffer occupancy for the same segments for a rate-based algorithm for the same traces buckets. The buffer behaviour varies substantially with respect to a buffer-based algorithm.*

trace sets can be found in Section 5.6. We then tested the behaviour of two adaptation algorithms, a rate-based (RB) and a buffer-based (BB) (which are described in Section 5.5). For both rate adaptation algorithms, we compute the (observed) probability of rebuffering at any point in playback across traces in each bucket. Fig. 5.1(b) shows the probability distributions for three trace buckets (S – slow, M –

medium, F – Fast) for a buffer-based (BB) adaptation algorithm from past work. Fig. 5.1(c) shows the corresponding average seconds of playback available in the client's buffer.

Playback context dependence: We observe from Fig. 5.1(a) and (b), that the probability of rebuffering of segments of similar complexity (bitrate) differs substantially depending on their placement in the stream. For instance, particularly for the lower-bandwidth bucket, even though S30 has lower bitrate than S37, S30 is substantially likelier to incur rebuffering. Thus, the playback context of a segment impacts its likelihood of suffering from performance impairments.

Adaptation algorithm dependence: Fig. 5.1(d) shows the average buffer occupancy for a rate-based adaptation algorithm, showing the stark contrast with BB in Fig. 5.1(c). Thus, for the same video and network traces, the same segment's vulnerability to performance impairments depends on the adaptation algorithm in use. (This is indeed obvious, our contribution is in accounting for and using this dependence.)

Network trace dependence: While ABR algorithms handle instantaneous bandwidth fluctuations, SEGUE finds common patterns among streaming sessions which lead to the determination of vulnerable parts of the video. These vulnerabilities depend not only on the expected playback state but also on the ABR adaptation logic. SEGUE uses a large number traces to minimize the effect of a single trace's fluctuation. The different trace sets are discussed in Section 5.6.

5.3.1 What levers can we tune?

Online rate adaptation must cope with highly unpredictable network bandwidth changes. However, the other time-varying determining factor, *i.e.,* video complexity variation and its interaction with rate adaptation, is more predictable, and can be accounted for in offline video chunking. We use two levers to adjust chunking to this end:

- Segmentation: we can adjust the lengths and boundaries of the video segments a client can later fetch.

- Augmentation: for select segments, we can add bitrate tracks to provide greater flexibility to online adaptation.

We next describe why these levers are interesting to tune, and some intuitions on how this might be done.

Segmentation: Fig. 5.2 shows the instantaneous bitrate per frame over playback time. The video is encoded using *ffmpeg* and H.264, with two-pass encoding. The red dashed lines show the key frames, with the rest of the frames encoded with reference to these. The maximum interval between successive key frames is passed as an argument to the encoder, and is 5 seconds in this instance. As Fig. 5.2(a) shows, key frames are not distributed uniformly across time: typically, relatively static parts of a video will feature larger gaps between successive key frames, while in complex, motion rich parts, key frames will occur more frequently. This property enables the use of key frames to group parts of the video with similar characteristics together. Recall constraint C2 from Section 5.2.1: video segments must each start with a key frame. We can thus collect such sets of adjacent key frames to form segments, but this will result in segments of non-uniform length. In contrast, a fixed-segment length setup forces the encoder to add key frames at fixed intervals corresponding to segment length, with additional key frames within each segment, as necessary. By carefully shaping segments of non-uniform length, we can let the client fetch shorter segments during parts of a video more vulnerable to streaming impairments, thus allowing finer-grained rate adaptation decisions. For less vulnerable parts, longer segments can be used.

Another aspect where variable-length segments help relates to the stability of perceived playback quality. Fig. 5.2(b) shows the perceptual quality (`VMAF`) computed *per frame* for a few constant-length segments of an action movie. For the segments highlighted in pink, there is a huge fluctuation in `VMAF` within the segment boundaries, perhaps due to a scene change. There can also be value in synchronizing these change points with segmentation, allowing more informed adaptation decisions that account for such changes.

Augmentation: Fig. 5.2(c) shows another aspect of temporal variabil-

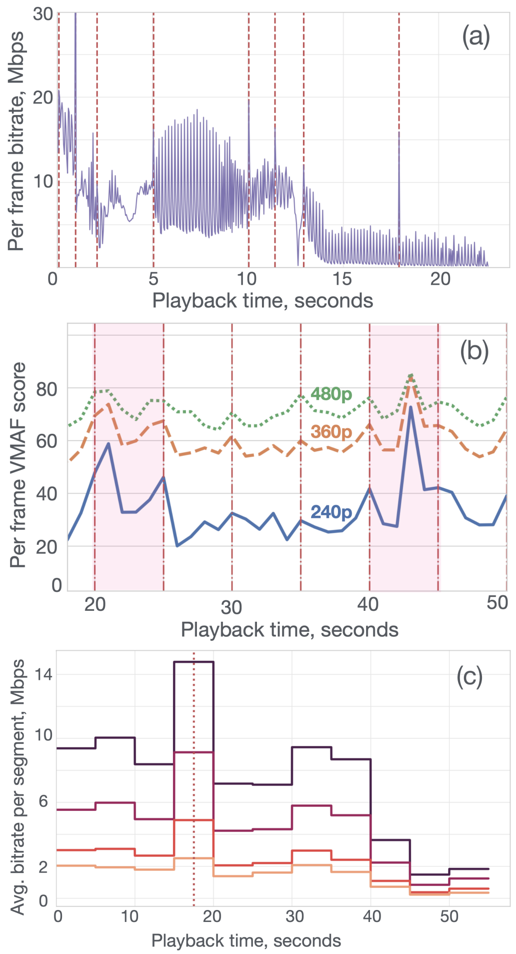


Figure 5.2: *(a) The violet (solid) line is the bitrate per frame, while the red (dashed) line marks the keyframes. (b) Breaking the video into fixed length segments produce segments with high internal perceptual quality instability. (c) Average bitrate for a video encoded at 4 resolutions: due to* `VBR` *encoding, the bitrate per segment varies.*

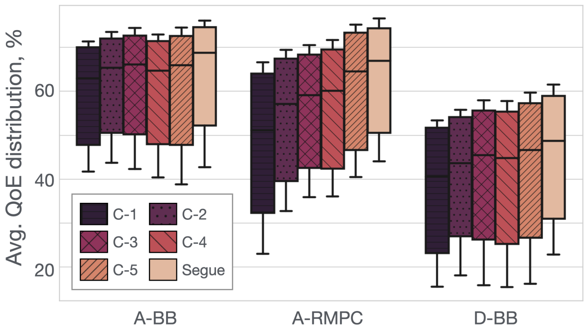


Figure 5.3: *The distribution of* QoE *per second of playback (normalized to max. possible QoE) across our test network traces for video* ***A*** *with* BB *and* RMPC *rate adaptation, and video* ***D*** *with* BB *adaptation. The box-plot shows the quartiles, with whiskers for 20th and 80th percentiles.*

ity using a video segmented into 5-second segments, with the average bitrate of each segment plotted across 4 tracks. Due to variable bitrate coding, the per-segment bitrate within each track varies substantially. In particular, the segment from 15–20 sec uses a much higher bitrate, so much so, that its bitrate at any track is comparable to the the rest of the video's bitrate at one higher track. This is because the encoder decides that the scenes of high complexity in this duration warrant higher bitrate for sufficient video quality.

Unfortunately, even with the freedom of variable bitrate coding, it is sometimes either hard to achieve sufficient perceptual quality for complex segments, or higher-than-necessary bitrate is "wasted" on simple segments.

For this problem, and the bitrate peaks illustrated in Fig. 5.2(c), instead of using the same number of tracks throughout, we could augment the encoding of segments vulnerable to streaming impairments with more tracks. These added choices would enable more fine-grained decisions during adaptation.

5.3.2 The need of variability

Before delving into Segue's design, we first illustrate the performance problems with baselines that do not account for variability in segments length and number of tracks.

We evaluate the QoE with different constant-length segmentations, ranging from 1 to 5 seconds (C1, C2, . . ., C5). Fig. 5.3 shows the QoE achieved across a set of test network traces for video **A**, using two rate algorithms (BB and RMPC) and for video **D** with BB. Comparing **A**-BB to **A**-RMPC, we see that for **A**-BB, C3 achieves a higher QoE than C5 especially at the lower tail, while C5 is better for **A**-RMPC. Similarly, comparing **A**-BB and **D**-BB reveals that for the same BB rate algorithm, C3 achieves better tail performance than C5 for video **A**, while for video **D**, C5 is marginally superior.

Note that just encoding multiple different constant-length segmentations and making them available to clients to choose from adaptively, as suggested in past work [Hoo+18], can address some of the above issues, but at huge expenses: if all of C1-C5 were made available, the content provider's storage and caching expense would be 5× larger.

In contrast to the above approaches, Segue consistently achieves higher QoE, as shown in Fig. 5.3, with only modest (under 10%) overhead in terms of additional bytes encoded.

5.4 Segue design

Segue tunes variable segment length and variable numbers of tracks across a video's segments to improve streaming quality. It does so in a manner that accounts for each segment's playback context and the rate adaptation algorithm. Segue uses the following inputs:

- A raw video whose encoding Segue will modify.
- The bitrate ladder the provider has designed for the video. This specifies the average bitrates of the different tracks.
- A target `QoE` function to optimize for.

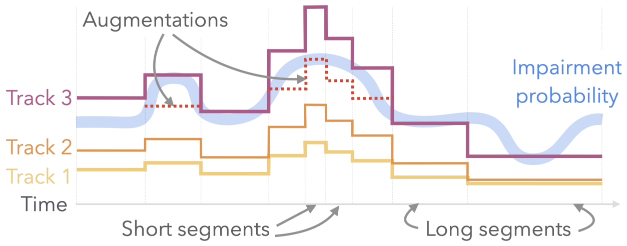


Figure 5.4: *We can segment vulnerable parts of the video into shorter segments, and augment them with additional tracks.*

- The rate adaptation algorithm the provider uses.

Given these inputs, SEGUE *segments* the video into variable-length segments, and *augments* some segments with added bitrate tracks. Fig. 5.4 shows a schematic of SEGUE's outputs.

5.4.1 Segmentation

Segmentation of a video must produce variable-length segments that should improve client `QoE` for the given `ABR`. SEGUE must output a segment sequence for every bitrate track specified in the input bitrate ladder. Further, the segments must obey the constraints **C1** and **C2** from Section 5.2.1.

Problem formulation: We first describe the problem ignoring the multi-track aspect. In this setting, segmentation involves first running a standard video encoder implementing the provider's codec of choice. We run the encoder on three inputs: the raw uncompressed video, the average bitrate to encode a track for, and a maximum gap between key frames. The encoder outputs a compressed video track of (roughly) the input average bitrate. We use this compressed track's key frame positions as an input for segmentation.

Each pair of successive key frames contains between them a video fragment. Our task is to decide which video fragments to combine together into segments for streaming. As we scan the video track from left to right and encounter a key frame, should this key frame demarcate the start of a new segment, or should we merge the video fragment between this key frame and the next into our current segment?

For certain simple optimization criteria, *e.g.,* minimize the number of segments while limiting the maximum segment length, the problem of finding the optimal segmentation can be framed elegantly as a dynamic program. However, this is not the case for the more sophisticated optimization criteria SEGUE uses to improve QoE, as we discuss below. Thus, we use brute-force search over a limited horizon, k, of future key frames: we allow each binary decision for each key frame, *i.e.,* merge with the previous segment or not. Each of the 2^k outcomes is a candidate segment sequence. SEGUE then uses one of two broad methods for assigning value to each segmentation, and choosing the best.

Intuitive heuristics: Segments that are too long or too short, or have too many bytes or too few bytes are undesirable. For instance, segments with too many bytes will increase the likelihood of rebuffering while they are fetched. Similarly, segments that are too short in their playback time will cause too many requests to the video server, and incur larger transport and application-layer protocol overheads. Thus, to prevent our segmentation from producing such undesirable frames frequently, we can penalize it for such segments. We frame three heuristics that penalize deviations from target values, where the target is defined in terms of:

- *Time*: Segments that are too long or too short in terms of their playback time in seconds are penalized.
- *Bytes*: Segments that have too few or too many bytes are penalized. The target number of bytes must be set based on the track and the video.
- *Time + Bytes:* Segments that are too long or too short are penalized, but there is an additional penalty if a segment exceeds a byte threshold.

In each case, we evaluate the 2^k segmentations and pick the one that minimizes the penalty for deviating from its target in terms of time, bytes, or a combination, as noted above. Only the first segment of the chosen segmentation is final; the procedure continues from the first key frame after it, in a sliding window manner, until all key frames are processed.

To extend to multiple tracks, we run the above process for the highest-bitrate track. With the segment boundaries known, we encode all the other tracks by asking the encoder to impose key frames at the segment boundaries.

Simulation-based assessment: Instead of applying heuristics derived from our intuitions, we can also just evaluate each of our candidate segmentations for our target `QoE` function and `ABR` adaptation algorithm, across a set of diverse test traces, and pick the one that performs the best.

Besides the philosophical distinction from the intuitive heuristics approach, a simulation-based approach requires a change in methodology. We can no longer start with a single-track approach and later use the segment boundaries to inform segmentation of other tracks. Instead, we need all tracks to be segmented simultaneously in progression, because the simulation will involve switching between tracks.

To this end, we again have the encoder encode the highest-bitrate track in the same manner as before. However, instead of optimizing segmentation using only this track, we also ask the encoder to encode all the other tracks enforcing all the key frames to be the same as those in the highest-bitrate track.[1]

We thus have all the tracks available simultaneously to perform segmentation on using a simulation.

For any candidate sequence of segments, $\mathcal{S}$, out of the 2^k options, the simulation, *Sim*, runs as follows:

[1]The impact of this imposition of keyframes on encoding efficiency compared to a standard `GOP` method is small: for our settings, the `VMAF` loss and the bytes overhead are negligible, both changing by under 0.03%.

1. For a set of network traces, we run the ABR on $\mathcal{S}$.[2]
2. For each trace, we compute the QoE, considering $\mathcal{S}$'s segments and all segments fetched preceding $\mathcal{S}$.
3. Across traces, QoE is aggregated based on a desired function, *e.g.,* the mean or an arbitrary specified percentile.

Across candidate sequences, the one that achieves the highest aggregated QoE across traces is selected, and its *first* decision — merge or not — is used. A merge decision results in the video fragment being added to the previous segment. If the decision is to not merge, the previous segment is closed. The simulation continues over the next k key frames, in the same sliding window manner as for the heuristics.

The above *Sim* approach is effective in many settings, but it can be myopic due to its limited lookahead, ignoring long-term effects of a segmentation strategy. However, a longer lookahead, k, is computationally expensive. We thus also test a *WideEye* strategy that combines *Sim* with the preceding intuitive heuristics: we use a longer lookahead, but we: (a) filter out candidate sequences using the *Time and bytes* heuristic; and (b) slide the window forward by multiple keyframes, thus freezing multiple decisions in each step instead of just one decision.

5.4.2 Augmentation

Augmentation aims to identify parts of a video vulnerable to streaming impairments, and add more bitrate tracks for their segments at appropriate bitrates.

Augmentation treats the video tracks and segmentation as inputs. The input bitrate ladder comprises a set $\mathbb{B}$ of tracks. The segment set $\mathbb{S}$ can be comprised of segments as today, equal-length, or be the output of our above segmentation. A video can be concisely described as a set $\mathbb{B} \times \mathbb{S}$ of segments across tracks. We define an augmentation

[2]Some algorithms, like Robust MPC [Yin+15], plan their decisions by looking ahead at several future segments. If this lookahead goes beyond the segments in $\mathcal{S}$, our simulation uses the future video *fragments* for this lookahead.

technique, λ, as a function $\lambda : \mathbb{B} \times \mathbb{S} \to \mathbb{A}$, with $\mathbb{A}$ being the set of new tracks added, with each element $a \in \mathbb{A}$ describing the position of a in the video and the average bitrate of a. We describe four augmentation functions, λ_v, λ_b, λ_{bv} and σ_{bv}, which use different heuristics to identify segments to augment.

λ_v based on VMAF drops: Despite the freedom afforded by VBR coding, complex scenes can end up with lower bitrate than needed to maintain perceptual quality. Our λ_v heuristic attempts to augment such parts of a video. Consider the i^{th} video segment on the j^{th} bitrate track, $s_{i,j}$. If the VMAF of $s_{i,j}$ falls below the median VMAF across segments in track j by a tolerance threshold, then $s_{i,j}$ is marked for augmentation. We add an additional encoding for the i^{th} video segment, using the average of the bitrates of $s_{i,j}$ and $s_{i,j+1}$.

λ_b based on bitrate peaks: Recall from Fig. 5.2(c) that segments corresponding to complex video scenes can be encoded at much higher bitrate than the average, with large gaps between the bitrates of successive tracks. This can make streaming these segments difficult, often requiring ABR adaptation to either switch to a lower track, or increase the risk of rebuffering.[3] By augmenting such segments with additional bitrates, we can offer greater flexibility in ABR adaptation.

Consider the i^{th} segment on the j^{th} track, $s_{i,j}$. If the bitrate of $s_{i,j}$ is above the average for track j by more than a tolerance threshold, $s_{i,j}$ is marked for augmentation. We then add an additional encoding for the i^{th} segment, with the bitrate corresponding to the average for track j. The VMAF of this newly added segment will lie between that of $s_{i,j-1}$ and $s_{i,j}$.

λ_{bv} using both bitrate and VMAF: Not all segments chosen by λ_b are challenging to stream in the same way. For instance, $s_{i,j}$ may have a high bitrate compared to track j's average, and would be augmented by λ_b. However, if the VMAF loss from downloading $s_{i,\boldsymbol{j-1}}$ instead of $s_{i,j}$ is relatively small then $s_{i,j}$ does not necessarily need augmentation.

[3] CAVA [Qin+18], which is designed to carefully account for variable bitrate, also experiences this trade-off, but it is better at navigating it than non-VBR-optimized algorithms (Section 5.8). Our goal is to improve the trade-off itself.

This is what our λ_{bv} heuristic does: $s_{i,j}$ is only augmented *if* its bitrate is large relative to track j *and* there is a substantial difference in the `VMAF` of $s_{i,j}$ and $s_{i,j-1}$.

σ_{bv}**, based on simulation:** Like for segmentation, we design an augmentation approach based on simulation. Unfortunately, the search space for augmentation is even larger than segmentation: each segment in our lookahead horizon can be augmented between every pair of its successive bitrate tracks. With just 6 bitrate tracks and a lookahead of 5 segments, the search space expands to ~1 billion iterations per simulation step. We limit this scope substantially by using the λ_{bv} heuristic as the basis: at each simulation step, we limit augmentation candidates to ones suggested by λ_{bv}. Each parameter configuration of λ_{bv} (in terms of the `VMAF` difference and bitrate difference thresholds) yields one candidate augmented segment sequence. We simulate ahead with each candidate sequence, as well as with the unaugmented (default) sequence. For each candidate, we quantify its `QoE` improvement compared to the default normalized by the overhead in terms of bytes added by that augmentation sequence. We pick the top scoring candidate, and continue this process from the next segment.

5.5 Implementation

We implement both the offline video chunking and online rate adaptation components to evaluate SEGUE.

5.5.1 Offline video chunking

SEGUE's chunking pipeline is implemented in Python3, and makes use of *ffmpeg* and libraries for standard codecs. Note that we are not devising new codecs, compression algorithms, or video formats; instead it is our explicit goal to stick to current, widely used codecs, as their implementations are heavily optimized, with mature provider-side pipelines, and client-side decoding often offloaded to hardware.

Rather, we use the same codecs in a manner different from that in ABR video streaming today, as described in Section 5.4.

Since the availability of raw video data sets of sufficient length for interesting ABR adaptation is limited, we instead use 4K compressed video as a stand in for raw video, and then limit our work to resolutions 1440p and lower. The bitrate ladders we adopt throughout are from Bitmovin [Bit19a], but arbitrary other bitrate ladders, including those customized per title [Aar+] could be used as input. We also follow the guidance in that reference for encoding, using the recommended maximum bitrate of a track, *i.e.,* 1.75× its average. Throughout, we use two-pass encoding, as is typical in the industry [Oze19].

Segmentation: We implement the constant-length segmentation strategy common today as the baseline. We use *ffmpeg-libx264*, which allows us to specify certain key frame locations precisely, with the encoder potentially inserting additional key frames as necessary. This enables straightforward implementation of both the constant-length segmentation, as well as our segmentation heuristics (Section 5.4.1). We use the following configuration parameters:

- The *constant*-length baseline uses 5s segments.
- The lookahead of video fragments for all our segmentation methods except *WideEye* is $k = 5$, such that each iteration evaluates all 2^5 segmentations of these fragments.
- For *Time*, the target segment length is 5s, with a penalty of 20% per second for deviations.
- For *Byte*, the bytes-per-segment target is the average bytes in 5s of video; excess bytes incur 20% penalty.
- For *Time+Bytes*, besides the time penalty, the byte penalty is also imposed for segments with too many bytes.
- For *Sim*, the QoE of a candidate sequence is aggregated as the mean QoE across traces.

- *WideEye* has a lookahead of 10 keyframes instead of 5, and a decision window of 5 instead of 1. We only simulate the 32 best candidate sequences as ranked by *Time+Bytes*.

The penalties thresholds have been tuned to better work with our encoding settings, while the simulation lookaheads have been picked to keep reasonable computational time.

Augmentation: Our augmentation strategies are simple to implement as described in Section 5.4.2 using *ffmpeg-libx264*: regardless of the particular augmentation function, we merely need to encode a specific time range of video at a particular average bitrate, as a standalone segment. The different augmentation strategies are configured as follows:

- λ_v : segments are augmented when their average VMAF is $\geq$ 8 points lower than the median for their track (a value that corresponds to a bump from 720p to 1080p on a 4k TV [Nat+19]).
- λ_b : segments are augmented when their bitrate is $\geq$ 10% above than the average bitrate for their track. This leads to an aggressive augmentation strategy, intended to provide an upper bound QoE gain of this general method.
- λ_{bv} : We tested λ_{bv} for several different configurations. Segments are augmented when their bitrate is $\geq B$% above the average for their track *and* the VMAF difference between their track and the one below exceeds V points, being V in $\{5, 6, 7, \ldots, 14\}$ and B in $\{5\%, 10\%, 15\%\}$.
- σ_{bv} : We generate 30 candidate sequences by running λ_{bv} with these thresholds — VMAF difference in $\{5, 6, 7, \ldots, 14\}$ and bitrate difference in $\{5\%, 10\%, 15\%\}$.

For augmentation, as well as for later evaluation, we need to compute the perceptual quality score, VMAF, for a video segment. We use the code made available by Netflix [Nat+19; Li+16]. For computing our augmentation strategies, we use the VMAF 4K model, while for our evaluation, we additionally evaluate the VMAF HDTV and VMAF Mobile models.

5.5.2 Online playback and rate adaptation

The approach we explore deliberately steps outside the DASH standard [SSS18], with constant-length segments and a fixed number of bitrate tracks per segment. We thus modified the DASH player to support Segue. However, we use this implementation only to verify the fidelity of an orders-of-magnitude faster simulator, which we use for extensive experimentation. The simulator is implemented following the methodology described by the Pensieve authors [MNA17b]. Section 5.7.7 details the DASH player implementation, demonstrating that it achieves results near-identical to the simulator.

We simulate both the network and the player state. The network environment takes as an input a trace of bandwidth over time, and simulates the download of segments. The link RTT is set to 80ms in our experiments. The player simulator interacts with the network environment by requesting the download of a certain video segment from a certain track (as decided by the adaptation algorithm), and adds the segment's playback duration to the playback buffer. Meanwhile, it also drains the buffer. The maximum playback buffer size is limited to 60s; if the buffer is full, the player waits before downloading additional video segments. The number of seconds of buffered video needed before the player starts playback is set to 10s, following prior work [Qin+18].

The simulator logs rebuffering time and the downloaded tracks, allowing us to calculate QoE metrics in hindsight.

We evaluate Segue on four different ABR algorithms:

Rate-based adaptation (RB) tries to fetch the next segment at a bitrate matching the estimated network bandwidth. We adapt the simple, demo implementation of this strategy provided by Bitmovin [Aya+18]. This approach requires no modification to use Segue's modified encoding.

Buffer-based adaptation (BB) makes decisions entirely based on the player buffer state [Hua+14]. Briefly, BB uses two parameters: reservoir, r, and cushion, c. If the buffer size, b, is smaller than r, the lowest-bitrate track is used. If $b \geq r + c$, the highest bitrate is used.

For $b \in [r, r + c]$, bitrate tracks are (roughly) linearly matched to the buffer sizes.

BB requires modest changes with SEGUE. Besides changing r dynamically to adapt to variable bitrate coding as suggested in the original paper [**reservoir_basedl**], we also bound r by a minimum of 8 seconds to account for variable-length segments. Further, when $b \in [r, r+c]$, we first map b to a bitrate range based on the unaugmented bitrate tracks available, but if additional tracks were made available by SEGUE, we further linearly match within this range to the appropriate track. These minor implementation tweaks substantially improve performance compared to a naive implementation.

Robust MPC (RMPC) uses control theory [Yin+15]. It uses the bandwidth estimate, current buffer size, and features of upcoming segments, to plan a sequence of requests based on the expected reward. It is flexible enough to incorporate knowledge about varying segment lengths and augmented bitrates. We tested two versions of RMPC: (a) RMPC-oblivious, where, as in [Qin+18], we modified the RMPC reward function to work with the instantaneous segments bitrates (rather than fixed weights); and (b) RMPC-aware, where we modified the reward function to account for VMAF score rather than bitrate. For both versions, to accommodate segments of different length, we weigh each segment's bitrate gain by its length.

5.6 Evaluation methodology

We evaluate our approach across network traces used in past work on ABR streaming, and test several videos.

Network traces and VMAF: We use a mix of ~600 traces across broadband 4G, HSDPA, and 3G networks [Akh+18; Rii+13b; Uni16]. The mean throughput of these traces spans from 350 Kbps to 60 Mbps. We divide the evaluation traces into three buckets:

- SLOW, containing traces with mean throughput <1.5 Mbps.
- MEDIUM, with mean througput between 1.5 and 4 Mbps.

- FAST, with mean througput >4 Mbps.

Train and test separation: Our simulation-based methods are data-driven. We use only 20% of the above ~600 traces to make segmentation and augmentation decisions.

For robustness, we test not only on the unseen 80% of traces from the above set, but also on an entirely different trace distribution from the Puffer project [Uni20]. We sampled Puffer traces from Dec. 2020 to May 2021, arbitrarily using data from the 5th of each month. We retain only those traces that are longer than 2 minutes, corresponding to 64% of Puffer traces. 3.1% of these traces fit the SLOW class, 5.6% MEDIUM and 91.3% FAST. Our test set uses an equal number of Puffer and non-Puffer traces, *e.g.,* half of the SLOW test set is a random sample of SLOW-class Puffer traces, while the other half is from the 80%-unseen data from the other trace distributions mentioned above. Note again that this implies that none of the test data has been used in decision-making, and that half of it comes from an entirely different data source. (Limiting our evaluation to only the Puffer trace dataset only makes the results more favorable to SEGUE.)

Unless noted otherwise, we evaluate SLOW traces on the `VMAF` mobile model, MEDIUM on HDTV, and FAST on 4K.

Videos: We use a set of 11 videos with different content, downloaded from YouTube, listed in Table 5.1. These videos are available in 4K, which we use as "raw" (Section 5.5.1), and then run experiments for 240p, 360p, 480p, 720p, 1080p and 1440p.

QoE function: Unfortunately, with variable-length segments, we cannot use the `QoE` function used in past work as is, because it aggregates `QoE` metrics across *equal-length* segments (Section 5.2.1). Instead, we adapt the formulation to sum `QoE` per unit time, at a granularity of 1s. This is small enough to capture any impact from our use of smaller segments.

This implies that we have to adjust the weights λ, β, and γ for the `QoE` components corresponding to `VMAF`, rebuffering, and `VMAF` switches respectively: the original weights used in past work, are for 4 second segments, and using that same formulation on 1s intervals

ID	Duration [mm:ss]	FPS	Content type
A	3:21	24	3D cinematic
B	3:25	25	Music video
C	6:34	25	Comedy
D	3:51	25	Festival
E	4:42	30	Action movie
F	5:49	30	Cooking tutorial
G	2:33	30	Sports (long-take-shot)
H	2:40	30	Sports (highlights)
I	2:39	24	Underwater
L	3:16	30	Drone footage
M	2:40	30	Video game

Table 5.1: *An overview of our video dataset. Videos* ***C*** *and* ***D*** *lie at the extremes of highly stable and unstable in terms of perceptual quality over time within one track.*

would effectively assign 4× the importance to `VMAF`. We thus scale λ by $\frac{1}{4}$. Further, as we compare schemes with different segment lengths, we cannot ignore the startup phase: doing so would benefit schemes that fetch longer segments, as they would build up more buffer. We simply account for the initial phase in the same manner as any other segment, incurring a rebuffering penalty until the first segment is downloaded and played.

To use `VMAF` instead of bitrate as in the `Robust MPC` work [Yin+15], we also need to adapt the weight for rebuffering. `MPC`'s `QoE` function, drawn from measurement work, penalizes each second of rebuffering, *i.e.*, β, as equivalent to losing 4s of full-resolution bitrate. For `VMAF`, full-resolution translates to a value of 100. Thus we use $\beta = 100$.

We decrease the switching penalty, γ, from 2.5 to 1. With a 1 sec cadence for `QoE` evaluation, we measure switching more often than prior work. This accounts for intra-segment changes in `VMAF`, and penalizes any additional switching caused by our potentially shorter segments. (Using prior work's $\gamma = 2.5$ setting only improves

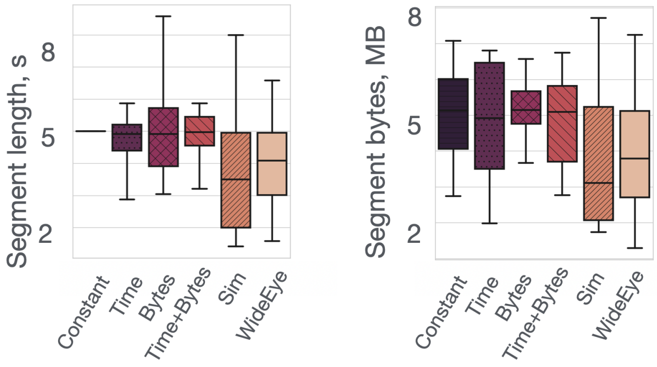


Figure 5.5: *Segment characteristics for different schemes — video* ***A****,* RB*. Boxes show mean and quartiles, whiskers are 5/95-percentile.*

Segue's results.)

In any case, Segue can be run with arbitrary `QoE` functions.

5.7 Results

We first describe the improvements from Segue for video **A** and `RB` adaptation. This helps draw out intuition in detail. Later, we evaluate Segue across 11 videos, 4 adaptation algorithms, hundreds of network traces, and 3 `VMAF` models. We then compare Segue's performance with the closest related works, and discuss its computational cost.

5.7.1 Segue's segmentation

Fig. 5.5 shows the characteristics of the segments produced by different approaches. *Time* and *Bytes*, by design, produce segments of similar duration and bytes respectively to *Constant*. However, by

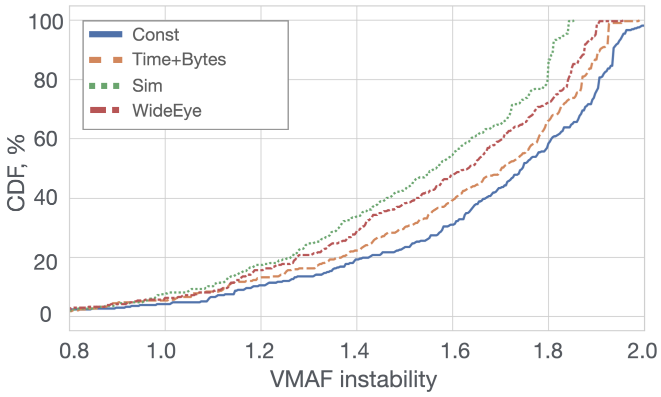


Figure 5.6: *`VMAF` fluctuations: video **A**, `RB` adaptation, SLOW traces.*

constraining only one factor, they introduce large variations in the other. *Time+Bytes* constrains both, and is thus conservative in its segmentation. *Sim*, with no direct constraints, naturally produces segments with the greatest variability. Consider, *e.g.*, a low-complexity credits scene, for which *Sim* may produce a very long segment to prevent `RB` from incurring switching penalties in `QoE`. *WideEye* strikes a balance, allowing greater freedom in segmentation than *Time+Bytes*, but trimming out *Sim*'s extreme, myopic choices. *Time* and *Bytes* are the least performant schemes, with obvious reasons, so we omit further discussion of these.

We measure `VMAF` fluctuation as the average change in `VMAF` per second of playback. We normalize this by the average `VMAF` fluctuation experienced by *Constant* across our full cross product of videos, traces, and rate adaptation algorithms. We calculate rebuffer ratio as the sum of seconds of rebuffering experienced during playback divided by video duration, and reported in seconds per minute (s/m).

Fig. 5.6 shows `VMAF` fluctuations across the SLOW traces. *Sim* achieves the most stable streaming, at the cost of higher rebuffer ratio (by 1 s/m) compared to *Constant*.

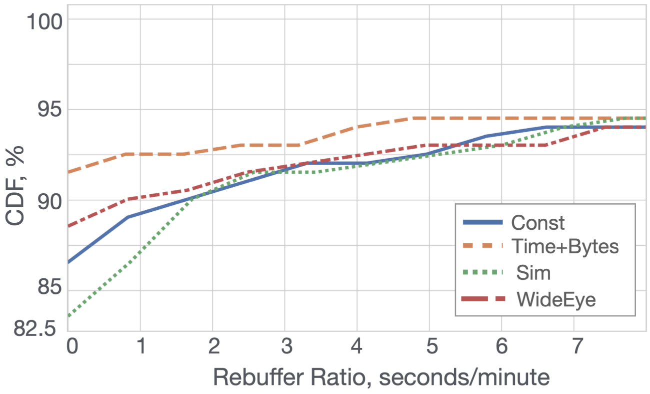


Figure 5.7: *Rebuffer ratio: video* **A***,* RB *adaptation, SLOW traces.*

Time + Bytes improves `VMAF` stability modestly, while simultaneously improving rebuffer ratio by 0.7 s/m compared to *Constant*.

Sim's numerous overly long segments, which help drive `RB` away from the frequent track switching it is prone to, result in an increased risk of rebuffering (Fig. 5.7). This is a consequence of its short-term, myopic decision making, which does not account for future risk of rebuffering. *WideEye* strikes the more favorable tradeoff here, not only improving stability substantially, but also limiting rebuffering to only 0.1 s/m higher than *Constant*, compared to *Sim*'s 1 s/m.

Changes in delivered `VMAF` are modest, with *WideEye* improving over *Constant* by ~0.8% for SLOW/MEDIUM traces.

Takeaway: Intuitive heuristics like *Time+Bytes* conservatively perform segmentation, avoiding risks like overly long or large segments. On the other hand, a short-term simulation approach can be overly aggressive, and increase longer-term rebuffering risk. Merging intuition with a longer-term simulation horizon strikes a favorable tradeoff.

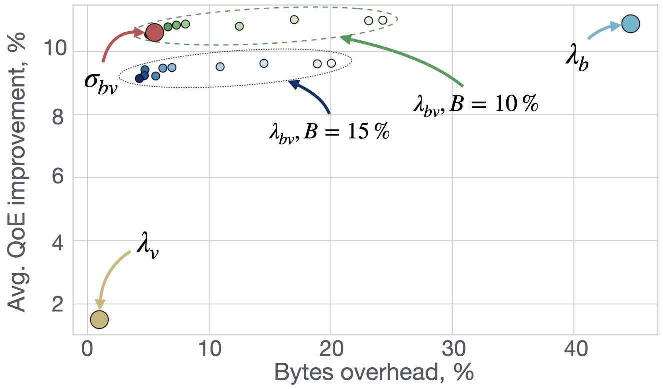


Figure 5.8: *σ_{bv} improves* `QoE` *with only small byte overheads, as does λ_{bv} with appropriate parameters. λ_{bv} is shown with $V \in \{5, 6, 7, \ldots, 14\}$ for both $B = 15\%$ and $B = 10\%$. ($B = 5\%$ is similar to $B = 10\%$.)*

5.7.2 Segue's augmentation

We next examine `QoE` improvements for video **A** with `RB`, by comparing *Constant* to Segue with *WideEye* segmentation coupled with each of our 4 augmentation heuristics in Fig. 5.8.

λ_v (bottom-left, yellow) and λ_b (top-right, cyan) show extreme points: the former augments too few segments and results in negligible improvements, while the latter augments too many segments (incurring more than additional bytes for encoding) to achieve its substantial `QoE` gains.

Our simulation-based strategy, σ_{bv}, (top-left, large red dot) achieves both high `QoE` improvement and low overhead in terms of bytes, due to its careful choices of which segments to augment. Mean `QoE` improvements compared to *Constant* are 24.9%, 4.1% and 1.4% for SLOW, MEDIUM, and FAST traces respectively, at the cost of 5.5% of more bytes encoded.

We also find that for our `QoE` reward and byte overhead definitions,

λ_{bv} achieves similar results as σ_{bv}, if λ_{bv}'s parameters are appropriately tuned (specifically, using a bitrate threshold, $B = 10\%$, and a VMAF threshold, $V = 13$ or 14).

We find that a small additional amount of bytes encoded for augmentation improve VMAF stability and reduces rebuffering, together with modest improvements on the average VMAF delivered. It is worth noting that augmentation is not as simple as "augment more bytes for higher QoE"; in fact, there are several heuristics that incur higher overhead, with lower QoE benefit, *e.g.*, compare several of the λ_{bv}, $B = 15\%$ results in Fig. 5.8 to σ_{bv}.

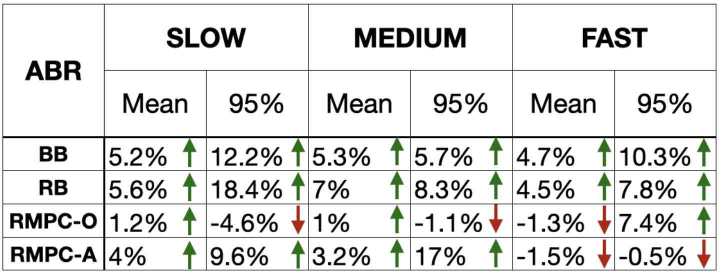
Takeaway: The simulation approach, by explicitly trading off QoE improvements with their cost, appreciably improves QoE at low overhead in terms of additional encoding and storage. At the same time, a careful tuning of parameters for a static policy can produce comparable results.

Takeaway: More augmentation does not always improve QoE; rather segments to be augmented need careful choice.

5.7.3 The impact of the adaptation algorithm

Segmentation: Across our experiments, the largest improvements from segmentation come from VMAF stability during playback, typically with some improvements in rebuffering, and negligible changes in VMAF. However, the details differ across rate adaptation algorithms as we discuss next.

Fig. 5.9 shows the improvement in VMAF stability. For each adaptation algorithm, we compute the VMAF switching penalty term of the QoE aggregated across the cross-product of videos and traces; we then show the mean and 95th-percentile in the table cells. The largest improvements are seen for RB, followed by BB, and the two versions of RMPC. For RMPC-oblivious we even see a deterioration, *i.e.*, higher VMAF switching by 4.6% at the 95th-percentile. It is worth noting that our segmentation simulations always use the VMAF 4K model to make decisions, while the evaluation uses different VMAF models for



ABR	SLOW		MEDIUM		FAST	
	Mean	95%	Mean	95%	Mean	95%
BB	5.2% ↑	12.2% ↑	5.3% ↑	5.7% ↑	4.7% ↑	10.3% ↑
RB	5.6% ↑	18.4% ↑	7% ↑	8.3% ↑	4.5% ↑	7.8% ↑
RMPC-O	1.2% ↑	-4.6% ↓	1% ↑	-1.1% ↓	-1.3% ↓	7.4% ↑
RMPC-A	4% ↑	9.6% ↑	3.2% ↑	17% ↑	-1.5% ↓	-0.5% ↓

Figure 5.9: *`VMAF` stability improvements divided by trace set and `ABR`. As expected, improvements are more significant for `RB`, given that no stability policy is implemented in the `ABR`.*

different trace buckets. If we evaluate using the `VMAF` 4K model, the result for `RMPC-O` is also positive, with 6.7% improvement. Using different `VMAF` models during segmentation tuning results in different weights for rebuffering, `VMAF`, and `VMAF` switching (*e.g.*, the mobile model is the most permissive for `VMAF`, weighting rebuffering more), so it is an open question as to how to optimize robustly against these differences.

For rebuffering, the differences from Segue's segmentation are smaller, with meaningful differences only at the tail. This is inherent to rebuffering: it is a corner case, as most rate adaptation approaches are conservative enough to avoid it in the typical case. For `BB` and RB, the number of traces for which rebuffers occur is cut by 1.3% for both, while for `RMPC-O` and `-A`, 0.2% and 1.3% more traces see rebuffers with Segue's segmentation compared to *Constant*.

We dissected the tail rebuffering and switching of Segue's segmentation with `RMPC` more deeply. `RMPC` plans for a limited lookahead of segments (5 in that paper), and when Segue produces several short segments, the lookahead becomes more and more myopic in terms of time, thus causing poor long-term planning. Thus, `RMPC`'s implicit assumption that a certain number of segments comprises a long-enough future planning horizon is contradicted in SEGUE. Unfortunately, increasing RMPC's lookahead is computationally expensive, so if the algorithm is not modified, there are two solutions: (a) disallowing series of short segments; and (b) instead of optimizing for the mean in SEGUE's segmentation, as we currently do, optimizing for higher percentiles (see §5.4.1). That *Time+Bytes* rebuffers on 1.9% and 2.1% fewer traces for RMPC-O and RMPC-A shows promise for strategy (a).

We also briefly illustrate the specificity of SEGUE's segmentation to different rate algorithms with experiments on video **A**: tuning segmentation using WideEye for RB and then using RMPC-O adaptation online actually *degrades* QoE by 7% compared to Constant, while correctly tuning segmentation for RMPC-O *improves* QoE by 6% compared to Constant.

Takeaway: A mismatch in what rate algorithm segmentation is tuned for versus used with can degrade QoE

Takeaway: Segmentation's interactions with rate adaptation algorithms that implicitly or explicitly assume constant length segments require additional effort to either modify such rate algorithms, or SEGUE's interaction with them.

Augmentation: Augmentation typically improves all three QoE metrics, at the cost of a small provider-side compute and storage overhead. The gains are largest for rebuffering and switching, with smaller improvements for VMAF.

We compare *WideEye* with and without σ_{bv} augmentation. For each of our 11 videos, we calculate the changes in the average metric across traces, *i.e.,* rebuffer ratio (in sec per min), VMAF stability (in percentage). The results shown in Fig. 5.10 are the distribution of these improvements from adding σ_{bv} across the 11 videos. For the SLOW traces, rebuffering is reduced on average by >3 s/m for all algorithms. The improvements stem primarily from augmentation enabling finer-grained quality decisions especially at low-bitrate tracks. Even for RMPC, this compensates for the shorter lookahead. The differences are

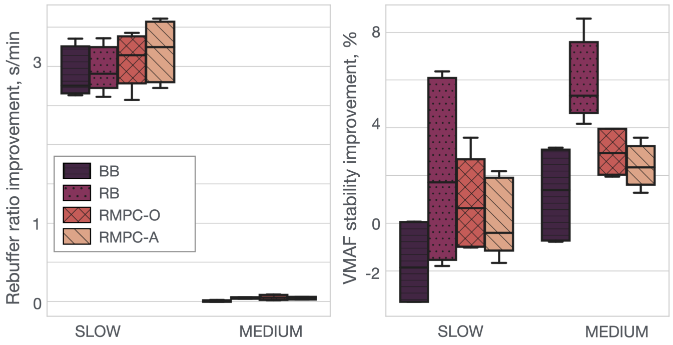


Figure 5.10: *Improvements from adding σ_{bv} to WideEye. We average metrics across traces, and show their distribution across videos. Boxes show mean and quartiles, whiskers are 5/95-percentile.*

smaller for FAST traces (as expected), which are omitted in the plot.

Rebuffering improvements for RB are more limited because having more choices sometimes enables more aggressive behavior in RB, where the estimated rate has greater chances of more closely matching an available bitrate. For the same reason, VMAF switches improve more for RB: it aggressively matches bitrate to rate estimates, and having more choices makes the fluctuations smaller.

VMAF gains are small in the aggregate, but this is a bit misleading: in many cases, augmentation improves VMAF noticeably (*e.g.*, ~10%) for parts of playback, but these 'local' gains are suppressed in the aggregate, as VMAF does not change much for most segments. (Fig. 5.11 highlights the locality of these improvements.) It is unclear to us how or if QoE functions should reward such local improvements.

The provider-side costs of σ_{bv} augmentation are small across all 4 rate adaptation algorithm, with roughly 8% overhead in bytes encoded on average across videos.

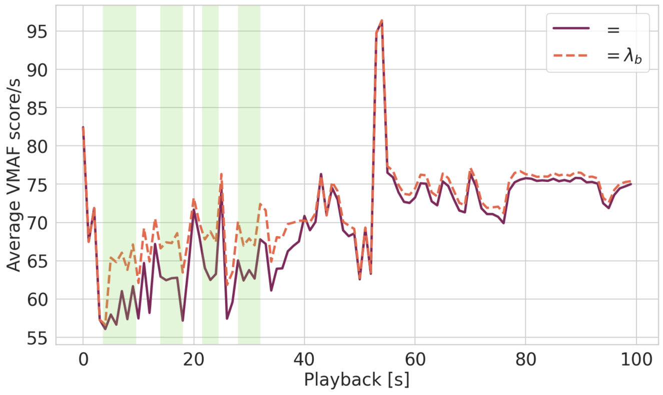


Figure 5.11: *Augmentation yields local* `VMAF` *improvements: average* `VMAF`*/sec for video* **E** *using* `RMPC`*, Constant segmentation*

Takeaway: Augmentation substantially improves rebuffering and `VMAF` stability, especially in low-bandwidth conditions, while incurring modest costs.

5.7.4 The impact of the video

How much a video's complexity varies over time greatly affects how much SEGUE benefits it. For instance, video **C** shows, on each of its tracks, very little variation in `VMAF` and bitrate. This lack of substantial temporal variation leaves little room for optimization beyond constant-length segments with fixed tracks. For video **C**, our segmentation's improvements in `VMAF` stability are smaller than on the rest of our data, and in some cases, there is even a degradation in performance (3.1% and 2.4% on average over SLOW traces with `RMPC-A` and `-O` respectively).

The other extreme is video **D**, with frequent changes across scenes. (Fig. 5.12 visually contrasts videos **C** and **D**.) For video **D**, even for

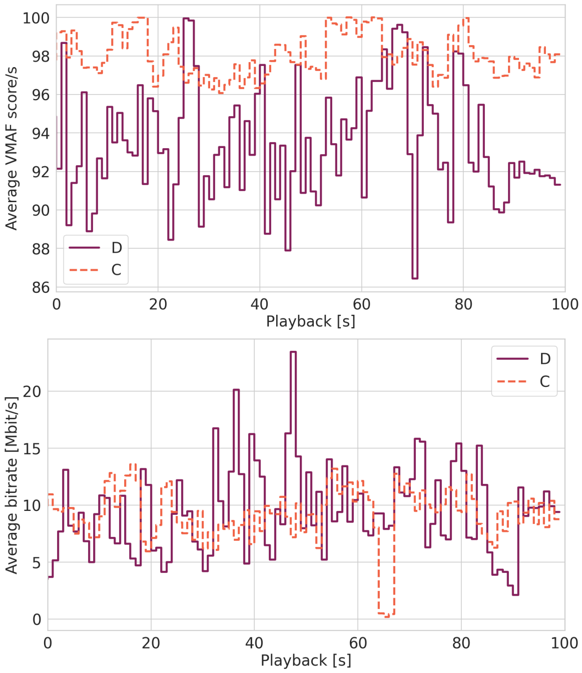


Figure 5.12: *`VMAF` and bitrate comparison between video **C** and video **D** for the highest quality track of each. `VMAF` and bitrate are averaged per second, and shown for the first* 100 *seconds of playback. Video **C** exhibits much greater stability than video **D**.*

FAST traces, *WideEye* improves `VMAF` stability by 12% on average for `RB` and `BB`.

For 3 videos in our dataset (video **B**, video **G** and video **I**), SEGUE's segmentation hurts the performance for both versions of `RMPC`. For video **B** and video **G** we have a degradation in terms of delivered `VMAF`, with average perceptual quality degradation of 3%. For video

B, this degradation also appears in VMAF stability. (Augmentation partially makes up for this deterioration.) This is due to the behavior in the startup phase: for some videos, producing small segments at the beginning greatly slows down the ramp-up of RMPC to higher bitrates. Modifying the objective function in RMPC to account for startup would likely ameliorate this issue.

For Video I, however, SEGUE with RMPC-A substantially degrades performance, with a perceptual quality loss in FAST traces of 12% and a loss in VMAF stability of 23%. This behavior is caused by a quirk of the bandwidth estimation approach (which we left untouched from prior work [MNA17b]), whereby the RTT is also incorporated to the calculation of the bandwidth estimate. The impairment occurs when, at the beginning of a video, there are one or more segments comprising as little as a few kilobytes of data, *e.g.,* a few seconds of a completely black screen or title screen. In this case, the bandwidth-dependent download time can be smaller than the RTT. Incorporating the RTT in the bandwidth estimation thus substantially underestimates bandwidth, and slows down the ramp up of video quality. A constant-length segmentation is immune to this bug, while *any* segmentation that allows smaller segments is affected by it.

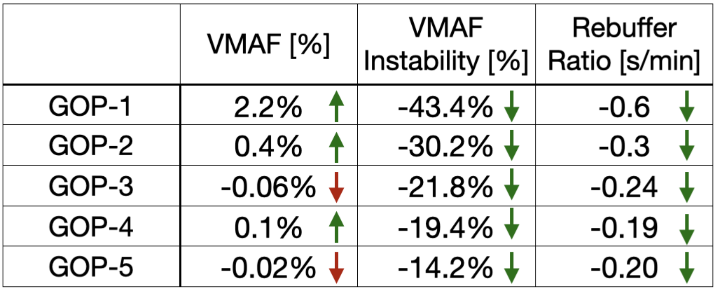
Simply separating the RTT estimate from the download time would eliminate this issue.

Takeaway: SEGUE's benefits are larger for more complex video content. This could be used to build a meta-heuristic to decide whether or not to use SEGUE for a particular video.

5.7.5 Performance comparison with related works

We now compare SEGUE against the three closest related works. First, we compare SEGUE's *WideEye* segmentation strategy to [Sch+20], in which videos are segmented and delivered following GOP boundaries. Then, we compare SEGUE's σ_{bv} augmentation strategy to CBF [Qin+19] and SIVQ [Rai+17]. Both solutions aim to remove *redundant* segments from the video representation. While CBF removes segments in order to be as close as possible to a target quality, SIVQ focuses on keeping the ones that differs enough in terms of perceptual quality.

5.7.5.1 Comparison with GOP delivery



	VMAF [%]	VMAF Instability [%]	Rebuffer Ratio [s/min]
GOP-1	2.2% ↑	-43.4% ↓	-0.6 ↓
GOP-2	0.4% ↑	-30.2% ↓	-0.3 ↓
GOP-3	-0.06% ↓	-21.8% ↓	-0.24 ↓
GOP-4	0.1% ↑	-19.4% ↓	-0.19 ↓
GOP-5	-0.02% ↓	-14.2% ↓	-0.20 ↓

Figure 5.13: *Performance improvements of SEGUE's segmentation strategy WideEye over the delivery of single GOPs, varying the GOP size.*

In order to compare SEGUE's segmentation strategy to [Sch+20], we encoded our full dataset of videos varying the maximum GOP size from 1s to 5s, with a step of 1s. We then tested the streaming performance of transmitting each GOP separately against SEGUE's *WideEye* segmentation strategy across the cross product of network traces and ABRs. Results are summarised in Fig. 5.13.

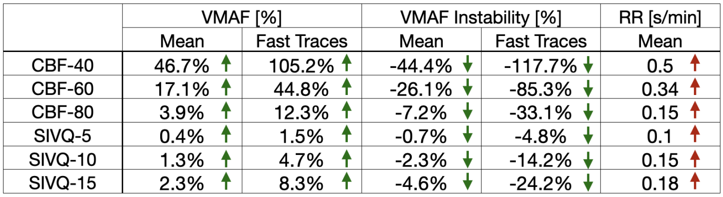
Improvements in perceptual quality are significant only in the case of GOP-1, where SEGUE delivers in average 2.2% better quality.

SEGUE consistently reduces perceptual quality fluctuations. This is an intrinsic property of SEGUE: by deciding which GOPs to pack together (or not) depending on the video flow and ABR behavior, SEGUE is able to successfully stabilize the video stream compared to a fully fragmented solution. These improvements are major against a GOP length of 1s, where SEGUE reduces instabilities by 43.4%, and become smaller, but still significant, when increasing the GOP size. VMAF instability reduction over the GOP-5 segmentation (that is, indeed, the GOP size in which SEGUE's *WideEye* is performed) is in average 14.2%.

SEGUE also substantially improves rebuffering ratio. Shorter segments, in fact, do not necessarily reduce the likelihood of rebuffering events, as they might mislead the ABR into poor buffer planning and, in general, greedier choices. Again, by tracking the video flow and ABR choices, SEGUE is able to reduce substantially the average rebuffering ratio, by 0.6 s/m in the case of GOP-1, and by around 0.2 s/m in the case of GOP-5.

Compared to GOP-5, SEGUE segmentation strategy's improvements in the linear QoE model utilised throughout this work are, in average, 4.2%. In the case of GOP-1, these improvements increase to 14%.

5.7.5.2 Comparison with CBF and SIVQ



	VMAF [%]		VMAF Instability [%]		RR [s/min]
	Mean	Fast Traces	Mean	Fast Traces	Mean
CBF-40	46.7% ↑	105.2% ↑	-44.4% ↓	-117.7% ↓	0.5 ↑
CBF-60	17.1% ↑	44.8% ↑	-26.1% ↓	-85.3% ↓	0.34 ↑
CBF-80	3.9% ↑	12.3% ↑	-7.2% ↓	-33.1% ↓	0.15 ↑
SIVQ-5	0.4% ↑	1.5% ↑	-0.7% ↓	-4.8% ↓	0.1 ↑
SIVQ-10	1.3% ↑	4.7% ↑	-2.3% ↓	-14.2% ↓	0.15 ↑
SIVQ-15	2.3% ↑	8.3% ↑	-4.6% ↓	-24.2% ↓	0.18 ↑

Figure 5.14: *Performance improvements and degradation of SEGUE's augmentation strategy σ_{bv} compared to CBF [Qin+19] and SIVQ [Rai+17]. Rebuffering ratio comparison for FAST traces is neglibile, and thus it has been omitted.*

We compare SEGUE's augmentation strategy σ_{bv} to CBF [Qin+19] and SIVQ [Rai+17]. All the approaches are applied to SEGUE's *WideEye* segmentation. We offer to CBF and SIVQ **all** the available options, in other words the ones that SEGUE uses as standard (and non removable) and the augmented ones. We test CBF under three

VMAF thresholds (40, 60, 80). We also modified the SIVQ algorithm to work with VMAF rather than PSNR, and we test it for three different thresholds (5, 10, 15).

In Fig. 5.14 we show the comparison with SEGUE's σ_{bv} aggregated across our cross product of videos, ABRs and traces. Both SIVQ and CBF substantially reduce rebuffering compared to SEGUE. This is expected, as both approaches (CBF more aggressively than SIVQ) almost entirely remove the 1440p track, and substantially cut down the 1080p track. This forces all the ABRs to perform safer choices, as the highest quality options are not available.

The removal of higher quality tracks has the drawback of reducing the delivered quality, in particular for FAST traces. This is an expected behaviour of both CBF and SIVQ, as they optimize for bandwidth and storage savings. The severity of this reduction depends on the selected threshold, with SIVQ-5 being the least affected: 0.4% degradation in average, with 1.5% degradation in FAST traces and 4.3% in the 95-th percentile of the best traces.

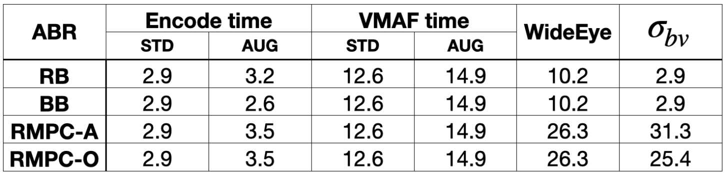
Compared to SIVQ-5, SEGUE improves the VMAF stability in FAST traces by 4.8%, an improvement that is coherent with the one experienced by introducing augmentation. For SLOW and MEDIUM traces, improvements in stability are not substantial. In any case, as explained in section §5.6, despite our trace set being balanced, 91% of traces in the analyzed Puffer set are classified as FAST. SEGUE's σ_{bv} improves substantially in challenging scenarios without affecting the user experience in the most common setting.

Compared to CBF-40 and SIVQ-15, SEGUE's improvements in average on FAST traces for the linear QoE formulation utilised in this work are of 45% and 5.3% respectively. These improvements are still substantial if compared to CBF-80 (5%), and become small if compared to SIVQ-5 (>1%), due to the low weight on VMAF instability in our linear QoE formulation. VMAF instability is indeed the optimisation metric that SEGUE improves the most.

Last but not the least, SEGUE can be combined with both CBF and SIVQ. In case of CBF, one could pre-filter both standard and non standard options for a specific quality setting, and then run the

Segue's optimization. Similarly, for SIVQ we could just present to Segue optimizer the video segments that are sufficiently different in terms of perceptual quality performance. However, since adding CBF or SIVQ to Segue would lead to a worse outcome on FAST traces, we did not include such a combination in our evaluation.

5.7.6 Segue's computational cost



ABR	Encode time		VMAF time		WideEye	σ_{bv}
	STD	AUG	STD	AUG		
RB	2.9	3.2	12.6	14.9	10.2	2.9
BB	2.9	2.6	12.6	14.9	10.2	2.9
RMPC-A	2.9	3.5	12.6	14.9	26.3	31.3
RMPC-O	2.9	3.5	12.6	14.9	26.3	25.4

Figure 5.15: *Benchmarking of Segue's performance for video **B** as a ratio between the computational time and the video length. Segue's brute force exploration time is heavily affected by the ABR algorithm efficiency.*

We benchmarked Segue's computational performance for video **B**, as it strikes a good tradeoff between bitrate variability and keyframes frequency. The benchmark runs on an AMD Ryzen 9 3900x 12-Core processor and Ubuntu 20.04.3 LTS. Results are presented in Fig. 5.15 as a fraction of the total computation time and the video length. Segue's performance highly depends on the ABR algorithm's efficiency. Segue's computation time using fast ABR algorithms like rate or buffer based is comparable to the VMAF computation time. Using slow ABRs, like both version of R-MPC, takes considerably more time due to the state space exploration (a problem that has been tackled by the authors in [Yin+15], and that lead to the formulation of the more lightweight version Fast-MPC).

Compared to the H.264 encoding time, Segue's segmentation takes 3.5x more time with the fastest ABR. Nevertheless, Segue's segmentation times are comparable to the computational time needed by more recent codecs, like VP9 and AV1, that take significantly more time compared to H.264 (5x and 10x respectively [Sim21]), while SEGUE's approach and costs are independent on the codec of choice.

SEGUE's current release is not optimized for runtime and written in Python3 using the multiprocessing module. This module uses expensive process based parallelism. A reimplementation in an unmanaged language with better multithreading support (like C++ or Rust) would likely offer at least an order of magnitude improvement in compute times, as for example Numba [LPS15] discusses. Also, given the substantial amount of time that is spent on VMAF calculations, SEGUE could be extended to either work with computationally cheaper quality metrics (like PSNR or SSIM), or approaches like the one in [KAB21] could be used for VMAF rate distortion curves prediction.

5.7.7 *dash.js* player implementation

5.7.7.1 Using SEGUE in DASH

To confirm that the simulated results are comparable to real world experiments we run a smaller number of real-time experiments on the DASH JavaScript reference player.

Variable length segments are natively supported by DASH and therefore `dash.js` [DAS12] through the use of a *SegmentTimeline* block in the MPEG-DASH Media Presentation Description (MPD). The *SegmentTimeline* is generated alongside the DASH-compatible media segments using the *sigcues* filter from GPAC [Le 20], which uses the pre-segmented SEGUE chunks and creates the corresponding DASH-playable segments (dashing). For augmented tracks, unavailable chunks are replaced by the corresponding chunks of the standard track during this process, with all the placeholder segments getting removed once dashing is complete.

Per-segment bitrate information is made available to the player using an additional JSON file containing detailed information about all segments. This file is generated as part of the preprocessing step and is based on the simulator input data. During the startup phase of the player, this file is downloaded from a location specified within the MPD.

The AbrController of dash.js has been adapted to provide per-segment bitrate information supplied by the additional file instead of the average bitrates reported in the MPD. It does so by updating the bitrate (of the next segment) of all tracks to the corresponding values whenever the bitrate list is assembled.

The same approach is used to introduce basic augmentation support: Since tracks cannot be removed or added unless the player switches DASH-Periods - which was not a viable option in this case - an unavailable track receives a bitrate of 1 Tbit/s instead. An augmentation-oblivious ABR should not choose a track with such high a bitrate under normal circumstances, while an augmentation-aware ABR can check for this (constant) value to see whether an augmented track is available or not.

The additional information contained in the JSON file can be accessed by an ABR through the AbrController if needed, which is used by non-myopic schemes like RMPC to get information about future segments. The three ABRs, RB, BB and RMPC-O, have been implemented in JavaScript based on their counterparts used in the simulation.

Three modifications to the default behaviour of dash.js were made for our experiments: First, only our custom ABR rule is active, instead of the combination of rules used normally. Second, the start of video playback is intercepted and triggered only once at least 10 seconds of video are in the buffer. Finally, the replacing of already downloaded but not yet played segments ('fast-switching') has been disabled.

5.7.7.2 Experiment setup

The tests run locally on Ubuntu 18.04.5 LTS using Apache webserver (version 2.4.29) to provide the website and video segments. Selenium WebDriver [Con] launches Google Chrome (version 87) in headless mode to load the page and play the video.

This process is run from within a Mahimahi shell [Net+15] to emulate different network conditions based on network traces. Metrics from `dash.js` are output through JavaScript log messages, which are retrieved and processed to generate the results.

A set of 100 traces is used to run experiments on video **A** in real-time with `dash.js`, the results of which are then used to compare the simulation results on the same set of traces to. All three `ABR`s were run on three configurations: constant-length segments without augmentation, the corresponding `ABR`-specific *WideEye* segmentations without augmentation, and the `ABR`-specific segmentation with σ_{bv} augmentation. The quality metrics are then aggregated using the `VMAF` 4K model.

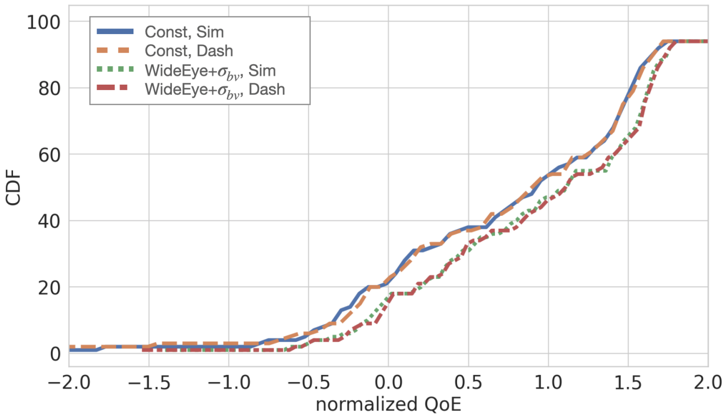


Figure 5.16: *Comparison of resulting* `QoE` *between simulation and* `dash.js` *implementation on the same set of 100 traces for video* ***A*** *and RB. The* `QoE` *is normalized by the mean of constant length.*

5.7.7.3 Results

Overall, results in the QoE distribution are similar. An example of such a comparison is shown in Fig. 5.16. The figure shows the CDF of the distribution of the QoE for video **A** and RB, evaluated using VMAF 4K. Lines are plotted for the video without Segue (using constant length segments), and with Segue (WideEye+σ_{bv}), and each of those once for simulation and execution in DASH.

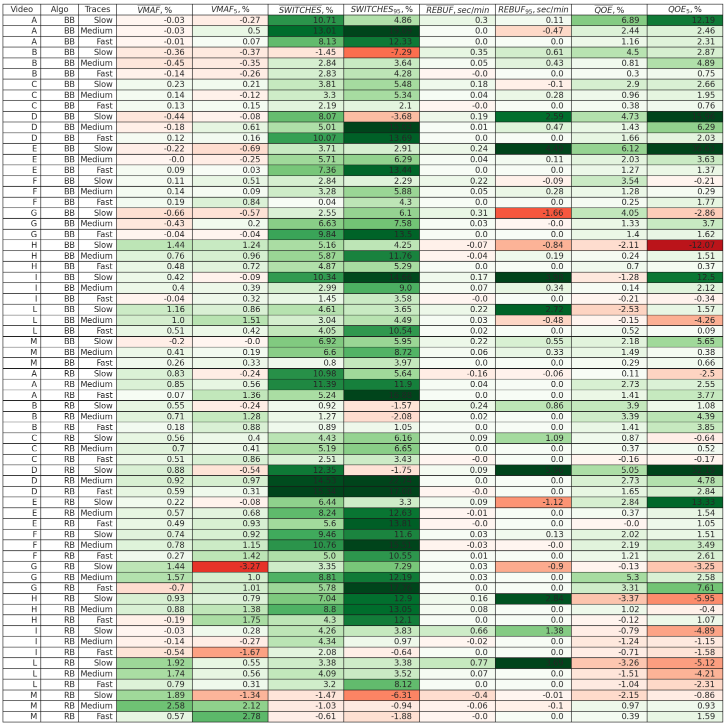
Similar results are obtained for the ABRs BB and RMPC-O. In particular, the improvements in performance in the mean of WideEye+σ_{bv} with respect to constant are of 5.3%, 6.7% and 5.6% in simulation for BB, RB and RMPC, while in DASH we obtained 5.1%, 7.3% and 5.1%. Improvements have been calculated following the formula expressed in §5.7.8.

5.7.8 Summary of results

We evaluated Segue across 4 adaptation algorithms, 11 videos, and 3 trace buckets. Segue typically maintains average VMAF, while reducing switching and tail rebuffering, at the expense of reduced VMAF for a small fraction of chunks. This is a highly favorable tradeoff for the QoE function.

We calculate QoE improvements as: $100 \cdot \frac{Q_{\text{Segue}} - Q_{Constant}}{Q_{\max}}$, where $Q_{\max}$ is the maximum achievable QoE. Comparing Q_{Segue} and $Q_{Constant}$ directly would only show larger numbers. Across our result matrix, Segue's mean QoE improvement is 8.6%, with 36.5% improvement in the 5th-percentile. When limited to SLOW traces, these numbers are 22.1% and 111% respectively. Full tables of results are in Fig. 5.17 5.18 and Fig. 5.19 5.20.

For context on Segue's QoE improvements, we can compare them to those for algorithmic improvements in rate adaptation. Across our traces, QoE for *Constant* improves by less than 2% when using R-MPC instead of BB. (This is in line with experiments in the recent Puffer work [Yan+20], providing validation for our evaluation.) Our improvements are larger than what Facebook measured [Mao+19] in testing reinforcement learning adaptation, where under 6% improvements



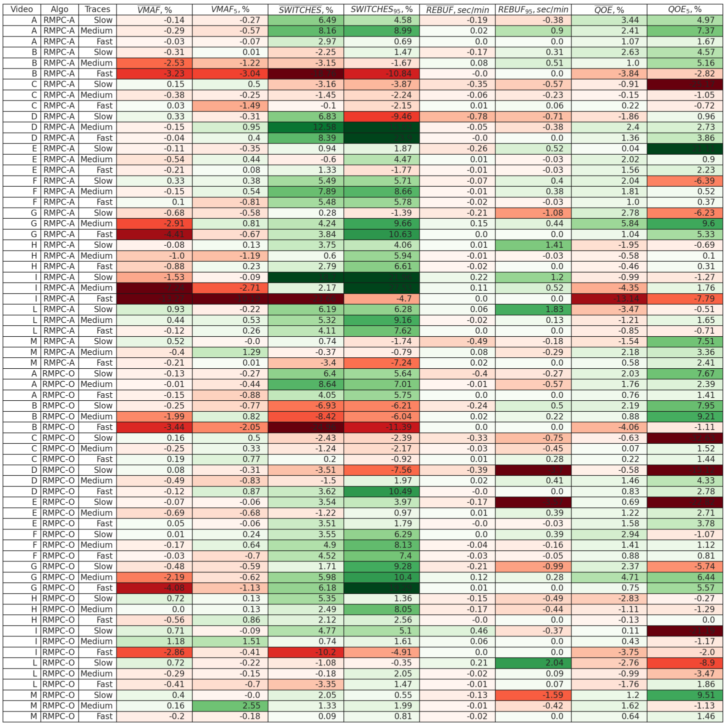
Video	Algo	Traces	$\overline{VMAF}$, %	$VMAF_5$, %	$\overline{SWITCHES}$, %	$SWITCHES_{95}$, %	$\overline{REBUF}$, sec/min	$REBUF_{95}$, sec/min	$\overline{QOE}$, %	QOE_5, %
A	BB	Slow	-0.03	-0.27	10.71	4.86	0.3	0.11	6.89	12.19
A	BB	Medium	-0.03	0.5	13.01	19.09	0.0	-0.47	2.44	2.46
A	BB	Fast	-0.01	0.07	8.13	12.33	0.0	0.0	1.16	2.31
B	BB	Slow	-0.36	-0.37	-1.45	-7.29	0.35	0.61	4.5	2.87
B	BB	Medium	-0.45	-0.35	2.84	3.64	0.05	0.43	0.81	4.89
B	BB	Fast	-0.14	-0.26	2.83	4.28	-0.0	0.0	0.3	0.75
C	BB	Slow	0.23	0.21	3.81	5.48	0.18	-0.1	2.9	2.66
C	BB	Medium	0.14	-0.12	3.3	5.34	0.04	0.28	0.96	1.95
C	BB	Fast	0.13	0.15	2.19	2.1	-0.0	0.0	0.38	0.76
D	BB	Slow	-0.44	-0.08	8.07	-3.68	0.19	2.59	4.73	15.96
D	BB	Medium	-0.18	0.61	5.01	15.03	0.01	0.47	1.43	6.29
D	BB	Fast	0.12	0.16	10.07	13.69	0.0	0.0	1.66	2.03
E	BB	Slow	-0.22	-0.69	3.71	2.91	0.24	4.49	6.12	36.63
E	BB	Medium	-0.0	-0.25	5.71	6.29	0.04	0.11	2.03	3.63
E	BB	Fast	0.09	0.03	7.36	13.44	0.0	0.0	1.27	1.37
F	BB	Slow	0.11	0.51	2.84	2.29	0.22	-0.09	3.54	-0.21
F	BB	Medium	0.14	0.09	3.28	5.88	0.05	0.28	1.28	0.29
F	BB	Fast	0.19	0.84	0.04	4.3	0.0	0.0	0.25	1.77
G	BB	Slow	-0.66	-0.57	2.55	6.1	0.31	-1.66	4.05	-2.86
G	BB	Medium	-0.43	0.2	6.63	7.58	0.03	-0.0	1.33	3.7
G	BB	Fast	-0.04	-0.04	9.84	13.5	0.0	0.0	1.4	1.62
H	BB	Slow	1.44	1.24	5.16	4.25	-0.07	-0.84	-2.11	-12.07
H	BB	Medium	0.76	0.96	5.87	11.76	-0.04	0.19	0.24	1.51
H	BB	Fast	0.48	0.72	4.87	5.29	0.0	0.0	0.7	0.37
I	BB	Slow	0.42	-0.09	10.34	14.66	0.17	3.98	-1.28	12.5
I	BB	Medium	0.4	0.39	2.99	9.0	0.07	0.34	0.14	2.12
I	BB	Fast	-0.04	0.32	1.45	3.58	-0.0	0.0	-0.21	-0.34
L	BB	Slow	1.16	0.86	4.61	3.65	0.22	2.72	-2.53	1.57
L	BB	Medium	1.0	1.51	3.04	4.49	0.03	-0.48	-0.15	-4.26
L	BB	Fast	0.51	0.42	4.05	10.54	0.02	0.0	0.52	0.09
M	BB	Slow	-0.2	-0.0	6.92	5.95	0.22	0.55	2.18	5.65
M	BB	Medium	0.41	0.19	6.6	8.72	0.06	0.33	1.49	0.38
M	BB	Fast	0.26	0.33	0.8	3.97	0.0	0.0	0.29	0.66
A	RB	Slow	0.83	-0.24	10.98	5.64	-0.16	-0.06	0.11	-2.5
A	RB	Medium	0.85	0.56	11.39	11.9	0.04	0.0	2.73	2.55
A	RB	Fast	0.07	1.36	5.24	16.88	0.0	0.0	1.41	3.77
B	RB	Slow	0.55	-0.24	0.92	-1.57	0.24	0.86	3.9	1.08
B	RB	Medium	0.71	1.28	1.27	-2.08	0.02	0.0	3.39	4.39
B	RB	Fast	0.18	0.88	0.89	1.05	0.0	0.0	1.41	3.85
C	RB	Slow	0.56	0.4	4.43	6.16	0.09	1.09	0.87	-0.64
C	RB	Medium	0.7	0.41	5.19	6.65	0.0	0.0	0.37	0.52
C	RB	Fast	0.51	0.86	2.51	3.43	-0.0	0.0	-0.16	-0.17
D	RB	Slow	0.88	-0.54	12.35	-1.75	0.09	5.96	5.05	52.19
D	RB	Medium	0.92	0.97	14.53	22.74	0.0	0.0	2.73	4.78
D	RB	Fast	0.59	0.31	15.34	17.08	-0.0	0.0	1.65	2.84
E	RB	Slow	0.22	-0.08	6.44	3.3	0.09	-1.12	2.84	13.33
E	RB	Medium	0.57	0.68	8.24	12.63	-0.01	0.0	0.37	1.54
E	RB	Fast	0.49	0.93	5.6	13.81	-0.0	0.0	-0.0	1.05
F	RB	Slow	0.74	0.92	9.46	11.6	0.03	0.13	2.02	1.51
F	RB	Medium	0.78	1.15	10.76	15.51	-0.03	-0.0	2.19	3.49
F	RB	Fast	0.27	1.42	5.0	10.55	0.01	0.0	1.21	2.61
G	RB	Slow	1.44	-3.27	3.35	7.29	0.03	-0.9	-0.13	-3.25
G	RB	Medium	1.57	1.0	8.81	12.19	0.03	0.0	5.3	2.58
G	RB	Fast	-0.7	1.01	5.78	17.76	0.0	0.0	3.31	7.61
H	RB	Slow	0.93	0.79	7.04	12.9	0.16	2.84	-3.37	-5.95
H	RB	Medium	0.88	1.38	8.8	13.05	0.08	0.0	1.02	-0.4
H	RB	Fast	-0.19	1.75	4.3	12.1	0.0	0.0	-0.12	1.07
I	RB	Slow	-0.03	0.28	4.26	3.83	0.66	1.38	-0.79	-4.89
I	RB	Medium	-0.14	-0.27	4.34	0.97	-0.02	0.0	-1.24	-1.15
I	RB	Fast	-0.54	-1.67	2.08	-0.64	0.0	0.0	-0.71	-1.58
L	RB	Slow	1.92	0.55	3.38	3.38	0.77	3.69	-3.26	-5.12
L	RB	Medium	1.74	0.56	4.09	3.52	0.07	0.0	-1.51	-4.21
L	RB	Fast	0.79	0.31	3.2	8.12	0.0	0.0	-1.04	-2.31
M	RB	Slow	1.89	-1.34	-1.47	-6.31	-0.4	-0.01	-2.15	-0.86
M	RB	Medium	2.58	2.12	-1.03	-0.94	-0.06	-0.1	0.97	0.93
M	RB	Fast	0.57	2.78	-0.61	-1.88	-0.0	0.0	0.39	1.59

Figure 5.17: *SEGUE's results with WideEye segmentation and no augmentation for BB and RB, compared to Constant.*

for traces with sub-500 Kbps bandwidth (as much 3× slower than our SLOW set) are reported as "substantial" for Facebook.

5.8 Discussion and Future Work

With SEGUE, we have only begun exploring new opportunities that arise from accounting for the temporal variations in video content

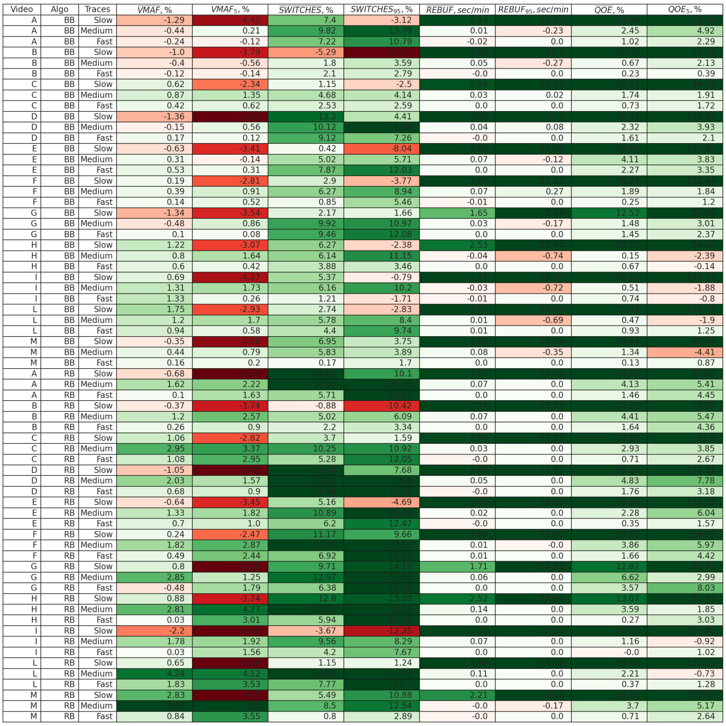


Video	Algo	Traces	$\overline{VMAF}$, %	$VMAF_5$, %	$\overline{SWITCHES}$, %	$SWITCHES_{95}$, %	$\overline{REBUF}$, sec/min	$REBUF_{95}$, sec/min	$\overline{QOE}$, %	QOE_5, %
A	RMPC-A	Slow	-0.14	-0.27	6.49	4.58	-0.19	-0.38	3.44	4.97
A	RMPC-A	Medium	-0.29	-0.57	8.16	8.99	0.02	0.9	2.41	7.37
A	RMPC-A	Fast	-0.03	-0.07	2.97	0.69	0.0	0.0	1.07	1.67
B	RMPC-A	Slow	-0.31	0.01	-2.25	1.47	-0.17	0.31	2.63	4.57
B	RMPC-A	Medium	-2.53	-1.22	-3.15	-1.67	0.08	0.51	1.0	5.16
B	RMPC-A	Fast	-3.23	-3.04	-18.76	-10.84	-0.0	0.0	-3.84	-2.82
C	RMPC-A	Slow	0.15	0.5	-3.16	-3.87	-0.35	-0.57	-0.91	-25.17
C	RMPC-A	Medium	-0.38	-0.25	-1.45	-2.24	-0.06	-0.23	-0.15	-1.05
C	RMPC-A	Fast	0.03	-1.49	-0.1	-2.15	0.01	0.06	0.22	-0.72
D	RMPC-A	Slow	0.33	-0.31	6.83	-9.46	-0.78	-0.71	-1.86	0.96
D	RMPC-A	Medium	-0.15	0.95	12.58	15.03	-0.05	-0.38	2.4	2.73
D	RMPC-A	Fast	-0.04	0.4	8.39	23.9	-0.0	0.0	1.36	3.86
E	RMPC-A	Slow	-0.11	-0.35	0.94	1.87	-0.26	0.52	0.04	21.11
E	RMPC-A	Medium	-0.54	0.44	-0.6	4.47	0.01	-0.03	2.02	0.9
E	RMPC-A	Fast	-0.21	0.08	1.33	-1.77	-0.01	-0.03	1.56	2.23
F	RMPC-A	Slow	0.33	0.38	5.49	5.71	-0.07	0.4	2.04	-6.39
F	RMPC-A	Medium	-0.15	0.54	7.89	8.66	-0.01	0.38	1.81	0.52
F	RMPC-A	Fast	0.1	-0.81	5.48	5.78	-0.02	-0.03	1.0	0.37
G	RMPC-A	Slow	-0.68	-0.58	0.28	-1.39	-0.21	-1.08	2.78	-6.23
G	RMPC-A	Medium	-2.91	0.81	4.24	9.66	0.15	0.44	5.84	9.6
G	RMPC-A	Fast	-4.41	-0.67	3.84	10.63	0.0	0.0	1.04	5.33
H	RMPC-A	Slow	-0.08	0.13	3.75	4.06	0.01	1.41	-1.95	-0.69
H	RMPC-A	Medium	-1.0	-1.19	0.6	5.94	-0.01	-0.03	-0.58	0.1
H	RMPC-A	Fast	-0.88	0.23	2.79	6.61	-0.02	0.0	-0.46	0.31
I	RMPC-A	Slow	-1.53	-0.09	18.7	21.96	0.22	1.2	-0.99	-1.27
I	RMPC-A	Medium	-7.24	-2.71	2.17	27.53	0.11	0.52	-4.35	1.76
I	RMPC-A	Fast	-13.27	-10.19	-23.66	-4.7	0.0	0.0	-13.14	-7.79
L	RMPC-A	Slow	0.93	-0.22	6.19	6.28	0.06	1.83	-3.47	-0.51
L	RMPC-A	Medium	0.44	0.53	5.32	9.16	-0.02	0.13	-1.21	1.65
L	RMPC-A	Fast	-0.12	0.26	4.11	7.62	0.0	0.0	-0.85	-0.71
M	RMPC-A	Slow	0.52	-0.0	0.74	-1.74	-0.49	-0.18	-1.54	7.51
M	RMPC-A	Medium	-0.4	1.29	-0.37	-0.79	0.08	-0.29	2.18	3.36
M	RMPC-A	Fast	-0.21	0.01	-3.4	-7.24	0.02	0.0	0.58	2.41
A	RMPC-O	Slow	-0.13	-0.27	6.4	5.64	-0.4	-0.27	2.03	7.67
A	RMPC-O	Medium	-0.01	-0.44	8.64	7.01	-0.01	-0.57	1.76	2.39
A	RMPC-O	Fast	-0.15	-0.88	4.05	5.75	0.0	0.0	0.76	1.41
B	RMPC-O	Slow	-0.25	-0.77	-6.93	-6.21	-0.24	0.5	2.19	7.95
B	RMPC-O	Medium	-1.99	0.82	-8.42	-6.04	0.02	0.22	0.88	9.21
B	RMPC-O	Fast	-3.44	-2.05	-24.96	-11.39	0.0	0.0	-4.06	-1.11
C	RMPC-O	Slow	0.16	0.5	-2.43	-2.39	-0.33	-0.75	-0.63	-32.63
C	RMPC-O	Medium	-0.25	0.33	-1.24	-2.17	-0.03	-0.45	0.07	1.52
C	RMPC-O	Fast	0.19	0.77	0.2	-0.92	0.01	0.28	0.22	1.44
D	RMPC-O	Slow	0.08	-0.31	-3.51	-7.56	-0.39	-3.7	-0.58	-15.12
D	RMPC-O	Medium	-0.49	-0.83	-1.5	1.97	0.02	0.41	1.46	4.33
D	RMPC-O	Fast	-0.12	0.87	3.62	10.49	-0.0	0.0	0.83	2.78
E	RMPC-O	Slow	-0.07	-0.06	3.54	3.97	-0.17	-3.57	0.69	-22.82
E	RMPC-O	Medium	-0.69	-0.68	-1.22	0.97	0.01	0.39	1.22	2.71
E	RMPC-O	Fast	0.05	-0.06	3.51	1.79	-0.0	-0.03	1.58	3.78
F	RMPC-O	Slow	0.01	0.24	3.55	6.29	0.0	0.39	2.94	-1.07
F	RMPC-O	Medium	-0.17	0.64	4.9	8.13	-0.04	-0.16	1.41	1.12
F	RMPC-O	Fast	-0.03	-0.7	4.52	7.4	-0.03	-0.05	0.88	0.81
G	RMPC-O	Slow	-0.48	-0.59	1.71	9.28	-0.21	-0.99	2.37	-5.74
G	RMPC-O	Medium	-2.19	-0.62	5.98	10.4	0.12	0.28	4.71	6.44
G	RMPC-O	Fast	-4.08	-1.13	6.18	19.29	0.01	0.0	0.75	5.57
H	RMPC-O	Slow	0.72	0.13	5.35	1.36	-0.15	-0.49	-2.83	-0.27
H	RMPC-O	Medium	0.0	0.13	2.49	8.05	-0.17	-0.44	-1.11	-1.29
H	RMPC-O	Fast	-0.56	0.86	2.12	2.56	-0.0	0.0	-0.13	0.0
I	RMPC-O	Slow	0.71	-0.09	4.77	5.1	0.46	-0.37	0.11	-21.19
I	RMPC-O	Medium	1.18	1.51	0.74	1.61	0.06	0.0	0.43	-1.17
I	RMPC-O	Fast	-2.86	-0.41	-10.2	-4.91	0.0	0.0	-3.75	-2.0
L	RMPC-O	Slow	0.72	-0.22	-1.08	-0.35	0.21	2.04	-2.76	-8.9
L	RMPC-O	Medium	-0.29	-0.15	-0.18	2.05	-0.02	0.09	-0.99	-3.47
L	RMPC-O	Fast	-0.41	-0.7	-3.35	1.47	-0.01	0.07	-1.76	1.86
M	RMPC-O	Slow	0.4	-0.0	2.05	0.55	-0.13	-1.59	1.2	9.51
M	RMPC-O	Medium	0.16	2.55	1.33	1.99	-0.01	-0.42	1.62	-1.13
M	RMPC-O	Fast	-0.2	-0.18	0.09	0.81	-0.02	0.0	0.64	1.46

Figure 5.18: *SEGUE's results with WideEye segmentation and no augmentation for RMPC-A and RMPC-O, compared to Constant.*

and their interactions with online adaptation.

Algorithmic work: Much like for rate adaptation, where new algorithms continue to be devised, we fully expect SEGUE to set off a new thread on how best to optimize chunking. A particularly promising opportunity for segmentation lies in doing chunking online, whereby the client could adaptively request video in terms of keyframe boundaries,

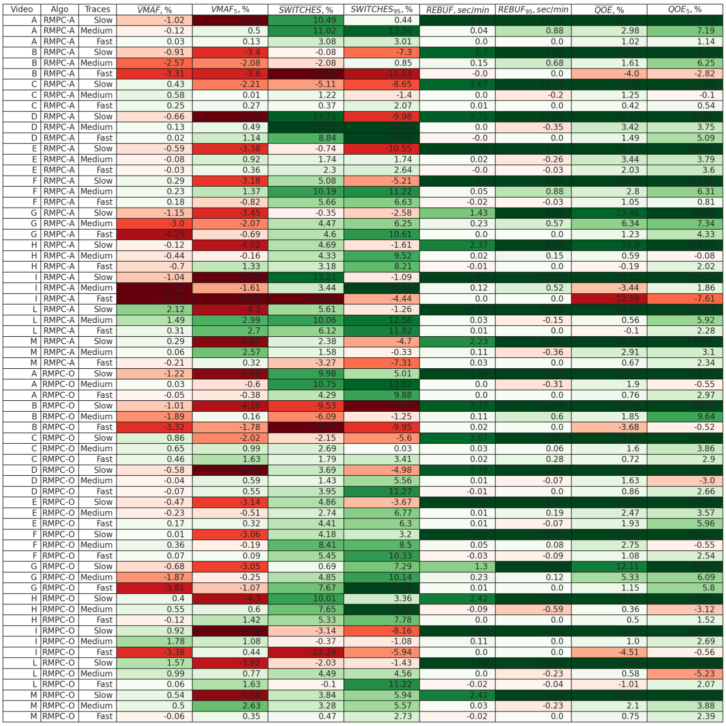


Video	Algo	Traces	$\overline{VMAF}$, %	$VMAF_5$, %	$\overline{SWITCHES}$, %	$SWITCHES_{95}$, %	$\overline{REBUF}$, sec/min	$REBUF_{95}$, sec/min	$\overline{QOE}$, %	QOE_5, %
A	BB	Slow	-1.29	-4.45	7.4	-3.12	2.93	13.03	25.86	78.58
A	BB	Medium	-0.44	0.21	9.82	13.79	0.01	-0.23	2.45	4.92
A	BB	Fast	-0.24	-0.12	7.22	10.79	-0.02	0.0	1.02	2.29
B	BB	Slow	-1.0	-3.79	-5.29	-19.87	3.03	10.7	21.36	73.45
B	BB	Medium	-0.4	-0.56	1.8	3.59	0.05	-0.27	0.67	2.13
B	BB	Fast	-0.12	-0.14	2.1	2.79	-0.0	0.0	0.23	0.39
C	BB	Slow	0.62	-2.34	1.15	-2.5	2.88	13.65	21.31	94.39
C	BB	Medium	0.87	1.35	4.68	4.14	0.03	0.02	1.74	1.91
C	BB	Fast	0.42	0.62	2.53	2.59	0.0	0.0	0.73	1.72
D	BB	Slow	-1.36	-5.01	13.2	4.41	3.35	21.7	26.43	146.64
D	BB	Medium	-0.15	0.56	10.12	14.99	0.04	0.08	2.32	3.93
D	BB	Fast	0.17	0.12	9.12	7.26	-0.0	0.0	1.61	2.1
E	BB	Slow	-0.63	-3.41	0.42	-8.04	3.59	17.08	34.34	131.84
E	BB	Medium	0.31	-0.14	5.02	5.71	0.07	-0.12	4.11	3.83
E	BB	Fast	0.53	0.31	7.87	12.03	0.0	0.0	2.27	3.35
F	BB	Slow	0.19	-2.81	2.9	-3.77	3.63	17.32	26.33	112.64
F	BB	Medium	0.39	0.91	6.27	8.94	0.07	0.27	1.89	1.84
F	BB	Fast	0.14	0.52	0.85	5.46	-0.01	0.0	0.25	1.2
G	BB	Slow	-1.34	-3.54	2.17	1.66	1.65	6.04	12.53	51.69
G	BB	Medium	-0.48	0.86	9.92	10.97	0.03	-0.17	1.48	3.01
G	BB	Fast	0.1	0.08	9.46	12.08	0.0	0.0	1.45	2.37
H	BB	Slow	1.22	-3.07	6.27	-2.38	2.53	13.94	15.13	69.42
H	BB	Medium	0.8	1.64	6.14	11.15	-0.04	-0.74	0.15	-2.39
H	BB	Fast	0.6	0.42	3.88	3.46	0.0	0.0	0.67	-0.14
I	BB	Slow	0.69	-4.27	5.37	-0.79	3.17	20.47	18.25	142.77
I	BB	Medium	1.31	1.73	6.16	10.2	-0.03	-0.72	0.51	-1.88
I	BB	Fast	1.33	0.26	1.21	-1.71	-0.01	0.0	0.74	-0.8
L	BB	Slow	1.75	-2.93	2.74	-2.83	3.61	17.29	21.31	113.72
L	BB	Medium	1.2	1.7	5.78	8.4	0.01	-0.69	0.47	-1.9
L	BB	Fast	0.94	0.58	4.4	9.74	0.01	0.0	0.93	1.25
M	BB	Slow	-0.35	-4.58	6.95	3.75	2.98	16.99	20.44	89.54
M	BB	Medium	0.44	0.79	5.83	3.89	0.08	-0.35	1.34	-4.41
M	BB	Fast	0.16	0.2	0.17	1.7	0.0	0.0	0.13	0.87
A	RB	Slow	-0.68	-5.23	16.81	10.1	2.97	14.79	24.93	97.63
A	RB	Medium	1.62	2.22	16.72	20.23	0.07	0.0	4.13	5.41
A	RB	Fast	0.1	1.63	5.71	21.61	0.0	0.0	1.46	4.45
B	RB	Slow	-0.37	-3.74	-0.88	-10.42	3.14	13.22	22.29	77.37
B	RB	Medium	1.2	2.57	5.02	6.09	0.07	0.0	4.41	5.47
B	RB	Fast	0.26	0.9	2.2	3.34	0.0	0.0	1.64	4.36
C	RB	Slow	1.06	-2.82	3.7	1.59	2.94	17.22	22.83	92.6
C	RB	Medium	2.95	3.37	10.25	10.92	0.03	0.0	2.93	3.85
C	RB	Fast	1.08	2.95	5.28	12.05	-0.0	0.0	0.71	2.67
D	RB	Slow	-1.05	-6.15	19.44	7.68	2.98	17.31	28.29	146.31
D	RB	Medium	2.03	1.57	21.13	26.8	0.05	0.0	4.83	7.78
D	RB	Fast	0.68	0.9	15.96	17.82	-0.0	0.0	1.76	3.18
E	RB	Slow	-0.64	-3.45	5.16	-4.69	3.45	16.93	33.35	155.58
E	RB	Medium	1.33	1.82	10.89	20.48	0.02	0.0	2.28	6.04
E	RB	Fast	0.7	1.0	6.2	12.47	-0.0	0.0	0.35	1.57
F	RB	Slow	0.24	-2.47	11.17	9.66	3.08	15.85	22.25	104.99
F	RB	Medium	1.82	2.87	16.35	22.59	0.01	-0.0	3.86	5.97
F	RB	Fast	0.49	2.44	6.92	16.17	0.01	0.0	1.66	4.42
G	RB	Slow	0.8	-6.36	9.71	14.19	1.71	14.91	12.82	88.41
G	RB	Medium	2.85	1.25	12.97	17.65	0.06	0.0	6.62	2.99
G	RB	Fast	-0.48	1.79	6.38	19.28	0.0	0.0	3.57	8.03
H	RB	Slow	0.88	-3.74	12.8	13.89	2.52	17.64	13.07	91.43
H	RB	Medium	2.81	4.27	17.37	19.15	0.14	0.0	3.59	1.85
H	RB	Fast	0.03	3.01	5.94	22.25	0.0	0.0	0.27	3.03
I	RB	Slow	-2.2	-5.45	-3.67	-12.25	5.32	31.26	28.02	193.91
I	RB	Medium	1.78	1.92	9.56	8.29	0.07	0.0	1.16	-0.92
I	RB	Fast	0.03	1.56	4.2	7.67	0.0	0.0	-0.0	1.02
L	RB	Slow	0.65	-5.14	1.15	1.24	4.48	19.18	29.04	133.03
L	RB	Medium	4.24	4.12	14.89	14.93	0.11	0.0	2.21	-0.73
L	RB	Fast	1.83	3.53	7.77	21.7	0.0	0.0	0.37	1.28
M	RB	Slow	2.83	-6.5	5.49	10.88	2.21	16.4	16.47	107.92
M	RB	Medium	5.02	6.0	8.5	12.54	-0.0	-0.17	3.7	5.17
M	RB	Fast	0.84	3.55	0.8	2.89	-0.0	0.0	0.71	2.64

Figure 5.19: *SEGUE's results with WideEye segmentation and σ_{bv} augmentation for BB and RB, compared to Constant.*

instead of being restricted to a particular offline chunking scheme. This approach can adapt chunking to both video content and network variations, without needing real-time reencoding.

Co-design of encoding and adaptation: While most ABR work treats video as an uncontrolled input and focuses on adaptation, we take the opposite perspective, treating rate adaptation as a given, and exploring how to modify video chunking. This obviously raises the question of



Video	Algo	Traces	$\overline{VMAF}$, %	$VMAF_5$, %	$\overline{SWITCHES}$, %	$SWITCHES_{95}$, %	$\overline{REBUF}$, sec/min	$REBUF_{95}$, sec/min	$\overline{QOE}$, %	QOE_5, %
A	RMPC-A	Slow	-1.02	-5.16	10.49	0.44	3.05	16.71	26.55	116.34
A	RMPC-A	Medium	-0.12	0.5	11.02	13.56	0.04	0.88	2.98	7.19
A	RMPC-A	Fast	0.03	0.13	3.08	3.01	0.0	0.0	1.02	1.14
B	RMPC-A	Slow	-0.91	-3.4	-0.08	-7.3	2.7	12.96	21.55	97.2
B	RMPC-A	Medium	-2.57	-2.08	-2.08	0.85	0.15	0.68	1.61	6.25
B	RMPC-A	Fast	-3.31	-3.6	-19.44	-12.53	-0.0	0.0	-4.0	-2.82
C	RMPC-A	Slow	0.43	-2.21	-5.11	-8.65	2.67	16.29	19.03	76.58
C	RMPC-A	Medium	0.58	0.01	1.22	-1.4	0.0	-0.2	1.25	-0.1
C	RMPC-A	Fast	0.25	0.27	0.37	2.07	0.01	0.0	0.42	0.54
D	RMPC-A	Slow	-0.66	-5.5	13.71	-9.98	2.75	12.68	22.12	84.62
D	RMPC-A	Medium	0.13	0.49	16.17	20.73	0.0	-0.35	3.42	3.75
D	RMPC-A	Fast	0.02	1.14	8.84	24.58	-0.0	0.0	1.49	5.09
E	RMPC-A	Slow	-0.59	-3.38	-0.74	-10.55	3.35	18.74	28.57	160.06
E	RMPC-A	Medium	-0.08	0.92	1.74	1.74	0.02	-0.26	3.44	3.79
E	RMPC-A	Fast	-0.03	0.36	2.3	2.64	-0.0	-0.03	2.03	3.6
F	RMPC-A	Slow	0.29	-3.18	5.08	-5.21	3.4	18.93	25.08	113.61
F	RMPC-A	Medium	0.23	1.37	10.19	11.22	0.05	0.88	2.8	6.31
F	RMPC-A	Fast	0.18	-0.82	5.66	6.63	-0.02	-0.03	1.05	0.81
G	RMPC-A	Slow	-1.15	-3.45	-0.35	-2.58	1.43	8.56	13.46	49.48
G	RMPC-A	Medium	-3.0	-2.07	4.47	6.25	0.23	0.57	6.34	7.34
G	RMPC-A	Fast	-4.29	-0.69	4.6	10.61	0.0	0.0	1.23	4.33
H	RMPC-A	Slow	-0.12	-4.02	4.69	-1.61	2.37	18.49	13.9	118.54
H	RMPC-A	Medium	-0.44	-0.16	4.33	9.52	0.02	0.15	0.59	-0.08
H	RMPC-A	Fast	-0.7	1.33	3.18	8.21	-0.01	0.0	-0.19	2.02
I	RMPC-A	Slow	-1.04	-6.36	13.21	-1.09	4.7	30.61	28.51	199.32
I	RMPC-A	Medium	-6.16	-1.61	3.44	26.14	0.12	0.52	-3.44	1.86
I	RMPC-A	Fast	-12.54	-8.99	-23.37	-4.44	0.0	0.0	-12.59	-7.61
L	RMPC-A	Slow	2.12	-4.3	5.61	-1.26	4.23	20.03	24.98	121.83
L	RMPC-A	Medium	1.49	2.99	10.06	12.56	0.03	-0.15	0.56	5.92
L	RMPC-A	Fast	0.31	2.7	6.12	11.82	0.01	0.0	-0.1	2.28
M	RMPC-A	Slow	0.29	-4.69	2.38	-4.7	2.23	21.05	16.72	130.41
M	RMPC-A	Medium	0.06	2.57	1.58	-0.33	0.11	-0.36	2.91	3.1
M	RMPC-A	Fast	-0.21	0.32	-3.27	-7.31	0.03	0.0	0.67	2.34
A	RMPC-O	Slow	-1.22	-4.66	9.98	5.01	2.94	16.67	25.97	118.67
A	RMPC-O	Medium	0.03	-0.6	10.75	13.52	0.0	-0.31	1.9	-0.55
A	RMPC-O	Fast	-0.05	-0.38	4.29	9.88	0.0	0.0	0.76	2.97
B	RMPC-O	Slow	-1.01	-4.16	-9.53	-14.8	2.77	12.68	21.29	88.55
B	RMPC-O	Medium	-1.89	0.16	-6.09	-1.25	0.11	0.6	1.85	9.64
B	RMPC-O	Fast	-3.32	-1.78	-23.57	-9.95	0.02	0.0	-3.68	-0.52
C	RMPC-O	Slow	0.86	-2.02	-2.15	-5.6	2.67	15.36	19.84	69.76
C	RMPC-O	Medium	0.65	0.99	2.69	0.03	0.03	0.06	1.6	3.86
C	RMPC-O	Fast	0.46	1.63	1.79	3.41	0.02	0.28	0.72	2.9
D	RMPC-O	Slow	-0.58	-5.12	3.69	-4.98	2.79	11.45	21.32	77.23
D	RMPC-O	Medium	-0.04	0.59	1.43	5.56	0.01	-0.07	1.63	-3.0
D	RMPC-O	Fast	-0.07	0.55	3.95	11.27	-0.01	0.0	0.86	2.66
E	RMPC-O	Slow	-0.47	-3.14	4.86	-3.67	3.44	16.37	29.97	145.93
E	RMPC-O	Medium	-0.23	-0.51	2.74	6.77	0.01	0.19	2.47	3.57
E	RMPC-O	Fast	0.17	0.32	4.41	6.3	0.01	-0.07	1.93	5.96
F	RMPC-O	Slow	0.01	-3.06	4.18	3.2	3.15	19.67	23.99	134.52
F	RMPC-O	Medium	0.36	-0.19	8.41	8.5	0.05	0.08	2.75	-0.55
F	RMPC-O	Fast	0.07	0.09	5.45	10.33	-0.03	-0.09	1.08	2.54
G	RMPC-O	Slow	-0.68	-3.05	0.69	7.29	1.3	7.22	12.11	49.42
G	RMPC-O	Medium	-1.87	-0.25	4.85	10.14	0.23	0.12	5.33	6.09
G	RMPC-O	Fast	-3.81	-1.07	7.67	24.42	0.01	0.0	1.15	5.8
H	RMPC-O	Slow	0.4	-4.3	10.01	3.36	2.42	19.22	14.71	123.9
H	RMPC-O	Medium	0.55	0.6	7.65	14.65	-0.09	-0.59	0.36	-3.12
H	RMPC-O	Fast	-0.12	1.42	5.33	7.78	0.0	0.0	0.5	1.52
I	RMPC-O	Slow	0.92	-5.42	-3.14	-8.16	3.89	24.79	24.57	160.45
I	RMPC-O	Medium	1.78	1.08	-0.37	-1.08	0.11	0.0	1.0	2.69
I	RMPC-O	Fast	-3.38	0.44	-12.29	-5.94	0.0	0.0	-4.51	-0.56
L	RMPC-O	Slow	1.57	-3.92	-2.03	-1.43	4.16	19.32	23.84	102.37
L	RMPC-O	Medium	0.99	0.77	4.49	4.56	0.0	-0.23	0.58	-5.23
L	RMPC-O	Fast	0.06	1.63	-0.1	11.22	-0.02	-0.04	-1.01	2.07
M	RMPC-O	Slow	0.54	-4.58	3.84	5.94	2.41	16.38	18.33	117.74
M	RMPC-O	Medium	0.5	2.63	3.28	5.57	0.03	-0.23	2.1	3.88
M	RMPC-O	Fast	-0.06	0.35	0.47	2.73	-0.02	0.0	0.75	2.39

Figure 5.20: *SEGUE's results with WideEye segmentation and σ_{bv} augmentation for RMPC-A and RMPC-O, compared to Constant.*

how closely we could integrate offline encoding and online adaptation.

As our results show, it is non-trivial to tweak algorithms like RMPC, which bake in today's typical constant-length segmentation in their design, to work well with SEGUE. Going further, how would SEGUE interact with an adaptation algorithm like CAVA [Qin+18], which explicitly tackles variable bitrate encoding. Does either reduce the other's utility? Or does CAVA's non-myopic behavior benefit from SEGUE's offline preparation, resulting in even larger benefits?

Likewise, on the encoding side, does video for which bitrates are tuned per scene, like Netflix has started doing [Net18a], reduce the benefit of SEGUE's augmentation? Does it increase the benefit of SEGUE's segmentation? How do the answers to these questions depend on the adaptation algorithms used?

In the context of co-designing adaptation logic with SEGUE, the most straightforward next step would be to modify SEGUE itself to output a set of representations, both in space and time, and to modify `ABR` logic to select (online) between these representations. We plan to investigate this path in future work.

Deployment considerations: SEGUE requires rethinking some aspects of video delivery: (1) As different segments have different numbers of tracks available, any user interface elements for manually selecting a track (disabling adaptation) need to hide that difference and make background decisions accordingly; (2) While we don't expect the potentially frequent and minor tweaks in a provider's adaptation algorithm to have large effects, large changes to adaptation will need to be compatible with the video library's chunking, although this is not very different from today — constant length segmentation is just one (implicit) choice.

5.9 Conclusion

SEGUE is the first work to investigate offline video chunking in a manner that accounts for the interactions of online rate adaptation with temporal variability in video complexity. Besides showing promising performance improvements, especially for challenging settings involving complex videos or low-bandwidth conditions, it calls for closer integration of offline and online phases. We discuss several exciting open questions, and release our code to enable their exploration.

6 Conclusions and suggested future works

In this dissertation, we show how network measurements and simulation can be used to understand, reconstruct and optimise video streaming applications.

In Chapter 3 we conduct a broad comparison of adaptive bitrate video streaming algorithms deployed in the wild across 10 large video platforms offering varied content targeted at different audiences. We find large differences in player behavior, with a wide spectrum of choices instantiated across virtually all metrics we examined. For instance, our results show that: (a) some deployed ABRs are conscious of perceptual quality metrics while others focused on bitrate; (b) no deployed ABRs follow available bandwidth as closely as research ABRs; and (c) several ABRs leave a large fraction of available network capacity unused. Whether this diversity of design choices and behaviors stems from careful tailoring towards different use cases and optimization objectives, or is merely a natural consequence of sub-optimal, independent design is unclear. But if large, otherwise extremely well-engineered platforms like YouTube differ so substantially from state-of-the-art research ABRs, then it is at least plausible that ABR research is more narrowly focused than desirable.

In Chapter 4 we take a further step into the analysis of propertary video streaming algorithms. We pursue an ambitious goal: reconstructing unknown proprietary streaming algorithms in a human-interpretable manner. We customize and evaluate a rule-set approach, achieving good results for reproducing the behavior of algorithms deployed at several popular online services. Our approach produces succinct output, open to expert interpretation and modification, and we discuss through several examples the utility of this interpretability. While promising, our results also expose a likely fundamental limitation — we need to encode and make available suitable domain knowledge to the learning approach. This can be interpreted as suggesting that we should reconcile learning with our already acquired human expertise, instead of starting afresh. We hope to apply this approach, suitably customized, to congestion control as well, where it is unclear how much diversity there is in actual deployment of different, unknown congestion control algorithms across popular Web services.

Based on what we learned throughout our investigations of ABR algorithms, in Chapter 5 we propose SEGUE, one of the first works to investigate offline video chunking in a manner that accounts for the interactions of online rate adaptation with the temporal variability in video complexity. To do so, SEGUE uses a simulation-based method, exploring and evaluating segmentations and augmentations of a video flow across a large set of network traces. In such a simulations an ABR algorithm is used to make decisions, and a QoE function is used to rank different chunking candidates. Besides showing promising performance improvements, especially for challenging settings involving complex videos or low-bandwidth conditions, it suggests many future opportunities for research. In the current implementation, SEGUE's tries to find common patterns among different classes of traces, and adapts the video segmentation and quality levels to counteract impairments. Given the promising results, an interesting further step in this direction could be the co-design of adaptation logic with SEGUE. In this context, SEGUE could be tuned to output a set of representations, rather than a single one, and the ABR logic should be modified to select (online) between these representations. We plan to investigate this path in future works.

List of Tables

3.1 We test a diverse set of large video platforms. 49

5.1 *An overview of our video dataset. Videos **C** and **D** lie at the extremes of highly stable and unstable in terms of perceptual quality over time within one track.* 122

List of Figures

2.1 VOD streaming architecture involve multiple steps. Video preparation usually happens offline. In this stage, the content is re-encoded at multiple bitrate-resolution pairs and partitioned into few seconds long segments. The content then is distributed and replicated in Content Delivery Networks (CDN).CDN's optimisation strategies can involve both offline and online content distribution algorithms. These control loops are out of the scope of this dissertation. Online, users' then fetch video segments from the closest CDN. An ABR algorithm decides the quality to fetch the next segment at. 20
2.2 Diagram of the prediction relationship between I-Frame, P-Frame and B-Frame. 22

2.3 Blender Sintel Trailer [Ble10] rate distortion curves varying the target bitrate from 200 Kbit/s to 2 Mbit/s with *libx264* library for `CBR`, `1-Pass ABR` and `2-Pass ABR` rate control modes. The quality metric under test is Netflix VMAF [Nat+19; Li+16], described in section 2.2.3. Both `ABR` rate control modes have been configuted to use, as maximum rate, two times the target rate. `2-Pass ABR` is able to reach an higher perceptual quality for the same target bitrate with respect to both `1-Pass ABR` and `CBR`, at the cost of higher encoding time (+35% in average for the depicted benchmark). 26

2.4 Two different video contents have been encoded at different `CRF` values. The content on the left represent a video of a vase of flower with a static white background, while the video on the right records a slowly moving crowd. We encoded these video at various `CRF` values and we then extracted the first frame of each video at two equal values of `PSNR`, one high (56dB, top) and one low (41dB, botton). The drop in quality from 56dB to 41dB is much more noticeable in the video content on the left, while it is barely noticeable on the content on the right. 28

2.5 The main concept behind rate adaptation algorithm is to download the best quality possible depending on varying network conditions. The red shaded area of the pictures highlight a very simple adaptation approach: when the capacity of the network decreases, a lower quality level is picked. 35

2.6 Graphical representation of a buffer based approach described in [Hua+14]. The red shaded area of the 2D plane identifies the risky buffer health-rate mapping, in which rebuffering events are more likely to happen. Conversely, the green shaded area identifies the safe area of the mapping, in which rebuffering events are unlikely to happen, at the cost of being too conservative with the chosen quality. 37

3.1 (a) Player behaviour is influenced through bandwidth throttling, and is recorded from multiple sources. (b) The proxy has little impact on player behavior as measured in terms of average linear QoE ($\overline{QoE_{linear}}$); the whiskers are the 95% confidence interval. 43

3.2 (a) Initialization: most providers start playback after one chunk is downloaded. (b) Convergence is measured in terms of changes in bitrate switching, *i.e.,* the (absolute) sum of bitrate differentials across all switches from the start, divided by the thus-far playback duration. As expected, switching is more frequent during startup, but the degree of switching varies across providers both in startup and later. . . . 52

3.3 Risk-tolerance: YouTube operates with nearly 4× the buffer for Fandom. The shaded regions show the 95% confidence interval around the mean. 54

3.4 We measure reactivity in terms of bitrate evolution after a bandwidth increase, *i.e.,* difference in average playback bitrate after and before the bandwidth change over time (in terms of chunk downloads). The plots show the reactivity differences between: (left) mobile and desktop versions of Vimeo; and (right) TubiTV and MPC. 55

3.5 (a) Bandwidth usage: many players use surprisingly little of the available network bandwidth (Played / Download-able) despite the potential to improve quality with more bandwidth, *e.g.,* XVideos uses only 50% of it; and some players, like Arte, spend a large fraction of their used bandwidth on redownloads. (b) `QoE` goal: we measure how much a player prefers high-`VMAF` chunks by quantifying the average quality-level difference between all chunks and only the top-x% of chunks by `VMAF` (*i.e.,* $\overline{Q_{[0...\%Top]}}$). Some players, like Twitch, show a large preference for high-`VMAF` chunks. 57

4.1 Minimal rule-set for a reservoir-based algorithm, which uses only the last chunk's throughput estimate to pick a quality level. 69

4.2 The big image is the target decision plane, with 2 labels. On the right are its approximations with decision trees of different rule-lengths, going from 1 to 4. 71

4.3 Here, the target decision plane (big, left) is governed by the mean of $\mathbb{T}_{N-1}$ and $\mathbb{T}_{N-2}$. The smaller figures show that we need long rules to approximate this if we are restricted to using individual literals ($\mathbb{T}_{N-1}$ and $\mathbb{T}_{N-2}$) in our rules. 73

4.4 A decision plane comparing the throughput estimate to the chunk bitrate is difficult to capture without long rules if rules can only be framed in terms of the individual literals. 74

4.5 The generated rule-sets are never worse by more than 8% and 20% than the blackbox approach on agreement and similarity respectively. In contrast, the rate-based approach achieves extremely poor results. 83

4.6 For all but 3 of the 12 targets, the reconstruction matches the target algorithm very closely. For YouTube, Fandom, and PornHub, there is a substantial difference in performance; these are the same 3 providers in the bottom-left of Fig. 4.5, for which we achieve the lowest agreement and similarity scores as well. 84

4.7 Domain knowledge helps the rule-set (Bitrate-QoE). . 86

4.8 Domain knowledge helps the rule-set (VMAF-QoE). . 86

4.9 The core of the decision tree generated by learning from the SRF example data. The green (solid) and red (dashed) arrows are the "True" and "False" evaluations of the predicates at the nodes. The video quality of the last fetched chunk is denoted by ∘ and the next chunk's by • . Potential decisions and decision outcomes are coded in terms of the relationships between these qualities: *e.g.*, ∘┘• denotes that the next chunk is requested at one higher video quality level. The predicates are in terms of expected buffer size after a certain potential decision, based on throughput estimates (*e.g.*, $\mathbb{T}^+$ is an aggressive/optimistic estimator); or on a reward ($\mathcal{R}_{relative}$) calculated relative to the other obtainable rewards involving throughput, rebuffering penalty ($\mathscr{P}$), the lookahead horizon over which these are estimated ($\mathscr{W}$), etc. 88

4.10 Reconstructing commercial ABR algorithms allows us to uniformly compare them to both other commercial and academic ones under the same test conditions. . . 89

4.11 Twitch shows only marginally more reluctance towards switching when compared to MPC 94

5.1 *Fig. (a) shows the variability in bitrate for three different resolutions (480p, 360p and 240p). Fig. (b) shows the observed probability of rebuffering for the segments plotted in Fig. (a) for a buffer-based algorithm for three different traces buckets (blue line: Slow, orange line: Medium, green line: Fast). Fig. (c) shows the correspondent average buffer occupancy. We highlight that higher observed rebuffering probability correspond to drops into the average buffer occupancy (red shaded in the picture). Conversely, drops into the observed rebuffering probability correspond to higher average buffer occupancy. Fig. (d) plots the average buffer occupancy for the same segments for a rate-based algorithm for the same traces buckets. The buffer behaviour varies substantially with respect to a buffer-based algorithm.* 105

5.2 *(a) The violet (solid) line is the bitrate per frame, while the red (dashed) line marks the keyframes. (b) Breaking the video into fixed length segments produce segments with high internal perceptual quality instability. (c) Average bitrate for a video encoded at 4 resolutions: due to VBR encoding, the bitrate per segment varies.* 108

5.3 *The distribution of QoE per second of playback (normalized to max. possible QoE) across our test network traces for video **A** with BB and RMPC rate adaptation, and video **D** with BB adaptation. The box-plot shows the quartiles, with whiskers for 20th and 80th percentiles.* 109

5.4 *We can segment vulnerable parts of the video into shorter segments, and augment them with additional tracks.* 111

5.5 *Segment characteristics for different schemes — video **A**, RB. Boxes show mean and quartiles, whiskers are 5/95-percentile.* 123

5.6 *VMAF fluctuations: video **A**, RB adaptation, SLOW traces.* 124
5.7 *Rebuffer ratio: video **A**, RB adaptation, SLOW traces.* 125
5.8 *σ_{bv} improves QoE with only small byte overheads, as does λ_{bv} with appropriate parameters. λ_{bv} is shown with $V \in \{5, 6, 7, \ldots, 14\}$ for both $B = 15\%$ and $B = 10\%$. ($B = 5\%$ is similar to $B = 10\%$.)* . . . 126
5.9 *VMAF stability improvements divided by trace set and ABR. As expected, improvements are more significant for RB, given that no stability policy is implemented in the ABR.* 128
5.10 *Improvements from adding σ_{bv} to WideEye. We average metrics across traces, and show their distribution across videos. Boxes show mean and quartiles, whiskers are 5/95-percentile.* 130
5.11 *Augmentation yields local VMAF improvements: average VMAF/sec for video **E** using RMPC, Constant segmentation* 131
5.12 *VMAF and bitrate comparison between video **C** and video **D** for the highest quality track of each. VMAF and bitrate are averaged per second, and shown for the first 100 seconds of playback. Video **C** exhibits much greater stability than video **D**.* 132
5.13 *Performance improvements of SEGUE's segmentation strategy WideEye over the delivery of single GOPs, varying the GOP size.* 134
5.14 *Performance improvements and degradation of SEGUE's augmentation strategy σ_{bv} compared to CBF [Qin+19] and SIVQ [Rai+17]. Rebuffering ratio comparison for FAST traces is neglibile, and thus it has been omitted.* 135
5.15 *Benchmarking of SEGUE's performance for video **B** as a ratio between the computational time and the video length. SEGUE's brute force exploration time is heavily affected by the ABR algorithm efficiency.* . . 137

5.16 *Comparison of resulting `QoE` between simulation and `dash.js` implementation on the same set of 100 traces for video **A** and `RB`. The `QoE` is normalized by the mean of constant length.* 140
5.17 *Segue's results with WideEye segmentation and no augmentation for BB and RB, compared to Constant.* 142
5.18 *Segue's results with WideEye segmentation and no augmentation for RMPC-A and RMPC-O, compared to Constant.* 143
5.19 *Segue's results with WideEye segmentation and σ_{bv} augmentation for BB and RB, compared to Constant.* 144
5.20 *Segue's results with WideEye segmentation and σ_{bv} augmentation for RMPC-A and RMPC-O, compared to Constant.* 145

Bibliography

[Aar+] Anne Aaron, Zhi Li, Megha Manohara, Jan De Cock, and David Ronca. *Per-title Encode Optimization*. https://link.medium.com/jEeb6GV0ZW (cit. on pp. 14, 25, 71, 117).

[ABD11] Saamer Akhshabi, Ali Begen, and Constantine Dovrolis. "An experimental evaluation of rate-adaptation algorithms in adaptive streaming over HTTP". In: *ACM MMSys*. 2011 (cit. on p. 65).

[Aka19] Akamai. *Enhancing Video Streaming Quality for ExoPlayer?Part 1: Quality of User Experience Metrics*. https://www.akamai.com/blog/performance/enhancing-video-streaming-quality-for-exoplayer-part-1-quality-of-user-experience-metrics. 2019 (cit. on p. 14).

[Akh+18] Zahaib Akhtar, Yun Seong Nam, Ramesh Govindan, Sanjay Rao, Jessica Chen, Ethan Katz-Bassett, Bruno Ribeiro, Jibin Zhan, and Hui Zhang. "Oboe: auto-tuning video ABR algorithms to network conditions". In: *ACM SIGCOMM*. 2018 (cit. on pp. 41, 46, 49, 50, 61, 65, 79, 80, 120).

[Ama] Amazon. *Amazon Prime Terms of Use*. URL: https://www.amazon.co.uk/gp/help/customer/display.html?nodeId=201909000&pop-up= (cit. on p. 48).

[Ami] Remita Amine. *YouTube Downloader*. URL: https://github.com/ytdl-org/youtube-dl/ (cit. on pp. 44, 45).

[Año+18] Javier Añorga, Saioa Arrizabalaga, Beatriz Sedano, Jon Goya, Maykel Alonso-Arce, and Jaizki Mendizabal. "Analysis of YouTube's traffic adaptation to dynamic environments". In: *Multimedia Tools and Applications* (2018) (cit. on p. 42).

[Aya+18] Ibrahim Ayad, Youngbin Im, Eric Keller, and Sangtae Ha. "A Practical Evaluation of Rate Adaptation Algorithms in HTTP-based Adaptive Streaming". In: *Computer Networks* 133 (Mar. 2018), pp. 90–103. DOI: 10.1016/j.comnet.2018.01.019 (cit. on pp. 36, 119).

[Bal+18] Roberto Baldoni, Emilio Coppa, Daniele Cono D'Elia, Camil Demetrescu, and Irene Finocchi. "A Survey of Symbolic Execution Techniques". In: *ACM Computing Surveys* (2018) (cit. on p. 90).

[Bit19a] Bitmovin. *Choosing the Right Video Bitrate for Streaming HLS and DASH*. https://bitmovin.com/video-bitrate-streaming-hls-dash/. 2019 (cit. on p. 117).

[Bit19b] Bitmovin. *Video Developer Report*. https://go.bitmovin.com/video-developer-report-2019/. 2019 (cit. on p. 21).

[BK19] Lionel Blondé and Alexandros Kalousis. "Sample-Efficient Imitation Learning via Generative Adversarial Nets". In: *PMLR*. 2019 (cit. on p. 66).

[Ble10] Blender. *Sintel*. 2010. URL: %5Curl%7Bhttps://durian.blender.org/download/%7D (cit. on pp. 25, 26).

[BPS18] Osbert Bastani, Yewen Pu, and Armando Solar-Lezama. "Verifiable Reinforcement Learning via Policy Extraction". In: *NeurIPS*. 2018 (cit. on pp. 63, 67, 76).

[Cho+14] Kyunghyun Cho, B van Merrienboer, Caglar Gulcehre, F Bougares, H Schwenk, and Yoshua Bengio. "Learning phrase representations using RNN encoder-decoder for statistical machine translation". In: *EMNLP*. 2014 (cit. on p. 75).

[Clo19] Cloudflare. *What is MPEG DASH?* https://www.cloudflare.com/learning/video/what-is-mpeg-dash/. 2019 (cit. on p. 31).

[Con] Open Source / Software Freedom Conservancy. *Selenium WebDriver*. https://www.selenium.dev/ (cit. on p. 139).

[DAS12] DASH Industry Forum. *A reference client implementation for the playback of MPEG DASH via JavaScript and compliant browsers.* https://github.com/Dash-Industry-Forum/dash.js. 2012 (cit. on pp. 32, 138).

[DCK19] Arnaud Dethise, Marco Canini, and Srikanth Kandula. "Cracking Open the Black Box: What Observations Can Tell Us About Reinforcement Learning Agents". In: *ACM NetAI*. 2019 (cit. on pp. 66, 88).

[De +13] Luca De Cicco, Vito Caldaralo, Vittorio Palmisano, and Saverio Mascolo. “Elastic: a client-side controller for dynamic adaptive streaming over HTTP (DASH)”. In: *IEEE Packet Video Workshop (PV)*. 2013 (cit. on p. 41).

[DM10] Luca De Cicco and Saveri o Mascolo. “An Experimental Investigation of the Akamai Adaptive Video Streaming”. In: *USAB*. 2010 (cit. on p. 65).

[EA19] Anis Elgabli and Vaneet Aggarwal. “Fastscan: Robust low-complexity rate adaptation algorithm for video streaming over HTTP”. In: *IEEE TCSVT*. 2019 (cit. on p. 65).

[Fed] Federal Communications Commission. *Validated Data September 2017 - Measuring Broadband America*. URL: https://www.fcc.gov/reports-research/reports/ (cit. on pp. 50, 79, 80).

[FFMa] FFMPEG. *AV1 Encode*. URL: https://trac.ffmpeg.org/wiki/Encode/AV1 (cit. on p. 21).

[FFMb] FFMPEG. *H.264 Encode*. URL: https://trac.ffmpeg.org/wiki/Encode/H.264 (cit. on p. 21).

[FFMc] FFMPEG. *H.265 Encode*. URL: https://trac.ffmpeg.org/wiki/Encode/H.265 (cit. on p. 21).

[Fou+18] Sadjad Fouladi, John Emmons, Emre Orbay, Catherine Wu, Riad S. Wahby, and Keith Winstein. “Salsify: Low-Latency Network Video through Tighter Integration between a Video Codec and a Transport Protocol”. In: *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*. Renton, WA: USENIX Association, Apr. 2018, pp. 267–282. ISBN: 978-1-939133-01-4. URL: https://www.usenix.org/conference/nsdi18/presentation/fouladi (cit. on p. 104).

[Gha+16] Mojgan Ghasemi, Partha Kanuparthy, Ahmed Mansy, Theophilus Benson, and Jennifer Rexford. “Performance characterization of a commercial video streaming service”. In: *ACM IMC*. 2016 (cit. on p. 42).

[GL] Maximilian Grüner and Melissa Licciardello. *Understanding video streaming algorithms in the wild - scripts*. URL: https://github.com/magruener/understanding-video-streaming-in-the-wild (cit. on p. 44).

[GLS20a] Maximilian Grüner, Melissa Licciardello, and Ankit Singla. “Reconstructing proprietary video streaming algorithms”. In: *2020 USENIX Annual Technical Conference (USENIX ATC 20)*. USENIX Association, July 2020. URL: https://www.usenix.org/conference/atc20/presentation/gruener (cit. on pp. 17, 67).

[GLS20b] Maximilian Grüner, Melissa Licciardello, and Ankit Singla. *Reconstructing proprietary video streaming algorithms*. https://github.com/magruener/reconstructing-proprietary-video-streaming-algorithms. 2020 (cit. on p. 64).

[Grü19] Maximilian Grüner. "Human-interpretable auto-inference of networked algorithms". Master's Thesis. Switzerland.: Department of Computer Science, ETH Zurich, 2019 (cit. on p. 17).

[GZN07] H. Guo, Q. Zhang, and A. K. Nandi. "Feature generation using genetic programming based on fisher criterion". In: *IEEE EUSIPCO*. 2007 (cit. on pp. 67, 75, 77).

[HE16] Jonathan Ho and Stefano Ermon. "Generative Adversarial Imitation Learning". In: *NIPS*. 2016 (cit. on pp. 66, 75).

[Hoo+16] 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". In: *IEEE Communications Letters* 20.11 (2016), pp. 2177–2180 (cit. on pp. 50, 80).

[Hoo+18] Jeroen van der Hooft, Dries Pauwels, Cedric De Boom, Stefano Petrangeli, Tim Wauters, and Filip De Turck. "Low-Latency Delivery of News-Based Video Content". In: *Proceedings of the 9th ACM Multimedia Systems Conference*. MMSys '18. Amsterdam, Netherlands: Association for Computing Machinery, 2018, pp. 537–540. ISBN: 9781450351928. DOI: 10.1145/3204949.3208110. URL: https://doi.org/10.1145/3204949.3208110 (cit. on pp. 103, 110).

[Hua+14] Te-Yuan Huang, Ramesh Johari, Nick McKeown, Matthew Trunnell, and Mark Watson. "A Buffer-Based Approach to Rate Adaptation: Evidence from a Large Video Streaming Service". In: *ACM SIGCOMM*. 2014 (cit. on pp. 36, 37, 62, 69, 119).

[Hug] Jason Huggins. *Selenium WebDriver*. URL: https://www.seleniumhq.org/projects/webdriver/ (cit. on p. 44).

[Hul] Hulu. *Hulu Terms of Use*. URL: https://www.hulu.com/terms (cit. on p. 48).

[Ibr+18] Ayad Ibrahim, Im Youngbin, Keller Eric, and Ha Sangtae. "A Practical Evaluation of Rate Adaptation Algorithms in HTTP-based Adaptive Streaming". In: *Computer Networks*. 2018 (cit. on p. 65).

[IWG16] Iheanyi Irondi, Qi Wang, and Christos Grecos. “Optimized adaptation algorithm for HEVC/H.265 dynamic adaptive streaming over HTTP using variable segment duration”. In: *Real-Time Image and Video Processing 2016*. Ed. by Nasser Kehtarnavaz and Matthias F. Carlsohn. Vol. 9897. Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series. Apr. 2016, 98970O. DOI: 10.1117/12.2227733 (cit. on p. 103).

[JSZ14] J. Jiang, V. Sekar, and H. Zhang. “Improving Fairness, Efficiency, and Stability in HTTP-Based Adaptive Video Streaming With Festive”. In: *IEEE/ACM Transactions on Networking* 22.1 (Feb. 2014), pp. 326–340. ISSN: 1063-6692. DOI: 10.1109/TNET.2013.2291681 (cit. on p. 41).

[KAB21] Angeliki V. Katsenou, Mariana Afonso, and David R. Bull. *Study of Compression Statistics and Prediction of Rate-Distortion Curves for Video Texture*. 2021. DOI: 10.48550/ARXIV.2102.04167. URL: https://arxiv.org/abs/2102.04167 (cit. on p. 138).

[Kaz+19] Yafim Kazak, Clark Barrett, Guy Katz, and Michael Schapira. “Verifying Deep-RL-Driven Systems”. In: *ACM NetAI*. 2019 (cit. on pp. 63, 66, 92).

[L+84] Breiman L., Friedman J. H., Olshen R. A., and Stone C. J. *Classification and Regression Trees*. Wadsworth and Brooks, 1984 (cit. on p. 70).

[LBE16] Y. Lin, T. Bonald, and S. E. Elayoubi. “Impact of chunk duration on adaptive streaming performance in mobile networks”. In: *2016 IEEE Wireless Communications and Networking Conference*. 2016, pp. 1–6 (cit. on p. 103).

[LBG11] C. Liu, I. Bouazizi, and M. Gabbouj. “Segment duration for rate adaptation of adaptive HTTP streaming”. In: *2011 IEEE International Conference on Multimedia and Expo*. 2011, pp. 1–4 (cit. on p. 103).

[Le 20] Jean Le Feuvre. “GPAC Filters”. In: *Proceedings of the 11th ACM Multimedia Systems Conference*. MMSys ’20. Istanbul, Turkey: Association for Computing Machinery, 2020, pp. 249–254. ISBN: 9781450368452. DOI: 10.1145/3339825.3394929. URL: https://doi.org/10.1145/3339825.3394929 (cit. on p. 138).

[LGS20] Melissa Licciardello, Maximilian Grüner, and Ankit Singla. “Understanding video streaming algorithms in the wild”. In: *PAM*. 2020 (cit. on pp. 17, 42, 65–67, 78, 79).

[Li+14] 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". In: *IEEE Journal on Selected Areas in Communications* 32.4 (Apr. 2014), pp. 719–733. ISSN: 0733-8716. DOI: 10.1109/JSAC.2014.140405 (cit. on p. 41).

[Li+16] Zhi Li, Anne Aaron, Ioannis Katsavounidis, Anush Moorthy, and Megha Manohara. *Toward A Practical Perceptual Video Quality Metric*. 2016. URL: https://medium.com/netflix-techblog/toward-a-practical-perceptual-video-quality-metric-653f208b9652 (cit. on pp. 26, 29, 45, 48, 68, 83, 101, 118).

[Lic+22a] Melissa Licciardello, Lukas Humbel, Fabian Rohr, Maximilian Grüner, and Ankit Singla. "Prepare Your Video for Streaming with Segue". In: *Journal of Systems Research* 2.1 (July 2022). DOI: 10.5070/SR32158113. URL: https://escholarship.org/uc/item/8m39f25q (cit. on pp. 17, 18).

[Lic+22b] Melissa Licciardello, Lukas Humbel, Fabian Rohr, Maximilian Grüner, and Ankit Singla. *Segue code repository*. https://github.com/melADTR/Segue. 2022 (cit. on p. 100).

[LPS15] Siu Kwan Lam, Antoine Pitrou, and Stanley Seibert. "Numba: A LLVM-Based Python JIT Compiler". In: *Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in HPC*. LLVM '15. Austin, Texas: Association for Computing Machinery, 2015. ISBN: 9781450340052. DOI: 10.1145/2833157.2833162. URL: https://doi.org/10.1145/2833157.2833162 (cit. on p. 138).

[LS10] De Cicco L. and Mascolo S. "A Mathematical Model of the Skype VoIP Congestion Control Algorithm". In: *IEEE Transactions on Automatic Control*. 2010 (cit. on pp. 65, 66).

[LTZ08] F. T. Liu, K. M. Ting, and Z. Zhou. "Isolation Forest". In: *IEEE ICDM*. 2008 (cit. on pp. 76, 81).

[Mao+19] Hongzi Mao, Shannon Chen, Drew Dimmery, Shaun Singh, Drew Blaisdell, Yuandong Tian, Mohammad Alizadeh, and Eytan Bakshy. "Real-world Video Adaptation with Reinforcement Learning". In: *Reinforcement Learning for Real Life (ICML workshop)* (2019) (cit. on pp. 42, 141).

[Men+19a] Zili Meng, Jing Chen, Yaning Guo, Chen Sun, Hongxin Hu, and Mingwei Xu. "PiTree: Practical Implementation of ABR Algorithms Using Decision Trees". In: *ACM Multimedia*. 2019 (cit. on p. 66).

[Men+19b] Zili Meng, Minhu Wang, Jiasong Bai, Mingwei Xu, Hongzi Mao, and Hongxin Hu. "Explaining Deep Learning-Based Networked Systems". In: *arXiv:1910.03835* (2019). arXiv: 1910.03835 (cit. on p. 66).

[Mil+15] Konstantin Miller, Dilip Bethanabhotla, Giuseppe Caire, and Adam Wolisz. “A control-theoretic approach to adaptive video streaming in dense wireless networks”. In: *IEEE Transactions on Multimedia* 17.8 (2015), pp. 1309–1322 (cit. on pp. 14, 34, 37, 41, 46, 48, 49, 53).

[MNA17a] Hongzi Mao, Ravi Netravali, and Mohammad Alizadeh. *Neural Adaptive Video Streaming with Pensieve*. https://github.com/hongzimao/pensieve. 2017 (cit. on p. 79).

[MNA17b] Hongzi Mao, Ravi Netravali, and Mohammad Alizadeh. “Neural adaptive video streaming with pensieve”. In: *ACM SIGCOMM*. ACM. 2017, pp. 197–210 (cit. on pp. 14, 38, 41, 42, 44, 48, 49, 66, 71, 90, 99, 119, 133).

[Mon+17] Abhijit Mondal, Satadal Sengupta, Bachu Rikith Reddy, M. J.V. Koundinya, Chander Govindarajan, Pradipta De, Niloy Ganguly, and Sandip Chakraborty. “Candid with YouTube: Adaptive Streaming Behavior and Implications on Data Consumption”. In: *ACM NOSSDAV*. 2017 (cit. on pp. 42, 45, 65, 84, 102).

[Nat+19] Vikram Nathan, Vibhaalakshmi Sivaraman, Ravichandra Addanki, Mehrdad Khani, Prateesh Goyal, and Mohammad Alizadeh. “End-to-End Transport for Video QoE Fairness”. In: *Proceedings of the ACM Special Interest Group on Data Communication*. SIGCOMM ’19. Beijing, China: Association for Computing Machinery, 2019, pp. 408–423. ISBN: 9781450359566. DOI: 10.1145/3341302.3342077. URL: https://doi.org/10.1145/3341302.3342077 (cit. on pp. 14, 26, 29, 80, 81, 85, 90, 101, 118).

[Net] Netflix. *Netflix Terms of Use*. https://help.netflix.com/legal/termsofuse (cit. on p. 48).

[Net+15] Ravi Netravali, Anirudh Sivaraman, Somak Das, Ameesh Goyal, Keith Winstein, James Mickens, and Hari Balakrishnan. “Mahimahi: Accurate Record-and-Replay for HTTP”. In: *2015 USENIX Annual Technical Conference (USENIX ATC 15)*. Santa Clara, CA: USENIX Association, July 2015, pp. 417–429. ISBN: 978-1-931971-225. URL: https://www.usenix.org/conference/atc15/technical-session/presentation/netravali (cit. on p. 140).

[Net18a] Netflix. *Dynamic optimizer - a perceptual video encoding optimization framework*. https://netflixtechblog.com/dynamic-optimizer-a-perceptual-video-encoding-optimization-framework-e19f1e3a277f. 2018 (cit. on pp. 14, 23–25, 102, 146).

[Net18b] Netflix. *Optimized shot-based encodes: Now Streaming!* https://netflixtechblog.com/optimized-shot-based-encodes-now-streaming-4b9464204830. 2018 (cit. on p. 102).

[OGR09] Michael Osborne, Roman Garnett, and Stephen Roberts. “Gaussian processes for global optimization”. In: *International Conference on Learning and Intelligent Optimization*. 2009 (cit. on p. 94).

[Oze19] Jan Ozer. *Introduction to ABR production and delivery*. http://conferences.infotoday.com/documents/347/W1_Ozer.pdf. 2019 (cit. on p. 117).

[Ped+11] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. “Scikit-learn: Machine Learning in Python”. In: *Journal of Machine Learning Research* (2011) (cit. on pp. 75, 76).

[PM15] R. Pantos and W. May. *HTTP Live Streaming Draft*. 2015. URL: https://tools.ietf.org/html/draft-pantos-http-live-streaming-17.html (cit. on p. 45).

[Qin+17] Yanyuan Qin, Ruofan Jin, Shuai Hao, Krishna R Pattipati, Feng Qian, Subhabrata Sen, Bing Wang, and Chaoqun Yue. “A control theoretic approach to ABR video streaming: A fresh look at PID-based rate adaptation”. In: *INFOCOM 2017-IEEE Conference on Computer Communications, IEEE*. IEEE. 2017, pp. 1–9 (cit. on p. 41).

[Qin+18] Yanyuan Qin, Shuai Hao, Krishna R Pattipati, Feng Qian, Subhabrata Sen, Bing Wang, and Chaoqun Yue. “ABR streaming of VBR-encoded videos: characterization, challenges, and solutions”. In: *ACM CoNEXT*. 2018 (cit. on pp. 14, 41, 42, 48, 65, 115, 119, 120, 145).

[Qin+19] Yanyuan Qin, Shuai Hao, Krishna R Pattipati, Feng Qian, Subhabrata Sen, Bing Wang, and Chaoqun Yue. “Quality-aware strategies for optimizing abr video streaming qoe and reducing data usage”. In: *Proceedings of the 10th ACM Multimedia Systems Conference*. 2019, pp. 189–200 (cit. on pp. 103, 104, 133, 135).

[Rai+17] B. Rainer, S. Petscharnig, C. Timmerer, and H. Hellwagner. “Statistically Indifferent Quality Variation: An Approach for Reducing Multimedia Distribution Cost for Adaptive Video Streaming Services”. In: *IEEE Transactions on Multimedia* 19.4 (2017), pp. 849–860. DOI: 10.1109/TMM.2016.2629761 (cit. on pp. 103, 104, 133, 135).

[RGB11] Stephane Ross, Geoffrey Gordon, and Drew Bagnell. “A Reduction of Imitation Learning and Structured Prediction to No-Regret Online Learning”. In: *PMLR*. 2011 (cit. on pp. 63, 67, 76).

[Rii+13a] Haakon Riiser, Paul Vigmostad, Carsten Griwodz, and Pål Halvorsen. “Commute path bandwidth traces from 3G networks: analysis and applications”. In: *ACM MMSys*. 2013 (cit. on pp. 50, 79, 80).

[Rii+13b] Haakon Riiser, Paul Vigmostad, Carsten Griwodz, and Pål Halvorsen. "Commute path bandwidth traces from 3G networks: analysis and applications". In: https://heim.ifi.uio.no/paalh/publications/files/mmsys2013-dataset.pdf. Feb. 2013, pp. 114–118. DOI: 10.1145/2483977.2483991 (cit. on p. 120).

[Roh21] Fabian Rohr. "CAVA on SEGUE and SEGUE in DASH: Verifying Simulated ABR Behaviour on the DASH-IF Reference Player". Master's Thesis. Switzerland.: Department of Computer Science, ETH Zurich, 2021 (cit. on pp. 17, 18).

[Rol+17] Reudismam Rolim, Gustavo Soares, Loris D'Antoni, Oleksandr Polozov, Sumit Gulwani, Rohit Gheyi, Ryo Suzuki, and Björn Hartmann. "Learning Syntactic Program Transformations from Examples". In: *ICSE*. 2017 (cit. on p. 67).

[San22] Sandvine. *The Global Internet Phenomena*. https://www.sandvine.com/phenomena/. 2022 (cit. on pp. 13, 14, 39).

[Sch+20] Susanna Schwarzmann, Nick Hainke, Thomas Zinner, Christian Sieber, Werner Robitza, and Alexander Raake. "Comparing Fixed and Variable Segment Durations for Adaptive Video Streaming: A Holistic Analysis". In: *Proceedings of the 11th ACM Multimedia Systems Conference*. MMSys '20. Istanbul, Turkey: Association for Computing Machinery, 2020, pp. 38–53. ISBN: 9781450368452. DOI: 10.1145/3339825.3391858. URL: https://doi.org/10.1145/3339825.3391858 (cit. on pp. 103, 133, 134).

[Sim21] Gary Sims. *3 things you should know about the AV1 codec*. 2021. URL: https://www.androidauthority.com/av1-codec-1113318/ (cit. on pp. 22, 138).

[SL18] Reliable Secure and Intelligent Systems Lab. *JSNice - STATISTICAL RENAMING, TYPE INFERENCE AND DEOBFUSCATION*. http://jsnice.org/. 2018 (cit. on p. 65).

[SSS18] Kevin Spiteri, Ramesh Sitaraman, and Daniel Sparacio. "From theory to practice: Improving bitrate adaptation in the DASH reference player". In: *ACM MMSys*. 2018 (cit. on pp. 42, 65, 89, 99, 119).

[Ste17] Trevor Stephens. *Genetic Programming in Python*. https://github.com/trevorstephens/gplearn. 2017 (cit. on p. 78).

[Sto+17] Denny Stohr, Alexander Frömmgen, Amr Rizk, Michael Zink, Ralf Steinmetz, and Wolfgang Effelsberg. "Where are the sweet spots?: A systematic approach to reproducible DASH player comparisons". In: *ACM Multimedia*. 2017 (cit. on p. 42).

[Sun+16] Yi Sun, Xiaoqi Yin, Junchen Jiang, Vyas Sekar, Fuyuan Lin, Nanshu Wang, Tao Liu, and Bruno Sinopoli. "CS2P: Improving video bitrate selection and adaptation with data-driven throughput prediction". In: *ACM SIGCOMM*. 2016 (cit. on p. 41).

[SUS16] Kevin Spiteri, Rahul Urgaonkar, and Ramesh K. Sitaraman. "BOLA: Near-Optimal Bitrate Adaptation for Online Videos". In: *IEEE INFOCOM*. 2016 (cit. on pp. 36, 41, 90).

[Tec22] Techjury. *Virtual Reality Statistics*. https://techjury.net/blog/virtual-reality-statistics. 2022 (cit. on p. 13).

[TMR16] Christian Timmerer, Matteo Maiero, and Benjamin Rainer. "Which Adaptation Logic? An Objective and Subjective Performance Evaluation of HTTP-based Adaptive Media Streaming Systems". In: *CoRR* (2016) (cit. on p. 42).

[Uni16] Ghent University. *4G/LTE Bandwidth Logs*. https://users.ugent.be/~jvdrhoof/dataset-4g/. 2015-2016 (cit. on p. 120).

[Uni20] Standford University. *Puffer player*. https://puffer.stanford.edu/. 2020 (cit. on p. 121).

[Ver+18] Abhinav Verma, Vijayaraghavan Murali, Rishabh Singh, Pushmeet Kohli, and Swarat Chaudhuri. "Programmatically Interpretable Reinforcement Learning". In: *PMLR*. 2018 (cit. on p. 67).

[VH13] B. J. Villa and P. E. Heegaard. "Group Based Traffic Shaping for Adaptive HTTP Video Streaming by Segment Duration Control". In: *2013 IEEE 27th International Conference on Advanced Information Networking and Applications (AINA)*. 2013, pp. 830–837 (cit. on p. 103).

[Wam+16] Florian Wamser, Pedro Casas, Michael Seufert, Christian Moldovan, Phuoc Tran-Gia, and Tobias Hossfeld. "Modeling the YouTube stack: From packets to quality of experience". In: *Computer Networks* (2016) (cit. on p. 42).

[Wan+19] Ruohan Wang, Carlo Ciliberto, Pierluigi Vito Amadori, and Yiannis Demiris. "Random Expert Distillation: Imitation Learning via Expert Policy Support Estimation". In: *PMLR*. 2019 (cit. on pp. 66, 75).

[Wes18] Dimitri Wessels. "Quantifying and Explaining Unfairness in Online Video Streaming". Bachelor's Thesis. Switzerland.: Department of Computer Science, ETH Zurich, 2018 (cit. on p. 17).

[WRZ16] Cong Wang, Amr Rizk, and Michael Zink. "SQUAD: A Spectrum-based Quality Adaptation for Dynamic Adaptive Streaming over HTTP". In: *ACM MMSys*. 2016 (cit. on p. 41).

[XSM20] Shichang Xu, Subhabrata Sen, and Z. Morley Mao. “CSI: Inferring Mobile ABR Video Adaptation Behavior under HTTPS and QUIC”. In: *ACM EuroSys*. 2020 (cit. on p. 66).

[Xu+13] Y. Xu, C. Yu, J. Li, and Y. Liu. “Video Telephony for End-Consumers: Measurement Study of Google+, iChat, and Skype”. In: *IEEE/ACM Transactions on Networking*. 2013 (cit. on p. 65).

[Yan+20] Francis Y. Yan, Hudson Ayers, Chenzhi Zhu, Sadjad Fouladi, James Hong, Keyi Zhang, Philip Levis, and Keith Winstein. “Learning in situ: a randomized experiment in video streaming”. In: *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*. Santa Clara, CA: USENIX Association, Feb. 2020, pp. 495–511. ISBN: 978-1-939133-13-7. URL: `https://www.usenix.org/conference/nsdi20/presentation/yan` (cit. on pp. 42, 141).

[Yeo+18] Hyunho Yeo, Youngmok Jung, Jaehong Kim, Jinwoo Shin, and Dongsu Han. “Neural adaptive content-aware Internet video delivery”. In: *USENIX OSDI*. 2018 (cit. on p. 65).

[Yin+15] Xiaoqi Yin, Abhishek Jindal, Vyas Sekar, and Bruno Sinopoli. “A Control-Theoretic Approach for Dynamic Adaptive Video Streaming over HTTP”. In: *ACM SIGCOMM*. 2015 (cit. on pp. 61, 75, 77, 90, 101, 114, 120, 122, 137).

[ZS18] Ondrej Zach and Martin Slanina. “Content Aware Segment Length Optimization for Adaptive Streaming over HTTP”. In: *Radioengineering* 27 (2018), pp. 819–826 (cit. on p. 103).

Melissa Licciardello (she/her)

melissal@ethz.ch @melADTR
https://www.linkedin.com/in/melissa-licciardello-a34393118/

Employment History

2017 – 2022	**Research Assistant, ETH Zurich.** As a PhD Student, I was also employed as Research Assistant in the Systems Group of ETH Zurich, under the supervision of Professor Timothy Roscoe and, previously, under the supervision of Professor Ankit Singla.
2021	**Research Intern, Disney Research.** I have been working at Disney Research in Zurich for three months. My work focused in applying machine learning for rate distortion prediction in video encoding applications.
2017	**Research Fellow, University of Bologna.** I have been working as a research fellow in UNIBO on project HABITAT, financed by the region Emilia Romagna. The project aims to build a suitable environment for aged people in an Internet-of-Things scenario.
2015	**Teaching Assistant, University of Bologna.** I supported Professor Wilma Penzo teaching Java programming language to the students of the first year of the Bachelor Degree of Management Engineering. I was responsible of the practical training of the students.

Education

2017 – 2022	**Ph.D. Computer Science, ETH Zurich.** Thesis advisor: Prof. Dr. Ankit Singla [2017-2021], Prof. Dr. Timothy Roscoe [2021-now] Research area: My research focuses on the intersection between video encoding and video streaming, although my interests span from raw Internet measurements to physical layer communication.
2014 – 2017	**M.Sc. Telecommunication Engineering, University of Bologna**, Networking curriculum Final Grade: 110/110 cum laude Thesis title: *Inter Data Center Communication: performance evaluation over an orchestrator-based architecture*. **The work has been carried on as a joint collaboration between University of Bologna and KTH, Stockholm** Thesis advisor: Carla Raffaelli (University of Bologna), Lena Wosinska (Royal Institute of Technology, Stockholm) Description: Software Defined Networking (SDN) has been recently proposed as a promising solution to dynamically manage the resource provisioning in the network. Orchestrator-based SDN architecture consists of a two layer SDN infrastructure, in which a super entity, called Orchestrator, coordinates the network behavior via the exchange of information with a set of secondary SDN components, called Controllers. Orchestration relies on a set of logical information provided by the controllers related to the underlying infrastructure. Controllers may offer different levels of visibility of available resources, establishing a so called abstraction strategy. Abstraction strategies have a strong impact on both scalability and connection blocking probability of the whole system. In this work, some abstraction strategies previously proposed in literature are analyzed in a distributed data centers scenario and their performances have been evaluated through simulation. Furthermore, a new solution based on weighted virtual links is shown to represent a flexible approach open to further possible improvements.

Education (continued)

2011 – 2014	**B.Sc. Computer Science Engineering, University of Bologna** Final Grade: 110/110 cum laude Thesis title: *Measurement of the charge distribution induced by a dielectric barrier discharge on a dielectric surface for fluid dynamics applications.* Thesis advisor: Prof. Dr. Gabriele Neretti Description: Dielectric-Barrier-Discharge (DBD) consists of an electrical discharge between two electrodes separated by a dielectric barrier. One of its main usage is the production of plasma. A plasma aerodynamic actuator supplied by a multilevel generator operating with different voltage waveforms has been built in order to potentially substitute FLAPs and SLATs in airplanes. This multilevel generator is connected to an Arduino micro-controller, which decides which waveform to submit to the system. Different waveform with different voltages have been shown to have a strong impact over the behavior of the plasma between the actuators. In this work, the DBD effect has been deeply investigated. An Arduino interface has been implemented, together with a waveform code generator, in order to deeply simplify the experimental setup of the multilevel generator.

Teaching Assistant

Spring 2021	Computer Networks, ETH Zurich Future Internet, ETH Zurich
Fall 2020	Systems programming and computer architecture, ETH Zurich
Spring 2020	Computer Networks, ETH Zurich Future Internet, ETH Zurich
Spring 2019	Computer Networks, ETH Zurich
Spring 2018	Computer Networks, ETH Zurich
Spring 2015	Fundamental of Java Programming, University of Bologna

Research Publications

Journal Articles

1. Licciardello, Melissa, Lukas Humbel, Fabian Rohr, Maximilian Grüner, and Ankit Singla. "Prepare Your Video for Streaming with Segue". In: *Journal of Systems Research* 2.1 (July 2022). DOI: `10.5070/SR32158113`. URL: `https://escholarship.org/uc/item/8m39f25q` (visited on 07/21/2022).
2. Borelli, Elena, Giacomo Paolini, Francesco Antoniazzi, Marina Barbiroli, Francesca Benassi, Federico Chesani, Lorenzo Chiari, Massimiliano Fantini, Franco Fuschini, Andrea Galassi, et al. "HABITAT: An IoT solution for independent elderly". In: *Sensors* 19.5 (2019), p. 1258.

Conference Proceedings

1. Cock, David, Abishek Ramdas, Daniel Schwyn, Michael Giardino, Adam Turowski, Zhenhao He, Nora Hossle, Dario Korolija, Melissa Licciardello, Kristina Martsenko, Reto Achermann, Gustavo Alonso, and Timothy Roscoe. "Enzian: An Open, General, CPU/FPGA Platform for Systems Software Research". In: ASPLOS 2022. Lausanne, Switzerland: Association for Computing Machinery, 2022, pp. 434–451. ISBN: 9781450392051.
2. Humbel, Lukas, Daniel Schwyn, Nora Hossle, Roni Haecki, Melissa Licciardello, Jan Schaer, David Cock, Michael Giardino, and Timothy Roscoe. "A Model-Checked I2C Specification". In: *International Symposium on Model Checking Software*. Springer, Cham. 2021, pp. 177–193.

3 Bhattacherjee, Debopam, Simon Kassing, Melissa Licciardello, and Ankit Singla. “In-orbit Computing: An Outlandish thought Experiment?” In: *Proceedings of the 19th ACM Workshop on Hot Topics in Networks*. 2020, pp. 197–204.

4 Grüner, Maximilian, Melissa Licciardello, and Ankit Singla. “Reconstructing proprietary video streaming algorithms”. In: *2020 USENIX Annual Technical Conference (USENIX ATC '20)*. 2020.

5 Licciardello, Melissa, Maximilian Grüner, and Ankit Singla. “Understanding Video Streaming Algorithms in the Wild”. In: *Proceedings of PAM 2020*. 2020.

6 Licciardello, Melissa, Matteo Fiorani, Marija Furdek, Paolo Monti, Carla Raffaelli, and Lena Wosinska. “Performance evaluation of abstraction models for orchestration of distributed data center networks”. In: *2017 19th International Conference on Transparent Optical Networks (ICTON)*. IEEE. 2017, pp. 1–4.

Supervised Students

Bachelor Theses

1 Nguyen, Thai Dai Vu. *Cross-layer Adaptive Bitrate Algorithms in the Puffer player*. 2021.

2 Renggli, Silvan. *Visualization and study on the impact of large numbers of video streams sharing a bottleneck*. 2021.

3 Raun, Christoffer. *Optimization of short video sequences - Building an application layer measurement infrastructure for TikTok*. 2020.

4 Foehn, Valentino. *Bringing Circuit Switching back from the Dead*. 2019.

5 Hug, Josua. *Exploring aggression and fairness in adaptive bitrate video streaming*. 2019.

6 Irani, David Bidjan. *Exploring Quality of Experience in video streaming combining adaptive bitrate and congestion control policy*. 2018.

7 Moser, Florian. *Identifying encrypted online video streams using bitrate profiles*. 2018.

8 Wessels, Dimitri. *Quantifying and Explaining Unfairness in Online Video Streaming*. 2018.

Master Theses

1 Cantieni, Josua. *Battery constrained video streaming*. 2022.

2 Rohr, Fabian. *CAVA on Segue and Segue in DASH: Verifying Simulated ABR Behaviour on the DASH-IF Reference Player*. 2021.

3 Zhou, Haoxiang. *Congestion control reverse engineering - Construct interpretable policies imitating the behaviour of TCP congestion control algorithms*. 2021.

4 Dan, Alexandru. *Cross-layer Adaptive Bitrate Streaming*. 2020.

5 Gruener, Maximilian. *Human-interpretable auto-inference of networked algorithms*. 2019.

6 Perevelli, Stefano. *De-anonymizing Encrypted Video Streams*. 2019.

Semester Projects

1 Joel Lindegger, Sebastian Leisinger. *Videohopping*. 2021.

2 Fizza Zafar, Akshay Jaggi. *SVC Video Encoding and Chunk Enhancement*. 2020.

Interns

1. Islamoglu, Gamze. *Video Streaming with Combined Congestion Control and ABR Algorithms*. 2019.
2. Munaf, Salman. *Exploring buffer size impact in video streaming ABR algorithms*. 2018.

Scientific Presentations

Posters

1. *PhD Workshop on Next-Generation Cloud Infrastructure* at Microsoft Research Cambridge, *invited*. *Prepare your video for streaming with Segue*. 2021.
2. N2Woman. *Prepare your video for streaming with Segue*. 2020.
3. *Systems Group Industry Retreat'20*. *Merging rate selection and rate control in adaptive bitrate video streaming applications*. 2020.
4. *Systems Group Industry Retreat'18*. *Adaptive Bitrate in Video Streaming: a reliable comparison*. 2018.

Scientific Talks

1. *ETHz-Huawei Software Tech Summit*. *Prepare your video for streaming with Segue*. 2022.
2. LEOCONN 2021, *invited*. *In-orbit computing: an outlandish thought experiment?* 2021.

Invited Lectures

1. University of Bologna, Alma Mater Studiorum. *Video streaming internet application: new trends from encoding to delivery*. 2021.
2. ETH Zürich, (various courses). *Everything you always wanted to know about video streaming but you never dared to ask!* 2019-2021.
3. University of Bologna, Alma Mater Studiorum. *Adaptive bitrate in video streaming*. 2018.

Skills

Languages	Strong reading, writing and speaking competencies for English, Italian and French. Currently learning German.
Coding	C, PHP, Python, Java, SQL, XML, Makefile, …
Web Dev	HTML, CSS, JavaScript, Apache Web Server, Tomcat Web Server.
Misc.	Academic research, teaching.

Miscellaneous Experience

Awards and Achievements

2014	**Excellence scholarship**, granted by University of Bologna, for achieving my BSc *cum laude* in less than three years.
2016	**Excellence scholarship**, granted by University of Bologna, to spend a semester in Stockholm to work on my master thesis.
2017	**Lions Club master thesis prize** for innovation in research.

Miscellaneous Experience (continued)

Additional experiences

2006-2010	**Theater education**: for several years I performed and studied dramatic arts (Scuola di Teatro Colli, Bologna).
2007-2017	**Private teacher**: for over 10 years I supported middle school, high school and, later, university students with private lessons on math, physics and latin.

Hobbies

Sports	I'm extremely passionate about acrobatics, and I try to train daily. I also enjoy playing volleyball and biking.
Social activism	I enjoy writing, reading and talking about intersectional feminism. I have been invited to a famous italian podcast to talk about the condition of women in STEM (**link**, italian only).
Crafting	I enjoy sewing, crocheting and knitting clothes. Sometimes I like to add a bit of sparkles adding some micro-controllers to them.
Nerd	I love reading *mangas* (japanese comics), of which I own a pretty impressive collection. Sometimes I find my peace of mind playing some FPS or MOBA games. I definitely know by hearth too many Disney movies songs.