QUALITY OF EXPERIENCE - DRIVEN DYNAMIC ADAPTIVE STREAMING OVER HTTP

by

Ihsan Mert Ozcelik

B.S., Computer Engineering, Bilkent University, 2012

M.S., Computer Engineering, Bilkent University, 2014

Submitted to the Institute for Graduate Studies in
Science and Engineering in partial fulfillment of
the requirements for the degree of
Doctor of Philosophy

Graduate Program in Computer Engineering

Boğaziçi University

2021

ACKNOWLEDGEMENTS

I would like to thank and express my deepest gratitude to my thesis supervisor Prof. Cem Ersoy. My Ph.D. years have had many domestic and international trips between London, Istanbul, Antalya and Bolu while working as a full-time engineer. Under these circumstances and the research problems, his wisdom, courage, and positive thinking helped me overcome all those challenges. So, this thesis would be impossible without his tremendous support, knowledge and vision. Specifically, his inexhaustible energy towards all kinds of problems in 2021 will always inspire me.

I would like to thank my core jury members, Prof. Özlem Durmaz İncel and Prof. Tuna Tuğcu for taking this journey with me over the years, making super-constructive, insightful and valuable comments. I would also like to thank the NETLAB family for their warm friendship and generous support.

And my biggest appreciation to my family for all the support. I am wholeheartedly thankful to my wife Selin for her patience and support over the years, without which I would have stopped my graduate studies multiple times. I would like to thank my sister Ece, my mother Yüksel and my father Mahircan for their invaluable support and love throughout my life. I would also like to thank my parents-in-law Emine and Fikret Gökkuş for supporting and considering me as their own child.

ABSTRACT

QUALITY OF EXPERIENCE - DRIVEN DYNAMIC ADAPTIVE STREAMING OVER HTTP

Dynamic Adaptive Streaming over HTTP (DASH) became the pillar of multimedia content delivery mechanisms in the last decade. Given fluctuating network conditions, over-the-top content platforms struggle with delivering a high quality of experience (QoE) and uninterrupted playback sessions. To overcome this difficulty, they keep multiple quality levels of the same content in a fragmented way. This mechanism enables players to adapt the video quality to varying network conditions by changing the video bitrate at the fragment boundaries. To maximize the QoE, adaptive bitrate algorithms have been widely studied in the literature with promising results. However, the majority of the state-of-the-art solutions do not take into account the presence of multiple DASH clients on the shared bottleneck link, whereas the existing studies considering multiple DASH players in the same network do not consider the diversity of fragment durations among different video titles, background traffic and users' privacy. Those gaps cause QoE fairness and stability problems along with feasibility concerns. First, to address these problems, we propose a centralized module assisted adaptation mechanism with a lightweight Software-Defined Networking (SDN) integration for on-demand video streaming. Second, we leverage our proposed SDN-assisted mechanism to deliver QoE-driven low-latency live event streaming over HTTP. Third, we implement a live streaming DASH client with a novel bandwidth measurement heuristic without requiring any extra component to the legacy systems. It reduces the live delay between the actual event to users' screens down to 1s. As a final contribution, we present a deep reinforcement learning framework to adapt the playback speed and video bitrate to maximize QoE in live streaming.

ÖZET

DENEYİM KALİTESİ ODAKLI HTTP ÜZERİNDEN DİNAMİK UYARLAMALI AKIŞ

HTTP üzerinden dinamik uyarlamalı akış (HDUA), son on yılda multimedya içerik dağıtım mekanizmalarının temel direği haline geldi. Değişen ağ koşulları göz önüne alındığında, üst düzey içerik platformları yüksek deneyim kalitesi ve kesintisiz oynatma oturumları sağlamakta zorlanıyor. Video servis sağlayıcılar bu zorlukların üstesinden gelmek için, aynı içeriğin birden çok kalite düzeyini parçalı şekilde oluştururlar. Bu yöntem, video kalitesini değişen ağ koşullarına uyarlamaya olanak sağlar. Literatürde, umut verici sonuçları olan uyarlamalı video bit hızı algoritmaları yer almaktadır. Fakat, bu çözümlerin çoğu, paylaşılan darboğaz bağlantısında birden çok HDUA istemcisinin varlığını hesaba katmazken, aynı ağda birden çok HDUA oynatıcısını ele alan mevcut çalışmalar ise, parça süresinin çeşitliliğini, arka plan trafiğini ve kullanıcıların gizliliğini dikkate almamaktadır. Bu boşluklar, gerçek hayatta uygulanabilirlik endişeleri ile birlikte birden fazla istemci arasında QoE dengesi ve sürekliliği sorunlarına neden olur. İlk olarak, bu sorunları çözmek için, bir Yazılım Tanımlı Ağ (YTA) entegrasyonuna sahip, merkezi modül destekli bir mekanizma öneriyoruz. İkinci olarak, HTTP üzerinden deneyim kalitesi odaklı canlı etkinlik yayını için YTA destekli mekanizmamızdan yararlanıyoruz. Üçüncüsü, eski sistemlere herhangi bir ekstra bileşen gerektirmeden yeni bir bant genişliği ölçüm yöntemiyle, gerçek etkinlik anı ile o anın kullanıcıların ekranlarında gösterilmesi arasındaki gecikmeyi 1 saniyeye kadar azaltan bir HDUA istemcisi geliştiriyoruz. Son bir katkı olarak, canlı etkinlik yayınlarındaki deneyim kalitesini artırmak için oynatma hızını ve video kalitesini uyarlayan derin pekiştirmeli öğrenme ortamı sunuyoruz.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS

a_t	Action taken at time t
c_0	Coefficient for the contribution of the video quality to QoE
c_1	Coefficient for the penalty factor of video stalls
c_2	Coefficient for the penalty factor of video quality switches
c_3	Coefficient for the penalty factor of live delay
cd_i	The details of the particular chunk i
$et(cd_i)$	The download end time of chunk i
k	Number of quality levels in the media presentation description
r_t	Reward value gained at time t
$s(cd_i)$	The size of the downloaded bytes of chunk i
s_t	State (i.e., the observation of the environment) at time t
t_{avg}	The average of the inter-arrival times between the chunks
$A^{\pi_\theta}(s,a)$	The advantage function
$BT(i)$	The measured background throughput at iteration i
F_{avg}	The average duration of freezes in a session
F_{freg}	The number of freezes relative to the total number of segments
G_t	The expected sum of future rewards
$H(\cdot)$	The entropy component
K	The number of past throughput measurement values
L	List of the download times of all chunks in a segment
$L^{'}$	List of the download times of the network-limited chunks
N	Number of fragments in a session
P	Number of playback speed levels
Q_i	The played quality level for fragment i
$Q(s,a)$	The Q-value function
$SBT(i)$	The smoothed version of $BT(i)$ at iteration i
T_{others}	The total download times of the chunks in $L/L^{'}$
T’	The total download times of chunks in $L^{'}$

$V(s)$	The expected reward before an action is taken in state s
α	Learning rate
α_{actor}	Learning rate of the actor network
α_{critic}	Learning rate of the critic network
β	Exploration factor (i.e., entropy weight)
β_{EPS}	Entropy constant
γ	Discount factor
ϵ	The smoothing factor to calculate the background traffic
μ	The normalized mean quality level
π	Policy as a vector of probabilities of each action
π_θ	A specific policy defined by policy parameters θ
ρ	Approximation coefficient for the encoder-limited chunks
σ	The standard deviation of video quality levels
ϕ	The impact factor of video freezes on the QoE
θ	Policy parameters
$\nabla_\theta J(\theta)$	The gradient of the expected cumulative reward with respect to the policy parameters

LIST OF ACRONYMS/ABBREVIATIONS

4G	Fourth Generation
5G	Fifth Generation
A3C	Asynchronous Advantage Actor Critic
ABR	Adaptive Bitrate
ACM	Association for Computing Machinery
API	Application Programming Interface
ATP	Association of Tennis Professionals
CDN	Content Delivery Network
CMAF	Common Media Application Format
CNN	Convolutional Neural Network
DASH	Dynamic Adaptive Streaming over HTTP
DASH-IF	DASH Industry Forum
DPI	Deep Packet Inspection
DRL	Deep Reinforcement Learning
EWMA	Exponential Weighted Moving Average
GPU	Graphical Processing Unit
HAS	HTTP Adaptive Streaming
HDS	HTTP Dynamic Streaming
HTTP	Hypertext Transfer Protocol
ITU	International Telecommunication Union
IP	Internet Protocol
LTE	Long Term Evolution
MDP	Markov Decision Process
MPD	Media Presentation Description
MPEG	The Moving Picture Experts Group
Mbps	Megabits per second
MSS	Microsoft Smooth Streaming
MVPD	Multi-channel Video Program Distributor

MVQ	Mean Video Quality
NAT	Network Address Translation
NSP	Network Service Provider
OTT	Over-the-top
PANDA	Probe And Adapt
PSNR	Peak Signal-to-Noise Ratio
QoE	Quality of Experience
RL	Reinforcement Learning
ReLU	Rectified Linear Unit
REST	Representational State Transfer
RTT	Round-trip Time
SAND	Server And Network-assisted DASH
SDN	Software-Defined Networking
SLA	Service-Level Agreement
SSIM	Structural Similarity Index Metric
TCP	Transmission Control Protocol
URL	Uniform Resource Locator
VMAF	Video Multimethod Assessment Fusion
VSP	Video Service Provider
VoD	Video-on-Demand
eMOS	Estimated Mean Opinion Score
fps	Frames per second
kbps	kilobits per second

1. INTRODUCTION

There has been a rapid penetration of Internet Protocol (IP)-based Multichannel Video Program Distributors (MVPDs) devices, game consoles, Smart TVs, and smartphones and tablets in the last decade. This increase in IP-connected devices with video rendering capabilities has accelerated advancements in video streaming over the Internet. This trend is reflected as a growing interest in over-the-top (OTT) service platforms and content providers such as Netflix, Amazon Prime Video, Youtube, and Twitch. As of the first quarter of 2021, only two of those providers, Netflix and Amazon Prime Video, have about 500 million subscribers in total all over the world [1]. Unsurprisingly, it makes video content the majority of all Internet traffic. According to Cisco Visual Networking Index prediction, video streaming traffic will be 81% of all Internet traffic by 2022 [2]. As in line with this prediction, the Ericsson Mobility Report forecasts the share of video content over global mobile traffic will be 76% by 2025 [3]. Given highly changing network conditions, considerable diversity in device capabilities and amount of video content distributed over the Internet, satisfying user expectations about high-quality, high-resolution, quickly started, and uninterrupted video service turns out a significant challenge for the content providers.

The crucial success factor for the tremendous growth of video steaming over the Internet in the last decade is the wide adoption of HTTP adaptive streaming (HAS) and its standardization as Dynamic Adaptive Streaming over HTTP (DASH). HAS eases the traversal through NATs and firewalls and allows the use of existing caching infrastructure (e.g., content delivery networks) to scale in a cost-effective manner. In HAS, the multiple versions of the same content at different quality levels are kept at the OTT back-end side. It enables video players to choose the appropriate video quality and change it over time to adapt to the varying network conditions. In other words, adaptive video bitrate selection has become the main knob of OTT applications to maximize the Quality-of-Experience (QoE) of users under fluctuating network conditions.

Adaptive Bitrate (ABR) algorithms aim to provide an uninterrupted playback session by adapting the video quality to the varying network conditions to maximize the users' QoE. A high QoE in video streaming is not only about high video quality and resolution but also a quick session start-up time, a low number of video freezes and video quality switches at the bare minimum. To maximize each particular QoE criterion, ABR algorithms need to trade off with the other QoE criteria. For example, the lower video bitrate the ABR algorithm chooses, the less risk of video stalls users take in case of sudden throughput drops by condemning users to low video quality or resolution. Similarly, the more content the player buffers before starting the playback, the less risk of video freezes the player takes by introducing a disturbingly high video start time. The challenge of ABR algorithms is to balance such trade-offs given the available throughput estimations and play-out buffer status along with the available video bitrates. Competing streaming clients on the same shared bottleneck link exacerbate the problem. Under these circumstances, the purely client-based adaptive video bitrate mechanisms may have QoE fairness and stability problems.

DASH and ABR algorithms were initially designed for the previously-stored videos at multiple quality levels at the content provider site (i.e., Video-on-demand use case), such as movies, TV shows and original series. In the last few years, live event streaming over DASH, as a new use-case, has also been growing in popularity as OTT service providers have procured broadcasting rights of worldwide premium sports events like English Premier League, ATP Tour Tennis, UEFA Champions League, Major League Baseball, American National Football League with 5G infrastructure rolling out more and more. At that point, Dynamic Adaptive Streaming over HTTP is considered the most promising approach for the rapid deployment of the infrastructure of those live sports events due to its existing wide adoption for VoD use cases. As an extra goal on top of the existing success criteria of ABR algorithms in the legacy VoD use cases, a new requirement of low live latency from the actual event to users' screens arises in a live streaming scenario. This tendency is echoed by the recent advancements of the HTTP chunked transfer and the Common Media Application Format (CMAF), which introduce the possibility to deliver a video segment in small chunks before the

full segment is generated. It can provide live latency of three seconds or less on a conventional DASH player with a small buffer capacity less than the target live latency. However, legacy ABR mechanisms inaccurately measure the available bandwidth due to idle times between the video chunks. It is because the download time is affected by not only the current network status but also idle times at the live encoder end as the chunks of the segment are being prepared during the segment download. It misleads ABR decisions and consequently condemns users to a low QoE.

This thesis addresses the above-mentioned issues by (i) utilizing Software-Defined Networking paradigm (SDN) and proposing a centralized joint optimization module-assisted adaptive video bitrate mechanism for VoD use-case while not invading users' privacy and continuously following the available throughput, ii) implementing a bitrate adaptation mechanism for low-latency live event streaming with a lightweight SDN assistance while keeping the coexistence with the legacy DASH clients, iii) presenting live streaming HAS client with a novel bandwidth measurement heuristic over HTTP 1.1 Chunked Transfer of CMAF packages, and iv) presenting a deep reinforcement learning (DRL) framework to maximize QoE for live streaming without any assumption about the environment or fixed rule-based heuristics by considering adaptive playback speed and video quality as two control knobs in a joint optimization problem.

1.1. Thesis Overview and Contributions

In this thesis, we concentrated on DASH systems for both VoD and live event streaming use cases to maximize users' QoE. We addressed QoE fairness and stability problems in the existence of multiple DASH clients and highly fluctuating background traffic on the same shared bottleneck link by using a lightweight SDN guidance. We also took the chunk duration diversity among different titles into account, as opposed to both purely client-based and state-of-the-art SDN-based ABR mechanisms. Furthermore, we extended the same system to support low-latency live streaming scenarios with SDN-assisted central guidance. Moreover, we proposed a novel bandwidth measurement heuristic for low-latency live streaming through HTTP 1.1 Chunked Transfer

of CMAF packages. Finally, we developed a DRL framework to learn video bitrate and playback speed adaptation strategy to maximize QoE for live video streaming without any fixed rule-based heuristics. The main contributions of this thesis are highlighted as follows:

- *Chunk Duration-aware DASH for VOD Using SDN Assistance*: With the help of the global network view provided by the Software-Defined Networking paradigm (SDN), we propose a centralized joint optimization module-assisted adaptive video bitrate mechanism which takes the diversity of chunk duration among different content into account. Our system collects possible video bitrate levels and chunk duration from DASH clients and calculates the optimal video bitrates per client based on the available capacity and chunk duration of each client's selected content. By continuously following the background traffic flows, it asynchronously updates the target video bitrate levels to avoid both buffer stall events and network under-utilization issues rather than bandwidth slicing which brings about scalability problems in practice. It also guarantees fair start-up delays for video sessions with various chunk duration. Our experiments clearly show that our proposed approach considering the diversity of chunk duration and background traffic fluctuations can significantly provide a better and fair QoE in terms of SSIM-based video quality and start-up delay compared to both purely client-based and state-of-the-art SDN-based adaptive bitrate mechanisms [4]. To the best of our knowledge, it is the first study in the literature, that considers the diversity of chunk duration among different video content for a fair and stable QoE across multiple DASH clients within the same shared network.
- *DASH for Low-latency Live Event Streaming Using SDN Assistance*: We design a bitrate adaptation mechanism for live streaming with HTTP 1.1 Chunked Transfer of CMAF packages while keeping the coexistence with the legacy DASH clients. We extend our previously proposed system [4] to achieve a low target latency for live video streaming. Results show that our proposed mechanism provides a better QoE while achieving a live latency within the range of three to six seconds in the existence of varying background traffic.

- *Low-latency HAS Client Using Rule-based ABR Heuristics*: We developed a live streaming HAS client with a novel bandwidth measurement heuristic. We conducted experiments on a real player in bandwidth-limited networks. We showed that our approach followed the live target more closely while achieving a lower video freeze rate (-94%) and higher video quality (+16%) than the existing approaches. Moreover, it allowed reducing the live delay down to 1 s without losing the user's QoE. Furthermore, it is easily applicable to real-life deployments because it does not require any change in the traditional OTT and network backbone [5].
- *Adaptive Playback Speed and Video Quality Selection for Live Streaming Using Reinforcement Learning*: Adaptive video bitrate and adaptive playback speed techniques are two separate control knobs known in the literature of live video streaming. As a novelty, we consider these two control parameters in a joint optimization problem and propose a deep reinforcement learning (DRL) framework in which we represent the state space from the key observations of the player, the action space of the available video bitrates and three playback speed levels, and a reward function in the form of a combined formula of QoE and live latency. We construct a neural network to provide the best action for a given state to map the state space to a joint decision of playback speed and video bitrate levels for the next video segment. In training, we leverage Asynchronous Actor-Critic algorithm (A3C), a state-of-the-art reinforcement learning (RL) algorithm, by introducing another neural network called the critic network, which receives the action taken by the actor and the state space observations to estimate the maximum future award. Simulation results through real network traces show that it outperforms both state-of-the-art DRL-based and rule-based algorithms in terms of QoE without sacrificing live latency.

1.2. Thesis Outline

We organize the thesis as follows: Chapter 2 explains background information while elaborating various use-cases of OTT applications, DASH standard with the

message flow between clients and video servers, and QoE metrics. Chapter 3 presents the related work along with a taxonomy of the previously proposed approaches in the literature. Our main contributions of the thesis are presented in Chapters 4-7.

In Chapter 4, we introduce our SDN-assisted, chunk duration-aware DASH system for VoD use cases. We also confront our approach with the state-of-the-art techniques by implementing all methods in a real DASH player and present our extensive evaluation results.

Chapter 5 presents a low-latency live streaming system over DASH using SDN assistance after highlighting the challenges of live streaming with CMAF and HTTP 1.1 Chunked Transfer. This chapter also includes comparative evaluation results by giving insights on the results.

Chapter 6 explains a HAS client with bandwidth measurement heuristics for live streaming. We also present the evaluation results with real 4G traces by comparing them with the state-of-the-art heuristics.

Chapter 7 describes a reinforcement learning framework that learns a strategy to choose adaptive playback speed and video quality for low-latency live event streaming over DASH. It also presents simulation results to compare our approach with both the state-of-the-art machine learning-based and rule-based approaches.

Chapter 8 concludes the thesis and presents future works.

2. BACKGROUND INFORMATION

2.1. Dynamic Adaptive Streaming over HTTP

HTTP has become a de facto delivery protocol in the application layer for video streaming by content providers at scale since the mid-2000s. As elaborated in [6], it is firstly because the ubiquitous nature of HTTP allows traversing NATs and firewalls while leveraging HTTP to enable to OTT applications to use commodity web servers and caching infrastructure. Secondly, TCP, the underlying transport protocol, brings extra reliability against packet drops and inter-flow friendliness (i.e., fairness) to some extent. Thirdly, it eases scaling up to millions of player clients by leaving the state management logic at the client-side. Since player clients are able to download video fragments without requiring content servers to keep track of the state for each video session, it gives clients the flexibility to switch video destination URLs seamlessly for adaptive video quality selection, and fault tolerance [7]. In other words, this stateless design prevents servers from being bottlenecks. In summary, the use of HTTP in video streaming has arisen with its simplicity and cost-effectiveness.

In the mid-2000s in HTTP video streaming, the early common practice was for players to download the content file at the same video quality into a playout buffer and start playing before the full download. Although this progressive download mechanism over HTTP and a fixed video quality representation had all inherited advantages of HTTP/TCP, it suffered from adapting to different network conditions and device capabilities. Under varying network conditions over time, it used to bring about disturbing video stalls and a consequent low QoE for end-users. To address this, HTTP Adaptive Streaming (HAS) techniques have been adopted since the late-2000s. In HAS, the multiple quality versions of the same content at the different encoded bitrates are kept at the server-side. Each quality version is called representation, in which content is stored as a series of fragments, each including short-duration content (e.g., 1-15 seconds of video). Fragment duration and boundaries are the same among all quality versions

of the same content. A manifest file for each content includes available video quality levels, total content duration, chunk sizes and URLs for each corresponding quality level and timestamp. Since players download the manifest file at the beginning of each session and fragments are aligned between one version to other versions of quality levels in the playback timeline, the player can seamlessly switch different quality levels at the fragment boundary over time within the same session. This flow between DASH clients and the server is illustrated in Figure 2.1 in which the same content is stored at k quality levels in a fragmented way, and the first two fragments are started at the lowest quality. In this example, ABR algorithm bumps the quality up to the next video bitrate after the initial two fragments. This illustrative figure does not include CDN nodes for simplicity though these fragments are distributed into CDN nodes from the origin servers in practice.

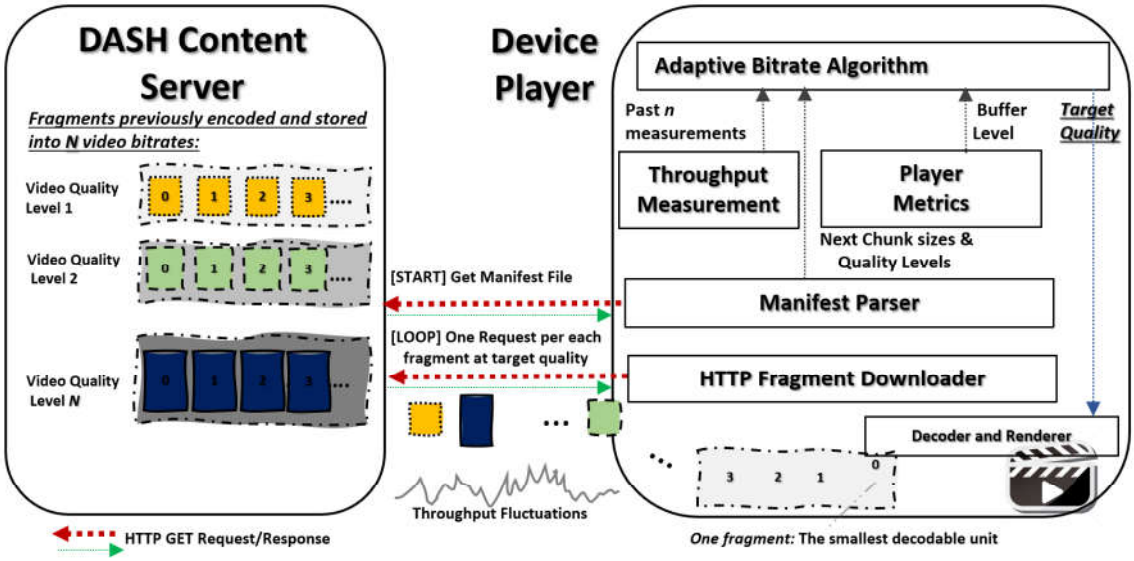


Figure 2.1. DASH Flow between player and content server.

To standardize the manifest file format, multimedia representation and fragmentation details in HAS, Moving Picture Experts Group - Dynamic Adaptive Streaming over HTTP (MPEG-DASH) was published in 2012 as an International Electrotechnical Commission standard [8]. Note that the MPEG-DASH standard does not dictate

any ABR mechanism (i.e., adaptive video quality selection rule) as it just defines the guidelines for the manifest file in an XML format, multimedia representation and segmentation as depicted in Figure 2.2 in which it represents four video quality levels encoded at {250, 500, 1000, 2000} kbps. In addition to DASH, there are other alternative commercial formats in HAS to define the same guidelines between clients and servers. Microsoft's Smooth Streaming (MSS) [9], Apple's HTTP Live Streaming (HLS) [10], and Adobe's HTTP Dynamic Streaming [11] exemplify alternative formats in HAS.

Figure 2.2. Sample DASH Manifest file.

In this thesis, we use DASH and HAS interchangeably. In our testbed during all our experiments, we also keep content manifestation files in DASH format, although all our proposed techniques are applicable to the other HAS standards such as HDS, HLS and MSS.

2.2. Quality-of-Experience Metrics in Video Streaming

In this section, we define QoE and explain the terminology about various QoE metrics in video streaming that are later used throughout the thesis. As presented by [12], QoE in video streaming is the measurement of annoyance or delight of the user

of a streaming application or service as the end users' experience and expectation.

Understanding what factors influence perceptual QoE in a particular playback session is key and a prerequisite to come up with any adaptation technique in HAS as quality adaptations are aimed to improve users' experience. QoE influence factors in HAS are comprehensively surveyed in [13]. These are grouped into three categories, which are waiting times, the perceptual impact of adaptation, and video quality. First, waiting times consist of initial start-up delay, stalling frequency and stalling duration. Second, the perceptual impact of adaptation is implied by the number of video quality switches during the same session. Third, the most complicated factor, the perceived video quality refers to the quality of the encoded video stream regarding spatial resolution, image quality, and temporal scalability. The simplest proxy for the video quality is the encoding video bitrate. As the video bitrate (i.e., the number of bits to present the same-duration content) increases, the perceived quality increases, although it is not a simple linear correlation. To measure the perceived quality more explicitly, the other approaches leverage either objective mathematical models or estimated quality scores generated by the user feedback-based subjective studies. Netflix's Video Multimethod Assessment Fusion (VMAF) [14], Structural Similarity Index Metric (SSIM) [15], and Peak Signal-to-Noise Ratio (PSNR) [16] perfectly exemplify the objective methods to measure the video quality. Both VMAF, SSIM and PSNR compare the encoded video with the original reference mezzanine files over different features.

In HAS, there are numerous trade-offs between various QoE influence factors. The trade-off between the video quality and the waiting times is one example. Because downloading higher quality content requires more time, it increases the risk and the duration of video stalls. Another example is the trade-off between the number of video freezes and the number of quality adaptations because frequent changes in video quality just to avoid stalling events bring about a negative perceptual impact of adaptation. In the nature of such trade-offs, holistic QoE models are proposed to enable multi-dimension adaptation to maximize the overall QoE rather than a single QoE factor. In other words, these models introduce algorithms to estimate the impact of different QoE

influence factors into a single holistic metric as defined by ITU-T Recommendation P.1201 [17]. As extensively investigated in [18], the most prominent QoE modeling approach is to run a regression analysis to map multiple QoE influence factors into an estimated mean opinion score (eMOS) over previously collected subjective study results. In this thesis, we use the estimated Mean Opinion Score (eMOS) [19] to reflect the overall QoE perceived by end-users in our evaluation results. It is calculated by combining the different QoE metrics into a single metric in the linear model as

$$eMOS = 5.67 * \mu - 6.72 * \sigma - 4.95 * \phi + 0.17, \tag{2.1}$$

where μ and σ denotes the normalized mean quality level and the standard deviation of video quality levels, respectively as

$$\mu = \frac{\sum_{i=1}^{n} \frac{Q_i}{k}}{n} \quad \text{and} \quad \sigma = \sqrt{\frac{\sum_{i=1}^{n} (\frac{Q_i}{k} - \mu)^2}{(n-1)}}. \tag{2.2}$$

ϕ represents the impact of video freezes on the QoE during the session with n segments, k quality levels and the played quality level Q_i for segment i. ϕ denotes a continuous interpolation of the levels of video freeze frequency and duration to measure the effect of video freezes on QoE, where F_{freq} and F_{avg} indicate the number of freezes relative to the total number of segments and the average duration over all freezes respectively as

$$\phi = \frac{7 * \max\left(\frac{\ln F_{freq}}{3} + 1, 0\right) + \frac{\min\left(F_{avg}, 6\right)}{6}}{8}. \tag{2.3}$$

In this thesis, we rely on such holistic QoE models to be able to improve and measure the users' QoE through a single metric as the combinations of multiple QoE influence factors.

3. LITERATURE REVIEW

This chapter provides a detailed survey on the recent adaptive video bitrate solutions and low-latency live streaming solutions over HTTP.

3.1. Related Work on Adaptive Video Bitrate (ABR) Algorithms

Adaptive bitrate mechanisms in DASH have been widely studied in the literature to provide a better QoE. Our literature survey classifies those mechanisms into two categories in terms of the deployment component, such as client-side and server-assisted solutions, as summarized in Figure 3.1. Our literature review in this thesis mostly discusses the recent studies that focus on network and server-assisted DASH mechanisms to address the stability in video quality, scalability and fairness issues in the presence of many DASH players on the shared bottleneck link as our proposed approaches in Chapter 4 and Chapter 5 are within this category. The detailed survey of initial studies about the conventional adaptive bitrate heuristics in DASH, which do not leverage a central coordinator and network assistance, can be found in [6] and [43].

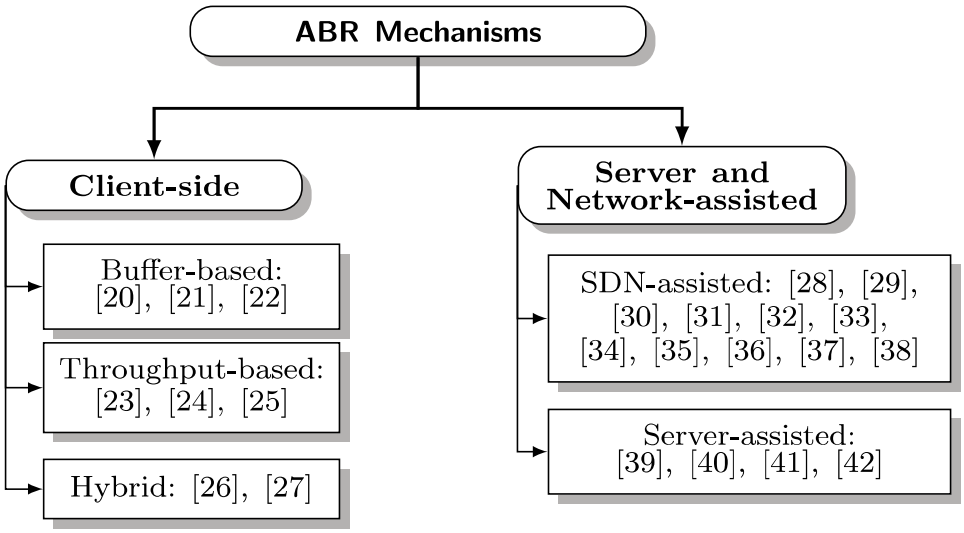


Figure 3.1. Classification of Adaptive Video Bitrate solutions in terms of the deployment component and input criteria.

We also explain the existing ABR strategies used in the DASH.js player [27], which is widely used in both industry and academia in addition to a few purely-client-based conventional ABR solutions.

3.1.1. Server and Network-Assisted Solutions

Thomas et al. [39] propose the Server and Network-assisted DASH (SAND) architecture, which points out the need for a bi-directional messaging plane between clients and other DASH-aware Network Elements. Using these asynchronous messages, it aims at giving a capability to measure the real QoE of ISP subscribers in a central authority for QoS reporting and returning network status information from the core network to clients to be used in their ABR heuristics. Although how to implement the messaging mechanism between network elements and the controller is not defined and privacy and scalability aspects are not discussed in this study, it concludes that the optimal quality for a streaming session is reached 10 times faster when DASH players benefit from SAND messages in estimating the available bandwidth to choose the video quality level.

In [28] and [29], Kleinrouweler et al. propose an SDN-assisted central manager to signal target video bitrates to players. However, their experiments clearly reflect that using only this approach causes unfairness in the presence of multiple DASH players. The reason of this unfairness problem is that their proposal allows players to ignore the target bitrate received from the manager and rely on player's local bandwidth estimations. To resolve this unfairness issue, they also propose and implement a kind of bandwidth slicing on network elements with dedicated queues which are specific to DASH players. Considering the number of configurable queues is limited on switches, the scalability problem for the real-life deployment is clear in this study. Nevertheless, it proves that using SDN-based assistance increases the average video bitrate quality and reduces the number of switches in the quality level.

Bentaleb et al. [30] propose an external SDN-based resource management module called as SDNDASH to address fairness and quality stability problems. SDNDASH periodically selects the optimal quality and bitrate level for each client in a centralized manner and respectively performs per-client resource allocation in the network. The communication channel between DASH players and the central control module is based on HTTP REST APIs. Hence, the central module is not capable for asynchronously sending commands to players. Additionally, the optimization problem in SDNDASH does not take the device display heterogeneity into the account. Furthermore, their experiment results show that SDNDASH does not scale well because of the communication overhead and attempting per-client optimization. To resolve all these problems, the same authors propose SDNHAS [31] with the help of per-cluster optimization approach instead of per-player optimization, in-band communication approach with no extra signaling overhead and adding device resolution, player status and type of the requested content to the optimization problem. As the in-band communication channel between player and the central module, SDNHAS inspects and modifies HTTP headers in the standard messages for the content and the manifest file between DASH players and the DASH server by using a commercialized deep packet inspection solution. However, it leads to privacy concerns and how to implement it for the encrypted content is an open issue. In SDNHAS, the offline clustering algorithms initially group the players based on the device resolution and subscription plan; and then perform per-cluster optimization where the players in the same cluster will receive the equal bandwidth. Considering that each content has different discrete quality levels even if its type is the same, the output of the per-cluster optimization heuristics cannot be close to the available bitrates in some of the players in the same cluster. Hence, it may cause underutilization issues. Despite these drawbacks, their simulations based on Mininet with 100 DASH players show that SDNHAS's scalability as the highest number of clients in the evaluations of the related studies. Similarly, the same authors in [44] apply SDN-based bandwidth allocation scheme for DASH flows with a lower communication overhead in hybrid fiber coax networks.

Cofano et al. in [32] and [33] focus on the implementation, classification and comparison of SDN-based network-assisted strategies in the simultaneous presence of several adaptive video streaming flows transmitted via a shared bottleneck link, which are bitrate guidance of the video control plane and bandwidth reservation. By periodically solving the max-min fairness problem on the video control plane, their study aims at achieving video quality fairness among multiple DASH players. The period for each run of the optimization module is 30 s while the segment size of the test content is fixed to 4 s in the study. Because they do not take the segment duration into the account, it means that their heuristics cannot be responsive for the consecutive 7-8 chunks. This study also does not consider the background traffic. Furthermore, while implementing the bandwidth slicing approach, the switch used in the testbed is limited to eight different queues. As like the issue in [29], this limitation can bring about scalability problems in practice. Despite the possible scalability issues and temporary unresponsiveness problem, their results indicate that the bitrate guidance of the central controller module provides the best results in terms of video quality fairness, whereas bandwidth allocation improves the average video quality depending on the client-side ABR mechanism.

Bagci et al. in [34] investigate the video quality fairness problem in the existence of multiple DASH players by proposing both centralized and distributed architectures for collaboration between network service provider (NSP), video service provider (VSP), and DASH players with the help of SDN. Compared to the previous studies, using a distributed architecture as an alternative to the centralized solution can be considered a novelty, which requires inter-client collaboration by eliminating the need for the central optimization module. However, they do not evaluate the communication overhead and the additional processing cost for each player. As another contribution, this study proposes modifying the TCP receive window (*rwnd*) behavior based on the observation that even if the player has a dedicated queue for the video flow, the player can have the video quality level switches because of the TCP *rwnd* fluctuations and its reset. This modified behavior sets *rwnd* size to an optimal value specified by the fair-share bitrate mechanism to provide a stable TCP goodput.

In [35], Georgopoulos et al. propose an SDN-assisted QoE fairness framework that monitors the status of all DASH applications in a network and dynamically allocates fair network resources. It can be evaluated as the first study in the literature which includes a central optimization logic to find the optimum set of bitrates that ensures QoE fairness across all DASH clients in the network. This study considers the issue as a max-min fairness problem. In the implementation, the central module parses media presentation description (MPD) files requested by the DASH players to retrieve the target bitrate list and informs the central optimization module. For the encrypted MPDs, it is not feasible in practice. Finally, even though their testbed is very small with only three concurrent flows, their real-life implementation with home network devices indicates that this approach is useful for the better QoE in terms of the fairness across heterogeneous devices in a network.

In the proposed architecture SABR [36], Divyashri et al. investigate an SDN-assisted bitrate guidance approach while, at the same time, using the content offloading into CDN and optimizing the utilization of the CDN caches. The central module in SABR periodically computes the available bandwidth forecasts to each cache and the players get all caches' status with the estimated bandwidth values for each segment at each quality level as a map for each cache. This status information is used in the player to be able to make an educated guess to decide the next video bitrate. Although the overhead caused by retrieving the cache status information is not analysed in the study, roughly it brings about *[(number of segments) * (number of caches) * (number of quality levels) * 4 bytes]* overhead per each status response in a streaming session. Considering this communication overhead and processing cost in clients, its feasibility in practice remains an open question for a large-scale deployment.

In [40], a lightweight network coordination is used to achieve quality fairness in the existence of multiple clients by collecting the aggregate statistics across clients and pushing it to all clients to be used on the client-side ABR heuristics. It requires a modification on the legacy manifest file by adding explicit video quality information. Periodically collecting per-fragment quality and buffer occupancy information from

clients has a negative influence on the communication overhead. Specifically short-duration fragments exacerbate the problem concerning the overhead.

Jingyan et al. in [38] propose an online reinforcement learning based algorithm to resolve the unfairness problem among multiple DASH clients with the help of dynamic network resources allocation to guide the bitrate selection in the client through rate limiting in OpenFlow switches according to the controller strategy. Their proposal relies on the argument that SDN provides a deep packet inspection (DPI) function for the packets filtered by the HTTP protocol via OpenFlow. However, the highest layer protocol header that can be tracked is transport layer and OpenFlow does not define any way to extract the payload itself. Even though DPI-based customized implementations can allow extracting HTTP payload, how to handle this for the encrypted content remains a significant challenge in this study.

Altamimi et al. in [41] introduce a server-side video bitrate adaptation solution to achieve the fairness of QoE among concurrent DASH users under the same shared network. The authors leverage reinforcement learning by defining a reward function representing fairness based on Jain's index. They also define the state space with the packet loss at the receiver side and the current video bitrates as the input values for decision. Their action space for the output consists of the available video bitrates. They use two neural networks as the standard Actor-Critic network in the training process. Their solution requires changing legacy DASH servers because they manipulate the media presentation description (MPD) files at the server-side by pruning the available bitrates higher than the target quality chosen by their actor network. In this way, they impose specific bitrates on clients because clients cannot be aware that higher quality levels actually exist via modified MPD files.

Another server-client cooperation model is presented by Marai et al. in [42] to improve the bandwidth utilization and fairness without affecting the stability of players through adaptive video bitrate decisions. The proposed approach aims to achieve a fair bandwidth share on the bottleneck link at the server-side among multiple clients

via their server-side controllers. Their proposed client-side controllers update the video bitrates to manage the stability of the playout buffer using the measured throughput and the buffer occupancy, while the server-side controllers converge to a fair bandwidth share. Their implicit assumption is that video streaming clients can fully consume the server-side available bandwidth, which is not the case in practice as there can be multiple bottlenecks on the path down to the clients at the edge. Additionally, since over-the-top (OTT) applications (e.g., Netflix, Prime Video and Youtube) utilize CDNs with highly large egress capacities at the server-side today, the recent literature focuses more on the bottleneck links in the core network or the local links at the edge rather than the server-side bottleneck links studied in this study.

3.1.2. Client-based Solutions

Purely client-based ABRs are more conventional approaches as more aligned with the initial motivation of the distributed design in DASH systems since players are able to choose the video quality independently by relying on their own local knowledge. Solutions in this category can be classified into three groups in terms of the input types they benefit from in the algorithm, such as playout buffer occupancy, throughput, and hybrid of both throughput and buffer level.

Buffer-based solutions rely on the remaining video data in the playout buffer as the feedback signal. Based on static or adaptive buffer-bounds, they map the current buffer occupancy to video quality level. For example, when the buffer occupancy is low, they simply choose lower video bitrates, whereas they go to a higher quality level in higher buffer occupancy values. One of the first studies that uses only the current buffer occupancy for the adaptive video bitrate selection in the steady-state without any capacity estimation is presented by Huang et al. in [22]. They demonstrate through millions of real video players in a commercial service that their purely buffer-based solution reduces the video freeze rate by up to 20% compared to Netflix's default ABR mechanism without sacrificing the average video quality. As another well-known purely buffer-based approach, Spiteri et al. introduces BOLA in [20] for the adaptive

quality selection without any available throughput estimation. The authors represent the ABR problem as a utility-maximization problem to maximize the video bitrate and minimize the video stalls using Lyapunov optimization. Moreover, Huang et al. in [21] present Stick framework that uses deep reinforcement learning to output the buffer-bound values to map the buffer occupancy to video bitrate outputs. So, although they use only the current buffer level at the final phase of the adaptive bitrate selection, they utilize the past throughput values in the state space of their proposed neural network in outputting buffer-bound values, unlike conventional purely buffer-based ABRs. In their reinforcement learning formulation, the reward is a simple linear QoE formula similar to Equation 2.1 we previously explained.

Throughput-based solutions use the past throughput measurements and the future capacity prediction as the main input signals for deciding the video quality to maximize QoE. FESTIVE [23] introduced by Jiang et al. exemplifies one of the prior studies that predicts future throughput using the throughput measurement values of the past fragment downloads. FESTIVE simply uses the harmonic mean of the measured throughput values during the previous twenty fragment downloads and then performs a gradual bitrate switching strategy with a delayed update. These gradual quality switches give players convergence time to a fair throughput use in the same shared network while avoiding frequent quality oscillations. Hence, it achieves both fairness, stability and efficiency without any noticeable computation overhead. Moreover, Li et al. [24] point out the problems of the throughput-based solutions due to the incorrect bandwidth predictions in the existence of multiple clients on the same bottleneck link. The authors highlight the challenge for a client to reach its fair-share throughput when it does not attempt to fetch higher video bitrates along with TCP receive window fluctuations and resets. To address this challenge, they present a Probe And Adapt (PANDA) technique to improve network utilization to eventually achieve its fair-share throughput as a kind of the application-layer equivalent of TCP congestion control. PANDA bumps up the video quality to the target level until it detects congestion via the reduction of the measured throughput. Furthermore, Qiao et al. in [25] investigate the accuracy of the throughput prediction approaches in mobile networks and find out

that the conventional prediction techniques like harmonic mean and simple moving average perform poorly in mobile video streaming, whereas environment-specific Markov property should be considered in the video bitrate selection in mobile networks. The authors, then, propose network environment identification based video bitrate adaption for the future throughput prediction by utilizing the environment-specific Markov property after training an offline Markov model using the traces from each environment. However, the practicality of such a solution given a huge number of environment types for real-life deployment remains an open question.

Finally, the hybrid use of both buffer occupancy and throughput as the input signals in adaptive video bitrate decisions is very common in the industry. DASH.js player perfectly exemplifies such hybrid models by allowing the customization and the combination of various throughput and buffer-based solutions. It is a common element in the testbed of most of the references explained above as a very popular reference player implementation in the literature. It is an open-source implementation by DASH Industry Forum [27]. We also benefit from DASH.js in our testbed and performance evaluation. The ABR logic in DASH.js includes a list of rules which are considered before every new chunk is downloaded. Each one returns a maximum bitrate level, and the minimum output is decided as the final requested video bitrate. There are mainly two primary rules to drive the ABR mechanism in the player: *abrBola* and *abrThroughput* [26]. The player is forced to choose either of these two primary rules. *abrBola* is implemented based on [20], which proposes using the buffer occupancy level to choose the bitrate by forcing lower bitrates when the buffer level is lower and higher bitrates when the buffer level is higher. As the other primary rule, *abrThroughput* is a baseline ABR logic, which uses the recent throughput history to predict the future throughput. Once it has a throughput estimate, the rule scales it down by a 90% safety factor and determines the highest bitrate, which does not exceed the safe throughput. While the primary rule is the main factor in DASH.js, there are various secondary rules to support the primary rules, which are $InsufficientBufferRule$, $SwitchHistoryRule$, $DroppedFramesRule$ and $AbandonRequestRule$. $InsufficientBufferRule$ provides forcing the lowest bitrate if the buffer stalling happens by switching the ABR mecha-

nism to the conservative mode. *SwitchHistoryRule* prevents the player from having bitrate oscillations for better stability by disallowing the quality switch for the consecutive eight segments. *DroppedFramesRule* does not allow any higher bitrate when the players drop more than 15% of the frames at some bitrate because it considers the frame drops as a CPU bottleneck problem. *AbandonRequestRule* provides abandoning the download request and switches to a lower bitrate by monitoring the downloading progress to observe whether there exists a dramatic drop in the estimated bandwidth and a rebuffering risk. DASH.js allows implementing custom rules on top of the already existing mechanisms to extend and/or modify its ABR algorithm. It also gives the opportunity to disable all built-in rules. In our testbed in this thesis, we benefit from this feature by implementing our custom rule, which fetches the video bitrate decision from our central stream tracker modules detailed in Chapters 4 and 5.

3.2. Related Work on Low-latency Live Streaming over HTTP

The studies explained in Section 3.1 about the on-demand video streaming do not consider the requirement of low live latency from the actual event to users' screens, which is critical for live event streaming. Hence, recent additional efforts have focused on reducing latency in HTTP live video streaming without sacrificing QoE as the popularity of live content providers such as Twitch, YouTube Live, and Amazon Prime Video has been rapidly growing. In this section, research works in both academia and industry on low latency and QoE in live video streaming over HTTP are explained.

There is a large group of existing work on reducing end-to-end delay in HTTP live streaming. El Essaili et al. [45] presented a prototype with 33 ms fragments using the chunked transfer to reduce latency without taking into account optimal video bitrate selection and measuring the overall QoE. Van der Hooft et al. [46] introduced a new low-latency approach for live streaming based on HTTP/2's push feature and super-short segments. It reduces end-to-end latency within the range of eight to ten seconds. In contrast, it brings about longer video freeze times compared to the legacy HTTP/1.1 approaches due to encoding overhead of super-short segments and conventional pull-

based ABR deficiencies in HTTP/2 push-based models. Ben Yahia et al. [47] described another HTTP/2-based approach by proposing frame discarding to achieve low-latency video streaming. With the same aim, Wang et al. [48] designed a new ABR algorithm called *BitLat* for the Live Video Streaming Grand Challenge [49] in which there exists a frame-level live video simulator for participants to implement ABR algorithms and benchmark. *BitLat* benefited from reinforcement learning to provide adaptive latency limits and control playback rate for frame skipping to achieve a target latency.

Figure 3.2. End-to-end Flow of HTTP Chunked Encoded Transfer of CMAF segments.

The straightforward way to reduce the live delay is to shorten the segment duration. However, it brings about numerous drawbacks such as a dramatic increase in the number of HTTP requests and responses, a few multiples of the relevant round-trip times, diminished visual quality, and excessive rates of switches between different video quality levels. Is there any solution to resolve the latency problem of live streaming over DASH at scale without sacrificing QoE of end-users? As demonstrated in [50], it is possible based on HTTP 1.1 chunked encoding transfer and the MPEG CMAF that was the recently introduced media container standard without having the overhead of additional HTTP request-response pairs. A segment in CMAF consists of multiple small pieces called chunks, i.e., the smallest decodable units. With the HTTP chunked transfer, it enables to distribute segments by chunks (e.g., even 100ms content) while keeping all main advantages of DASH systems such as quality switching at segment boundaries, leveraging the caches in content delivery networks (CDNs), only one request for each segment and firewall friendliness. Fig. 3.2 depicts the end-to-end flow of HTTP chunked encoding transfer of CMAF packages in which chunks are encoded and packaged by the adaptive bitrate encoder and packager to different quality levels and

then immediately transferred to the live origin chunk-by-chunk for the distribution over CDNs. HTTP 1.1 Chunked Encoded POST standard is used to deliver chunks from the encoder to the edge servers in CDNs, while players on a large variety of devices pull chunks through HTTP 1.1 Chunked Encoded GET at a specific video quality chosen by their ABR algorithms.

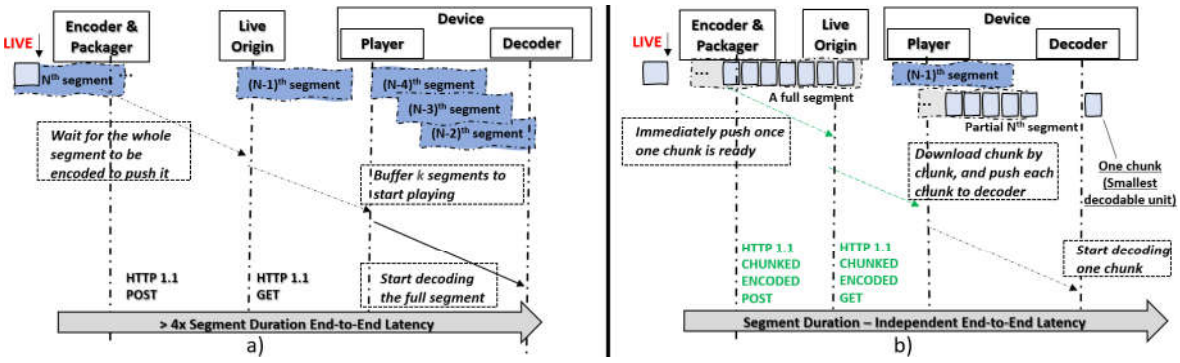


Figure 3.3. a) Legacy Segment Distribution, b) HTTP 1.1 chunked encoding CMAF for live streaming.

Fig. 3.3 reflects how the HTTP Chunked Encoded transfer of CMAF segments achieves a better low latency against the conventional DASH systems for live streaming. The encoder in the legacy system waits for a full segment to be encoded to be able to push to the live origin making at least one segment duration delay in the content generation. When adding the delay of the buffered segments in players, the time difference between the moment when the live event is rendered at the client device and when it happens can go to above four times the segment duration. On the other hand, since one HTTP request can gradually deliver multiple CMAF chunks on the fly, chunks can be posted to the edge servers without waiting for the full segment, while players can be simultaneously pulling chunks of the CMAF segment via HTTP chunked encoding transfer and immediately starting decoding once even one chunk is ready on the device.

The HTTP Chunked Encoded transfer of CMAF segments brings extra complexities to conventional ABRs. First, the buffered content duration should be short (e.g., less than a few seconds) to limit the delay from the actual event to users' screens.

It prevents buffer-based solutions from achieving a high QoE due to frequent video quality changes because buffer occupancy bars to increase or decrease the quality are extremely close to each other. In other words, legacy buffer-based ABR solutions are not effective in low-latency live streaming. Second, the core logic of the throughput-based adaptive bitrate algorithms, throughput estimation logic, estimates the available bandwidth as almost equals to the encoded bitrate in live streaming over chunked encoded transfer because they simply divide the segment size by the download time as pointed out in [51] when the chunks of the segment are being prepared during the segment download. Therefore, it clearly prevents players from switching up the quality due to underestimating the available bandwidth because of the idle times between chunks. There is theoretically no exact way in the standard to determine the total idle times to be able to calculate the accurate download rate by subtracting idle times from the total download time of the segment.

There are relatively new workarounds for the ABR problems in HTTP 1.1 chunked encoding CMAF packages. In *ACTE* [51], the available bandwidth is estimated by implementing a sliding window and disregarding the download rate of chunks not close to the moving average to measure the chunk-level bandwidth. Similarly, the recent DASH.js [27] version (i.e., *v*3.1.2) by DASH Industry Forum ignores small-size chunks and all the chunk download rates lower than the average download rate of the previous chunks in the same segment. The filtering process removes the download times of those outliers in the throughput measurement, although it still considers them in the total segment size calculation. While these workarounds resolve the bandwidth underestimation problem, they induce overestimating the available bandwidth and video freezes, especially in bandwidth-limited networks. *LoL* (Low-on-Latency) [52] also emphasizes the impact of the encoder-side idle times on the inaccurate throughput measurement and introduces a novel throughput measurement module by parsing the chunk payloads to identify chunk boundaries based on the headers (i.e., *moof* box) in fragmented MP4 data. It is aimed at determining the idle periods explicitly using chunk boundary identifications. It perfectly addresses the throughput overestimation problems, whereas it can go in the opposite direction with the underestimation. As a follow-up study,

Bentaleb et al. in [53] also propose *LoL+* to enable the wide deployment of *LoL* by introducing a novel adaptive playback speed control mechanism and configurable QoE objectives rather than the singular QoE. Furthermore, we presented a purely client-based heuristic to measure the available bandwidth by approximating the idle times at the encoder side and the active download times of all the chunks limited by only network status in live streaming over DASH [5]. Moreover, Sun et al. in [54] use deep reinforcement learning to choose the adaptive video bitrate and playback speed in HTTP live streaming. Even though the bandwidth measurements are not accurate, their model adapts to the inaccuracy of the raw measurements to maximize QoE, while it also skips the content to catch up with the live event in case live latency is increasing. The main difference of our proposed SDN-aided mechanism in Chapter 5 compared to these workarounds is to directly use the available bandwidth precisely in choosing the appropriate video bitrate and be highly responsive to all sudden network fluctuations via a central network view, rather than any client-side assumptions or heuristic about the idle times between chunks and inconsistent throughput estimations.

4. CHUNK DURATION-AWARE DASH FOR VOD USING SDN ASSISTANCE

In this chapter, we explain our SDN-assisted, chunk duration-aware DASH system for VoD use cases, which are for previously stored multimedia files ready to be served on demand.

4.1. Introduction

In DASH, each encoded version at different qualities is divided into small chunks, each including short duration content as elaborated in Chapter 2. Each chunk can be considered as the smallest decodable video stream. Chunk duration and boundaries are the same among all versions of the same content. Therefore, chunks are aligned between one version to other versions of quality levels in the video time line so that the player can smoothly switch different quality levels at the chunk boundary. However, chunk duration changes depending on the OTT provider and content. Because the displaying order and the encoding order can highly differ based on the encoding profiles, player needs to decode the whole chunk to be able to render the first frame of the video content. Hence, ignoring the chunk duration diversity and downloading the initial fragments at the same quality level among different clients with different chunk-duration content negatively affects the fairness and the user friendliness in terms of start-up delays.

The decision to choose the video quality level during and at the beginning of playback relies on a player's selection in a fully isolated manner. This selection is the output of the adaptive bitrate algorithms which conventionally run on the client-side as explained in Section 3.1.2. Simply, the output of the ABR algorithm is based on the current buffer occupancy, device capabilities, and the current available throughput with the player's local view of the network. Consequently, a purely client-driven adaptation logic can bring about unstable playback sessions, frequent video stalls, network utilization problems and/or selfish decisions, which cause an unfair and disturbing QoE

for the other players in the same shared network. In the same manner, a more complicated ABR logic which requires more resources in terms of memory and processing can be infeasible especially for low-end devices to be implemented on the client side.

To address the problems due to the bitrate adaptation logic fully managed by clients, a central controller which has a global view of the network and DASH players can be useful. At that point, as an enabling technology, Software-Defined Networking (SDN) paradigm can be a perfect match to develop a centralized controller for DASH players. Although the state-of-art SDN-based mechanisms surveyed in Section 3.1.1 obviously demonstrate the benefit of SDN assistance to a better and fair video QoE among multiple clients, none of them considers diversity of the chunk duration. All recent SDN-assisted ABR approaches commonly leverage a target bitrate signaling where the optimal bitrate for each player is collaboratively computed and informed to the players. These studies prove the positive impact of SDN assistance to ABR mechanisms on QoE and fairness. On the other hand, they mostly have open issues in terms of practicality and real-life deployment such as privacy violation, considerable communication overhead and incompatibility with legacy systems. In summary, none of them takes the chunk duration diversity into account.

SDN is proposed to decouple data and control planes in network devices with a standardized traceability and programmability of the data plane and centralized view of the global network as the current emerging technology for core networks [55]. SDN includes three main functional layers, which are infrastructure layer, control layer and application layer. The infrastructure layer includes forwarding elements also known as data plane while the control layer provides control and monitor functionality via open APIs. By using the northbound interface to communicate with the control and monitor functionality of the controller, an SDN-assisted adaptive bitrate logic in a central manner can be proposed to satisfy user expectations about high-quality, uninterrupted and quickly started video service by offloading ABR computations from clients to a central control module.

In this chapter, we investigate the integration of a privacy-preserving SDN-assisted central ABR mechanism with DASH players to achieve high quality of experience for end users while avoiding network under-utilization and unfairness problems in the existence of the multiple DASH players and other background traffic on the same bottleneck link. Additionally, since our proposed mechanism effectively and continuously follows the background traffic fluctuations, it is highly responsive for dynamic network conditions without requiring any bandwidth allocations between background and DASH flows. Hence, it can be evaluated as a lightweight central assistance without causing any scalability issues in practice. In the same manner, it does not require any modifications in the legacy DASH content servers and manifest files. Moreover, considering each player's sharing only target video bitrate list, chunk durations and total duration of the video content with the controller at the beginning of the playback session, our mechanism does not invade the users' privacy because it does not have the information of what a user is watching. Furthermore, our proposed model is also compatible with legacy DASH players by taking their existence into the account as background traffic.

We implemented our proposed SDN-assisted approach by slightly modifying the existing DASH.js player [27] and locally establishing a DASH content server with various chunk-size content. For performance evaluation, we used the Mininet [56] environment by mimicking various network conditions and different number of DASH players and background flows in the shared bottleneck link. We show that our SDN-based central DASH stream controller implementation exposing chunk duration of each individual DASH flow and traffic fluctuations of background flows clearly outperforms the existing purely client-based and the state-of-the-art SDN-assisted mechanisms in terms of a fair QoE based on the average video quality, the time to first frame (i.e., start-up delay), and number of video stalls at the expense of a few number of switches among all players.

4.2. Architecture of The SDN-assisted DASH

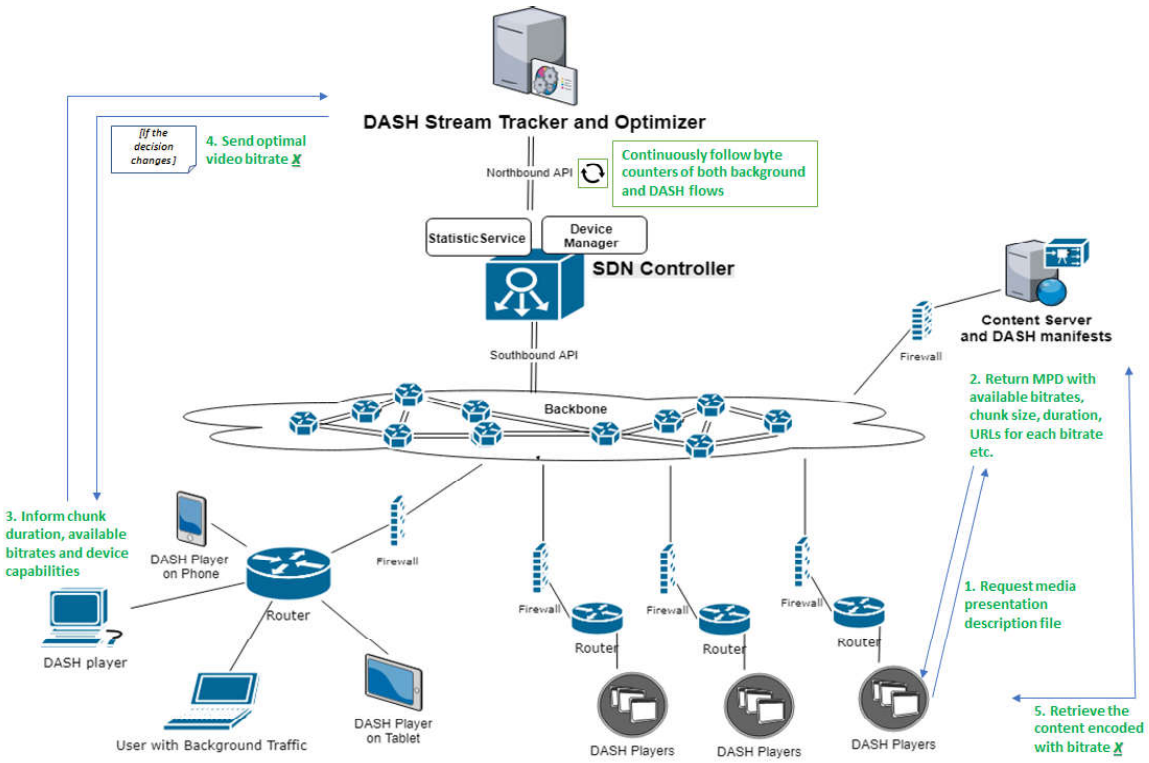


Figure 4.1. System Architecture of SDN-Assisted DASH.

As depicted in Figure 4.1, our proposed system consists of 4 layers: DASH Stream Tracker and Optimizer (STO) as a northbound application, an SDN Controller, OpenFlow-enabled forwarding planes and DASH players on different-capability devices which can communicate with the proposed central DASH flow tracker and optimizer module. The system also includes conventional DASH content and manifest servers, which keep MPDs as manifest files for each content and video content encoded at different quality levels. A manifest file for each content includes available quality levels, total content duration, chunk sizes and URLs for each quality level and each chunk.

All messages between DASH players and DASH servers are based on REST APIs over HTTP as designed in the legacy systems. In other words, our proposed model does not require any changes on the standard DASH manifest servers. As different from the conventional systems, it requires an out-band bidirectional communication channel between the players and STO. This two-way communication is provided over

*websocket*s. It does not violate one of the main advantageous of the DASH that enables clients to reach DASH server sitting behind firewall and NAT because *websocket* is a widely implemented web technology which uses the Port 80 to make the connection appear to the HTTP server [57]. Finally, STO communicates with the SDN controller via the standard northbound APIs while the controller communicates with the networking elements via OpenFlow.

When a user attempts to play a specific DASH content, our proposed system works as follows:

(i) The player requests the media presentation description file which corresponds to the desired content from the DASH manifest server.

(ii) The server returns the MPD which includes video bitrate levels with resolution information, total content duration, chunk duration for the requested content.

(iii) The player sends the target video bitrate lists, total and chunk duration in addition to its display capabilities. In this step, user's privacy is not violated, and the client device does not share the information of what a user is watching.

(iv) STO returns the output of the QoE optimization heuristics for the video bitrate level to the player. Each time the decision is changed, new video bitrate level is pushed to the player. On the background, STO collects network statistics to determine the available capacity on the shared bottleneck link. Leveraging these statistics and all active DASH players' information about available video bitrates, it runs the optimization heuristics to specify a bitrate level for each player while considering QoE fairness among all DASH players. Because the system effectively follows the background traffic flows, it is highly responsive for network fluctuations. The central optimization heuristics are explained in Section 4.3 in a detailed manner

(v) Once the video bitrate level is retrieved from the central optimizer, the DASH player fetches the content encoded at that bitrate level.

The main design principle of the proposed system is simplicity for not causing challenges in possible real-life deployment scenarios. First, the proposed system does not require any active network programming for explicit per-client bandwidth allocation on bottleneck links. It is helpful for not only a simple design but also the overall network utilization, background and legacy DASH client flows. Second, it just collects video bitrate list, total and chunk duration from players once at the beginning of each session and does not require client feedback messages anymore during the playback. From the central controller to a particular DASH player, the target video bitrate is sent only when the output of the QoE optimization heuristics for that player is changed due to any considerable network fluctuations or arrival/exit of DASH players. Hence, the proposed system does not induce any noticeable overhead because of the out-band bidirectional communication channel between the players and STO. Thirdly, it does not expose any deep packet inspection approaches to sniff manifest files to extract video bitrate alternatives and the relevant video metadata. Therefore, it is totally compatible with encrypted manifest files over HTTPS. Finally, there is no modification on the legacy DASH servers, content and manifest files. In the same manner, as a fall-back mechanism due to any failure in the communication with the DASH STO explained in Step 3 and Step 4, DASH players continue to work with their purely client-based conventional ABR mechanisms. With the help of this fall-back mechanism, we aim to recover any failure and unresponsive state in the system.

4.3. SDN-assisted central ABR heuristics

In this section, we elaborate our proposed SDN-assisted central ABR mechanisms integrated with the northbound application shown in Figure 4.1.

4.3.1. Chunk-duration aware, start-up delay optimizer among multiple DASH clients

We first study the fairness of start-up delays among multiple DASH clients with various chunk duration. By start-up delay, we mean the difference between the times

customer clicks on the play button and sees the first frame on the screen. It is affected by the download time and the processing time of decoding and rendering on platforms. In this section, we do not focus on the processing time because it is much smaller compared to the download time and not noticeably affected by the chunk size.

To render the first frame of the chunk, the player needs to download the whole chunk because the display order is not same with the decoding order, and decoding the first frame might require the last frames depending on encoding. Consequently, the diversity in chunk duration among clients affects start-up delays in a significant manner.

When the playback is started with the lowest quality, the start-up delay is smaller considering the fact that downloading initial fragments is faster because the size of the low-quality fragments is much smaller. However, it condemns users to the lowest quality in the very beginning of the playback during the initial chunks. In this trade-off, our heuristics relies on customer or OTT app provider preference-based maximum start-up delay value and chunk duration to compute the initial video bitrates of multiple DASH clients in the same shared networks. The threshold of the start-up delay can be assumed as a service-level agreement (SLA) value.

Algorithm in Figure 4.2 targets the optimal initial quality level for each DASH player while keeping the start up delay smaller than the SLA threshold set by OTT app provider or user preference. The algorithm starts with the lowest quality and incrementally increases the quality level until the download time required to fetch the video fragment at this quality level exceeds up to the target start-up delay. At the end of each client iteration, the optimal initial quality value is sent to the client and the last communication time with the client is updated with the current real time. This is repeated for each client. Even though the chunk duration varies among multiple clients, the algorithm aims at keeping start-up delays closer to each other for the sake of a better and fair QoE among the clients.

Input: Total capacity, list of n DASH clients with video quality levels and chunk duration

Output: Initial quality level for each DASH player

$i = 0$;

$perDeviceCapacity = totalCapacity$ / n;

repeat

  if $(client_i[lastCommTime] == 0)$ **then**

    $currLevel_i = 0$;

    $estimatedDownloadTime_i = 0$;

    repeat

      $estimatedFragmentSize_i = chunkDuration_i *$

       $videoBitrate[currLevel_i] * SLACK\_FACTOR$;

      $estimatedDownloadTime_i =$

       $estimatedFragmentSize_i / perDeviceCapacity$;

      $currLevel_i = currLevel_i + 1$;

    until $estimatedDownloadTime_i >$

     $TARGET\_MAX\_START\_UP\_DELAY$;

    $client_i[lastCommTime]$ = currTime;

    Send $currLevel_i$ to $client_i$ as the target video bitrate level;

  end

until $i >= n$;

Figure 4.2. Start-up delay optimizer algorithm.

Input: Total capacity, list of n DASH clients with video quality levels and chunk duration

Output: Next quality level for each DASH player

$i = 0$;

$perDeviceCapacity = totalCapacity$ / n;

repeat

  if *(*$client_i[lastCommTime] > 0$ *)* **then**

    $currLevel_i = 0$;

    $estimatedDownloadTime_i = 0$;

    repeat

      $estimatedFragmentSize_i = chunkDuration_i *$
      $videoBitrate[currLevel_i] * SLACK\_FACTOR$;

      $estimatedDownloadTime_i =$
      $estimatedFragmentSize_i / perDeviceCapacity$;

      $currLevel_i = currLevel_i + 1$;

    until $estimatedDownloadTime_i > chunkDuration_i$;

    if *(*$client_i[currLevel]$ *!=* $currLevel_i$*)* **then**

      $client_i[currLevel] = currLevel_i$;

      Send $currLevel_i$ to $client_i$ as the target video bitrate level;

      $client_i[lastCommTime]$ = currTime;

    end

  end

until $i >= n$;

Figure 4.3. Video quality optimizer algorithm during playback.

4.3.2. Chunk-duration aware, video quality optimizer among multiple DASH clients

If players would continue with the initial video quality computed in Algorithm in Figure 4.2 based on the principle that downloading time per fragment should be less than $TARGET\_MAX\_START\_UP\_DELAY$, players with various chunk duration

would obviously have unfair video quality values during playbacks because players with smaller chunk sizes would have better video quality while the ones with larger chunk sizes would have lower video bitrates. Similarly, if $TARGET\_MAX\_START\_UP\_DELAY$ is much smaller than the largest chunk duration in the content catalog, a low quality is chosen for the initial fragment of the players with large chunk duration to be able to hit the start-up delay SLA value. To resolve the unfairness and under-utilization problem, we propose Algorithm in Figure 4.3 to update video bitrates during playback.

As very similarly to Algorithm in Figure 4.2, Algorithm in Figure 4.3 simply chooses the optimal video quality level so that the download time is not greater than the fragment duration. The motivation is to provide the highest video bitrate without causing any video stalls. Because each previous video quality level per each client is cached, the next run of the algorithm returns the same quality level per a particular client and it is not shared with the client. Hence, the communication channel between players and our STO is efficiently used to minimize the additional communication overhead in the system.

4.3.3. Background traffic-aware, QoE optimizer among multiple DASH clients

The algorithms we have explained so far do not take background traffic into account. Hence, in practice, considering there is mostly varying background traffic, these algorithms are not sufficient to avoid buffer stalls. In order to avoid video stalls while providing a better and fair video quality to users, we also propose an SDN-assisted central mechanism to track all background traffic flows on the bottleneck link. Algorithm in Figure 4.4 is the combined version of the algorithms in Figure 4.2 and Figure 4.3 by replacing *totalCapacity* with the available capacity (that is, *totalCapacity* - *smoothedBackgroundBw*) while tracking all background traffic flows. We periodically run this algorithm in our central optimization module. It is also aware of the active and ended playback sessions. Therefore, the proposed mechanism is highly responsive to both the background traffic fluctuations and the possible increasing available network resources due to the arbitrarily ended playbacks.

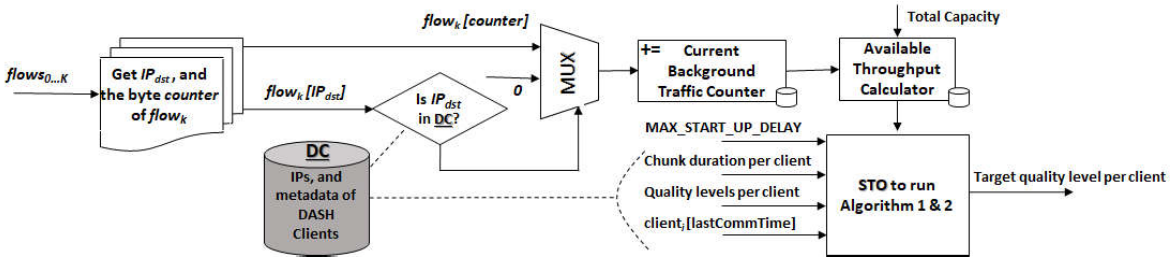


Figure 4.4. Block diagram of the background traffic-aware, QoE optimizer.

Algorithm in Figure 4.5 exposes TCP fairness by expecting available bandwidth being evenly shared among DASH clients in the long run. In cases such as highly varying round-trip time (RTT) values among the clients, re-initiating TCP connections frequently, and very different initial values of CWND, relying on TCP fairness does not work properly. However, our approach focuses on the clients on the same bottleneck in the shared network at the edge. Hence, the clients will be geographically close to each other and consequently their RTTs will not be very different. Second, we expect that all DASH servers of the same OTT application (e.g., Netflix) have the same initial CWND values considering these servers are consistently configured by the same business logic. Even if the OTT app uses the CDN servers from a third-party stakeholder, the stakeholder will also have the same configuration for the initial CWND size value among all its CDN servers. Third, in the case of the persistent HTTP connection with *keep-alive* option, we observe that consecutive HTTP requests for the consecutive chunks are going through the same TCP connection. Considering the player sends the requests with this way, we do not encounter re-initiating TCP connections very frequently. Based on the three points above, TCP fairness makes sense in our case. Even in the worst case due to the different initial CWND size values or connection re-establishments, it eventually provides sharing the bandwidth evenly among the clients.

Figure 4.4 depicts the block diagram and sequence in our central optimization module. Because each DASH player connects to the central module via a separate communication channel, IP addresses of connected DASH players are known and kept in a hash set of the DASH clients (DC). When each player stops streaming, it closes the connection and it is removed from DC store. Additionally, statistics of each flow

are retrieved with the help of SDN. In Algorithm in Figure 4.5, accepting these values, the total background traffic is calculated by summing the used bandwidth of all flows not in the hash set of DASH player IP addresses at any time. The historical values of used bandwidth by background flows are kept in a sliding window. It stores the last M values. The algorithm relies on the average of these values in the window to smooth the estimated background traffic to avoid higher number of quality switches.

The smoothed background traffic is based on Exponential Weighted Moving Average (EWMA) [58] as

$$SBT(i) = (1 - \epsilon) \times SBT(i - 1) + \epsilon \times BT(i), \tag{4.1}$$

where $BT(i)$ and $SBT(i)$ are the measured background throughput and its smoothed version at the iteration i respectively, and ϵ is the smoothing factor. For ϵ, we select the mostly used value, $\frac{2}{M+1}$ as the simple moving average. This value is helpful to minimize fluctuations in video quality although it causes reacting with more latency to network fluctuations especially for the larger M values. To be able to react much faster to highly varying background traffic, alternatively an adaptive smoothing factor can be used depending on the current trend of the measured throughput value. A sample application of such a dynamically selecting smoothing factor can be seen in Sobhani et al. [59].

As the final step, the available capacity for DASH players is simply calculated by subtracting the smoothed background traffic from the total capacity of the bottleneck link.

Input: Total capacity, byte counters of all flows ($flows$) on the bottleneck link, list of n clients, the background traffic values ($fifoQueueOfBackgroundBwValues$)

Output: Next quality level for each DASH player

$i = 0$, $k = 0$,$currBackgroundBw = 0$;

for *each* $flow_k$ *in* $flows$ **do**
  if $flow_k[dstIP]$ *is not in the list of IP addresses of DASH clients* ***AND*** $flow_k.getByteCount() >$ $flow_k[lastByteCount]$ **then**
    $timeDiff = flow_k.getDurationSec() - flow_k[lastDuration]$;
    $flow_k[currBw] = (flow_k.getByteCount() - flow_k[lastByteCount]) * 8/timeDiff$;
    $currBackgroundBw = currBackgroundBw + flow_k[currBw]$;
    $flow_k[lastDuration] = flow_k.getDurationSec()$;
    $flow_k[lastByteCount] = flow_k.getByteCount()$;
  end
end
if $fifoQueueOfBackgroundBwValues_{size} < M$ **then**
  add $currBackgroundBw$ to $fifoQueueOfBackgroundBwValues$;
  $sum_{fifoQueueOfBackgroundBwValues} += currBackgroundBw$;
end
else
  $oldestBwValue = fifoQueueOfBackgroundBwValues.remove()$;
  add $currBackgroundBw$ to $fifoQueueOfBackgroundBwValues$;
  $sum_{fifoQueueOfBackgroundBwValues} = sum_{fifoQueueOfBackgroundBwValues} + currBackgroundBw - oldestBwValue$;
end
$smoothedBackgroundBw = sum_{fifoQueueOfBackgroundBwValues}/M$;
$perDeviceCapacity$ = ($totalCapacity$ - $smoothedBackgroundBw$) / n;
repeat
  $currLevel_i = 0$, $estimatedDownloadTime_i = 0$;
  if $(client_i[lastCommTime] == 0)$ **then**
    repeat
      $estimatedFragmentSize_i = chunkDuration_i * videoBitrate[currLevel_i] * SLACK\_FACTOR$;
      $estimatedDownloadTime_i = estimatedFragmentSize_i/perDeviceCapacity$;
      $currLevel_i = currLevel_i + 1$;
    until $estimatedDownloadTime_i > TARGET\_MAX\_START\_UP\_DELAY$;
  end
  else
    repeat
      $estimatedFragmentSize_i = chunkDuration_i * videoBitrate[currLevel_i] * SLACK\_FACTOR$;
      $estimatedDownloadTime_i = estimatedFragmentSize_i/perDeviceCapacity$;
      $currLevel_i = currLevel_i + 1$;
    until $estimatedDownloadTime_i > chunkDuration_i$;
  end
  if $(client_i[currLevel] != currLevel_i)$ **then**
    $client_i[currLevel] = currLevel_i$;
    Send $currLevel_i$ to $client_i$ as the target video bitrate level;
    $client_i[lastCommTime]$ = currTime;
  end
until $i >= n$;

Figure 4.5. Background traffic-aware, QoE optimizer algorithm.

4.4. Performance Evaluation

In this section, we perform extensive experiments to compare our proposed mechanism with the purely client-based and the state-of-the-art SDN-assisted mechanisms in terms of QoE.

4.4.1. Testbed and Implementation Details

To implement th chunk-size aware SDN-assisted DASH ($CSASDN$) architecture as illustrated in Figure 4.1, we use DASH.js [27] for the players, Floodlight [60] for the SDN controller, Mininet [56] for the data plane, Firefox [61] for the browser and $iperf$ [62] for generating background traffic. Because Mininet end hosts run the TCP/IP stack of the Linux kernel, our setup considers TCP slow-start behavior as the same with the real-life deployment.

We first expose the *CustomRule* interface of the DASH.js reference player for the ABR decision so that it shall be able to communicate with STO via a *websocket*, which allows a two-way asynchronous communication. After the connection is established, we disable all built-in ABR rules in the player to evaluate the performance of $CSASDN$. On the STO-side, we attach a *websocket* server on top of the Floodlight controller, which has a bidirectional communication channel to each player. Third, our network topology is generated by Mininet using Open vSwitches [63]. The testbed includes 8 Mininet-based DASH players competing for the 30Mbps total shared capacity in the same bottleneck link. We run each DASH player in a new instance of the Firefox browser on an Ubuntu machine with a quad-core CPU running at 2.60GHz and 16GB RAM. To avoid the CPU bottleneck in the testbed, we benefit from the blank decoder capability of the Firefox browser by setting **use-blank-decoder** to true. We also run our $iperf$-based background traffic generator script in Mininet hosts for highly varying time intervals with a given maximum traffic amount. To mimic a fluctuating background traffic, both traffic amount and time are randomized while the total background traffic is capped with the given input value.

In our DASH server, we store two different types of content, namely *Big Buck Bunny* and *Of Forest And Men* with various chunk-size versions such as 2, 6, 10 and 15s [64]. The total duration of the *Big Buck Bunny* content is 596 seconds encoded using H.264 at 20 different quality levels {*47Kbps, 92Kbps, 138Kbps, 186Kbps, 232Kbps, 278Kbps, 367Kbps, 464Kbps, 555Kbps, 646Kbps, 829Kbps, 1Mbps, 1.3Mbps, 1.5Mbps, 1.8Mbps, 2.1Mbps, 2.5Mbps, 3.2Mbps and 3.7Mbps*}. Similarly, *Of Forest And Men* is 453 seconds encoded using H.264 at 19 quality levels {*46Kbps, 89Kbps, 128Kbps, 177Kbps, 218Kbps, 255Kbps, 321Kbps, 474Kbps, 506Kbps, 573Kbps, 780Kbps, 1Mbps, 1.2Mbps, 1.5Mbps, 2.1Mbps, 2.4Mbps, 2.9Mbps, 3.3Mbps, 3.6Mbps and 3.9Mbps*}. Because we have 8 DASH players in the experiments and the maximum available representation is about 3.7Mbps, we set the total capacity in the simulations to 30Mbps.

In this testbed, we compare our proposed approach $CSASDN$ with the other ABR heuristics such as $BOLA$ as a purely client-based ABR [20] and the $ExplicitSDN$ as a state-of-the-art SDN-based approach [29]. $ExplicitSDN$ is implemented as a northbound application by dividing the bandwidth evenly among the players and informing these values to the players. For BOLA, we use the default implementation in DASH.js. Both competitors are also strengthened by DASH.js built-in secondary rules. Our evaluation in terms of the QoE metrics explained in Section 6.3.2 is based on the following four categories:

(i) Impact of different target start-up delays
(ii) Impact of light and heavy background traffic
(iii) Impact of the number of background flows
(iv) The case of the players with arbitrary arrival/departure times

4.4.2. The Metrics

In order to evaluate QoE among different approaches, we benefit from various metrics such as the perceptual video quality, number of switches in the quality, number of stalls and start-up delays.

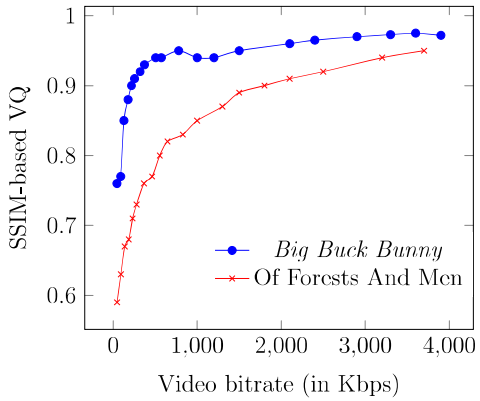


Figure 4.6. The measured SSIM values for different video bitrates of Animation and Documentary content.

For the perceptual video quality, we rely on the $StructuralSIMilarityIndex$ metric (SSIM) [15] as an objective method. We select this metric since it is widely used in the literature and it is successful to represent the non-linear relation between the perceived video quality and its bitrate level. The measured SSIM values of *Big Buck Bunny* and *Of Forests And Men* for different video bitrates are shown in Figure 4.6. As depicted in the figure, though the video bitrate gradually improves the video quality, the increasing pattern is not linear. Moreover, the correlation highly changes depending on the content. For example, the perceptual quality difference among video bitrates is much more visible in $OfForestsAndMen$ whereas the quality values are very close to each other especially for higher video bitrates in *Big Buck Bunny*.

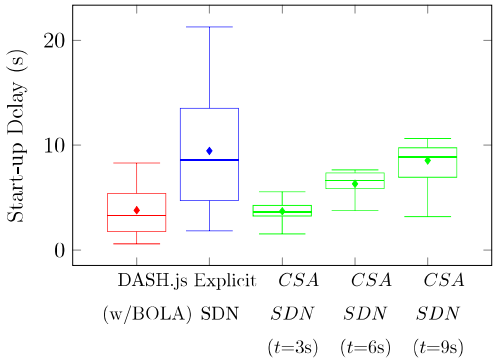
To measure the average video quality of a particular session, we use the following metric as the average of SSIM-based quality values of each chunk during the session with n chunks as

$$VQ_{Session} = \frac{\sum_{i=1}^{n} SSIM_{chunk_i}}{n}. \tag{4.2}$$

As the factors with the negative impact on QoE, we also follow the number of stalls and the number of quality level switches during the session. The number of video stalls is simply the number of occurrences when the playback buffer is empty while the number of the switches is the total number of quality drops or jumps during the session. In our

performance evaluations, we also use the estimated Mean Opinion Score (eMOS) [19] to better reflect the overall QoE perceived by end users. It is calculated by combining the different QoE metrics into a single metric in the linear model provided in Equation (2.1).

4.4.3. Impact of different target start-up delays



	Avg. Initial Video Quality
Dash.js	0.952
Explicit SDN	0.957
CSASDN **DASH (with t=3s)**	0.948
CSASDN **DASH (with t=6s)**	0.957
CSASDN **DASH (with t=9s)**	0.961

Figure 4.7. Start-up Delay Comparison between *CSASDN* DASH and state-of-the-art ABRs.

We begin with the performance comparison in terms of start-up delays between our proposed approach (*CSASDN* DASH) and other ABR heuristics such as *BOLA* and *ExplicitSDN*. *Big Buck Bunny* content is used for all experiments in this subsection with the setup configuration explained in Section 5.1.

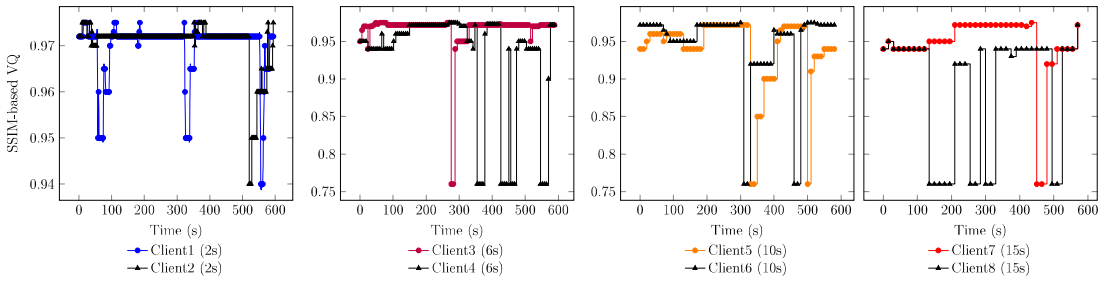
First, we evaluate the impact of different $TARGET\_MAX\_START\_UP\_DELAY$ on the start-up delays in our proposed approach. As the SLA value set by OTT app provider or users, we choose three different values: 3s, 6s and 9s. With a different input value of $TARGET\_MAX\_START\_UP\_DELAY$, each experiment is repeated five times and we collect start-up delays from 8 different clients with various chunk duration in each experiment. The mean values in the box plot in Figure 4.7 clearly show that the measured start-up delays in our proposed approach converge to the specified target values despite the diversity of chunk duration and sizes. The reason of the slight difference between the exact target SLA value and the real start-up delay is that the nominal bitrate represented in the video manifest files and the real encoded

bitrate does not fully match because of the variable bitrate encoding. Additionally, as expected, the initial video quality increases as the target start-up delay increases because the algorithm choose a better quality level when a larger download time is given as the constraint.

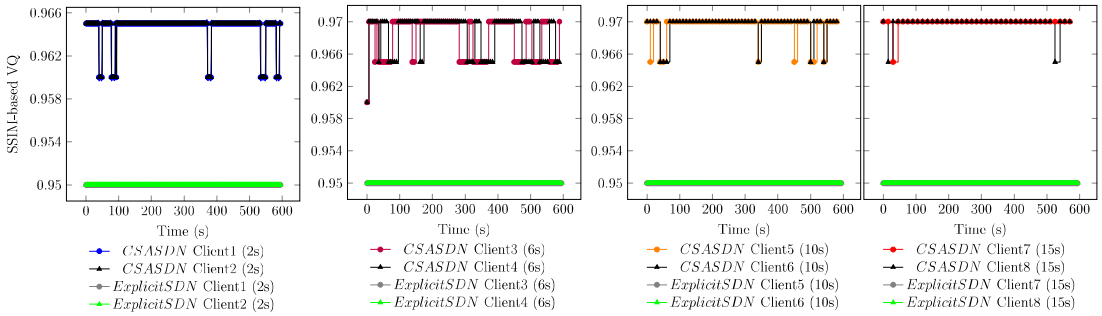
Second, start-up delays are measured by applying the same methodology for the other heuristics. Because they do not take the chunk size variability into account and download the initial fragments with the same video bitrate levels, they can bring about extremely high start-up delays (e.g., about 21 seconds). As depicted in Figure 4.7, the other heuristics have a very high variance of start-up delays since the larger fragments have larger delays while the smaller ones have much smaller start-up delays. The results also reflect that there is no noticeable video quality difference in the first fragments between $CSASDN$ and the other heuristics whereas it provides better and fair start-up delay values.

4.4.4. Impact of light and heavy background traffic

We conduct experiments to observe the overall QoE metrics over time during the whole video session for different clients with various chunk duration for different background traffic scenarios. Based on the experiment results, we compare the performance of $CSASDN$ with $BOLA$ and $ExplicitSDN$ in terms of the QoE metrics. We benefit from both *Big Buck Bunny* and *Of Forest And Men* content with the setup configuration explained in Section 5.1. In each scenario, in total, there exists 16 clients in the same shared network as 8 DASH clients and 8 background flows.



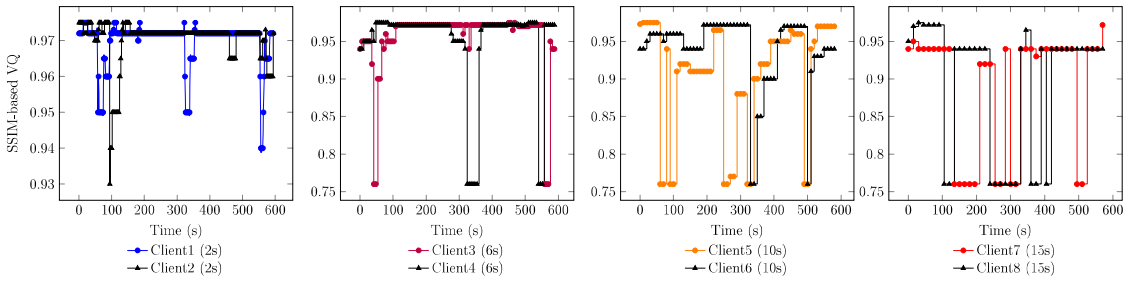
(a) BOLA



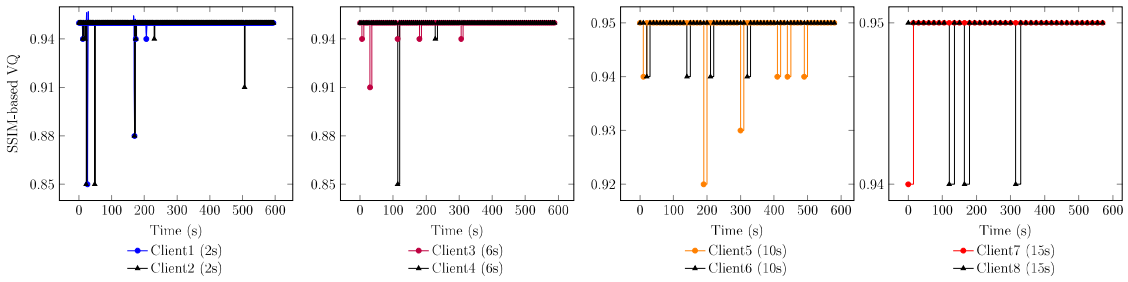
(b) *CSASDN* and *ExplicitSDN* experiments

Figure 4.8. Video quality during 10-min *Big Buck Bunny* session among 8 clients with *BOLA*, *CSASDN*, *ExplicitSDN* in the existence of light background traffic (*rand*(0.5..2) Mbps).

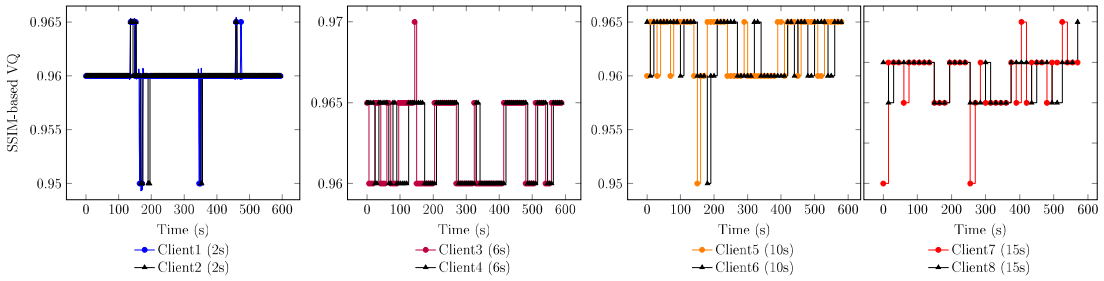
First, we follow the video quality values over time for each session streaming *Big Buck Bunny* content under a very light background traffic randomly generated by 8 background flows. The total background traffic amount is capped to 2Mbps. In each 10min experiment, we commonly apply a different ABR mechanism to all 8 DASH clients with various chunk duration. Figure 4.8 shows video quality values on each client over time for *BOLA*, *ExplicitSDN*, and *CSASDN*. Figure 4.8(a) reflects video quality values in case of *BOLA* while Figure 4.8(b) depicts values from both *CSASDN* and *ExplicitSDN* experiments. *CSASDN* explicitly outperforms *BOLA* and *ExplicitSDN* in terms of video quality at the expense of a few more switches compared to *ExplicitSDN*. The existence of multiple DASH players in the same network dramatically affects BOLA by causing very low quality especially for larger chunk sizes in spite of the frequent switches as reflected in Figure 4.8(a).



(a) Clients with *BOLA*



(b) Clients with *ExplicitSDN* assistance

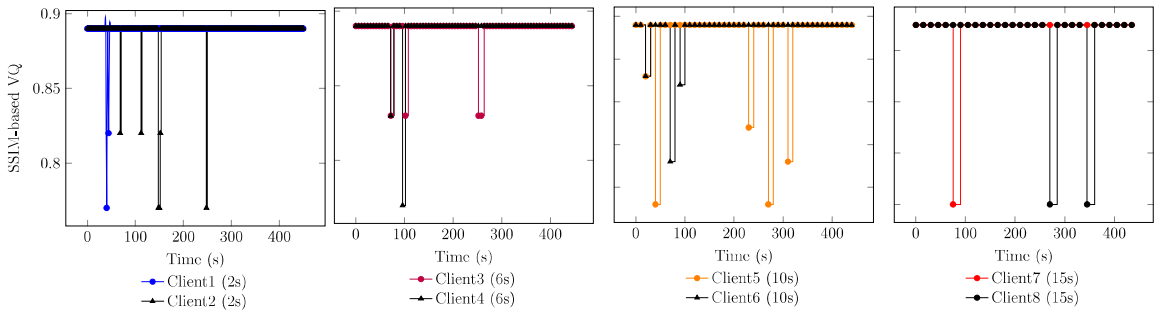


(c) Clients with *CSASDN*

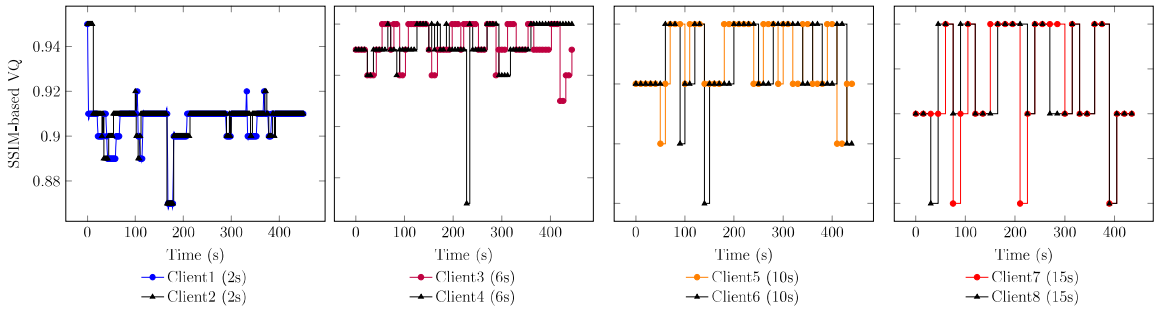
Figure 4.9. Video quality during 10-min *Big Buck Bunny* session among 8 clients with *BOLA*, *ExplicitSDN*, and our proposed *CSASDN* approach in the existence of heavy background traffic (*rand*(2...10) Mbps).

Second, we apply the same procedure under a more realistic background traffic scenario by generating a random traffic between 2Mbps and 10Mbps for random intervals. Figure 4.9(a), 4.9(b) and 4.9(c) depict video quality values over time for *BOLA*, *ExplicitSDN*, and *CSASDN*, respectively. The fairness problem is obviously observed with the purely client-based ABR mechanism *BOLA* because a client suffers from the lowest vido quality while,at the same time, some other clients stream with the highest video quality as reflected in Figure 4.9(a). Because *ExplicitSDN* does not follow the background traffic fluctuations, its central assistance always informs DASH players about the same quality level. However, when the background traffic increases, the buffer levels in the DASH players drop and the DASH.js players switch to their

built-in mechanism with the conservative mode by progressively choosing the lowest available bitrates. Hence, as shown in Figure 4.9(b), clients with *ExplicitSDN* can have hard quality drops in the existence of a heavy background traffic. In parallel, as the background traffic decreases, *ExplicitSDN* does not update the quality decisions to increase. So, *CSASDN* is better in terms of the video quality over time by actively following the background traffic and triggering the quality level switches.



(a) Clients with *ExplicitSDN* assistance



(b) Clients with *CSASDN*

Figure 4.10. Video quality during 8-min *Of Forest And Men* session among 8 clients in the existence of heavy background traffic (*rand*(2..10) Mbps).

Third, by replacing the content in the setup with *Of Forest And Men*, we compare the video quality values over time between *ExplicitSDN* and *CSASDN* under a heavy background traffic as the same with the previous scenarios. As shown in Figure 4.10(a) and 4.10(b), the results are aligned with the previous experiments with *Big Buck Bunny*. With this new content, the contribution of our proposed approach is much more visible because video quality difference between two consecutive quality levels and bitrates in *Of Forest And Men* is higher compared to *Big Buck Bunny* as we already presented in Figure 4.6. To sum up, *CSASDN* is much better in terms of utilization of the network resources by enabling higher video bitrates.

Table 4.1. Comparative evaluation of different approaches in terms of all QoE metrics over *Big Buck Bunny*.

		$VQ_{Session}$			**AVG. # SWITCHES**	**AVG. # STALLS**	**AVG. TTFF (ms)**
		MIN	**MED**	**MAX**			
DASH.js (BOLA)	**Client1-2 (2s)**	0.9627	0.9656	0.9697	57	0	1136
	Client3-4 (6s)	0.9368	0.9491	0.969	53	1.4	2677
	Client5-6 (10s)	0.8674	0.9037	0.9292	34	3.3	3658
	Client7-8 (15s)	0.8951	0.8985	0.9209	29	4.6	7205
Explicit SDN	**Client1-2 (2s)**	0.9493	0.9499	0.9499	4	0	3923
	Client3-4 (6s)	0.9488	0.9490	0.9499	10	0.6	5922
	Client5-6 (10s)	0.9493	0.9496	0.95	7	0.7	13662
	Client7-8 (15s)	0.9494	0.9497	0.95	5	1.4	16540
CSASDN **DASH**	**Client1-2 (2s)**	0.959	0.9592	0.96	8	0	6568
	Client3-4 (6s)	0.9609	0.9617	0.9625	15	0.6	6206
	Client5-6 (10s)	0.9615	0.962	0.9626	14	1.2	6586
	Client7-8 (15s)	0.9621	0.9627	0.9633	11	2.4	6957

Table 4.2. All QoE metrics over *Of Forest And Men* (The state-of-the-art SDN vs. our proposed approach).

		$VQ_{Session}$			**AVG. # SWITCHES**	**AVG. # STALLS**	**AVG. TTFF (ms)**
		MIN	**MED**	**MAX**			
Explicit SDN	**Client1-2 (2s)**	0.8876	0.8876	0.8892	6	0.6	4364
	Client3-4 (6s)	0.8836	0.8871	0.89	7	1	6356
	Client5-6 (10s)	0.8796	0.884	0.89	6	1.2	9772
	Client7-8 (15s)	0.8853	0.89	0.89	6	1.6	10589
CSASDN **DASH**	**Client1-2 (2s)**	0.9086	0.9099	0.9105	8	0.2	2101
	Client3-4 (6s)	0.9139	0.9159	0.9182	8	0.2	3350
	Client5-6 (10s)	0.9144	0.9149	0.9182	9	0.4	5254
	Client7-8 (15s)	0.9133	0.9157	0.916	7	1.6	6218

Table 4.3. Performance comparison of all the approaches in terms of the avg. eMOS over *Big Buck Bunny*.

	BOLA	**ExplictSDN**	**CSA-SDN**
Client1-2 (2s)	4.112	4.072	4.257
Client3-4 (6s)	2.959	4.114	4.273
Client5-6 (10s)	1.203	4.156	4.208
Client7-8 (15s)	1.159	4.175	4.169
Avg. of All Clients	**2.359**	**4.129**	**4.226**

Table 4.4. The avg. eMOS over *Of Forest And Men* (The state-of-the-art SDN vs. our proposed approach).

	ExplictSDN	**CSA-SDN**
Client1-2 (2s)	4.421	4.721
Client3-4 (6s)	4.339	4.833
Client5-6 (10s)	4.349	4.739
Client7-8 (15s)	4.492	4.517
Avg. of All Clients	**4.401**	**4.702**

Finally, we evaluate the performance by relying on not only SSIM-based video quality values but also the other QoE metrics. Each experiment we previously presented is repeated five times for both *Big Buck Bunny* and *Of Forest And Men*. Start-up delays (i.e., TTFF - time-to-first-frame), number of stalls and number of switches are also collected from each session. All average QoE metrics are summarized in Table 4.1 and Table 4.2. $CSASDN$ obviously provides a better $VQ_{Session}$ among sessions with fair start-up delays while not causing more buffer stalls and a considerable increase in quality switches compared to the state-of-the-art SDN-based and the purely client-based approaches. To compare the performance over a combined QoE metric, we also compute eMOS values in all experiments. The average eMOS values in playback sessions with various chunk duration are presented in Table 4.3 and Table 4.4. The results of eMOS in Table 4.3 show that our proposed $CSASDN$ approach outperforms both $BOLA$ and $ExplicitSDN$ over *Big Buck Bunny* in terms of the average eMOS. As depicted in Table 4.4, the performance of our approach compared to $ExplicitSDN$ is clearly better also in the content *Of Forest And Men*.

4.4.5. Impact of different number of background flows

We observe the impact of number of background flows on the QoE metrics in both our proposed approach and *ExplicitSDN*. In the experiments, we rely on the content *Of Forest And Men* with various chunk-size versions. In this section, we exclude *BOLA* because the purely client-based ABR mechanisms obviously underperform SDN-assisted approaches based on our previous experiments and the literature detailed in Section 3.1.2.

For each ABR approach, we run three experiments with different number of background flows: 8, 12 and 16 with the same maximum background traffic capped to 10Mbps in the same setup explained in Section 6.3.1. Each experiment is repeated five times. The total number of DASH players is 8 same as in the previous experiments. In other words, there totally exists 16, 20, 24 clients in the experiments, respectively. Although the number of the background flows changes, we keep the cap of the total background traffic the same.

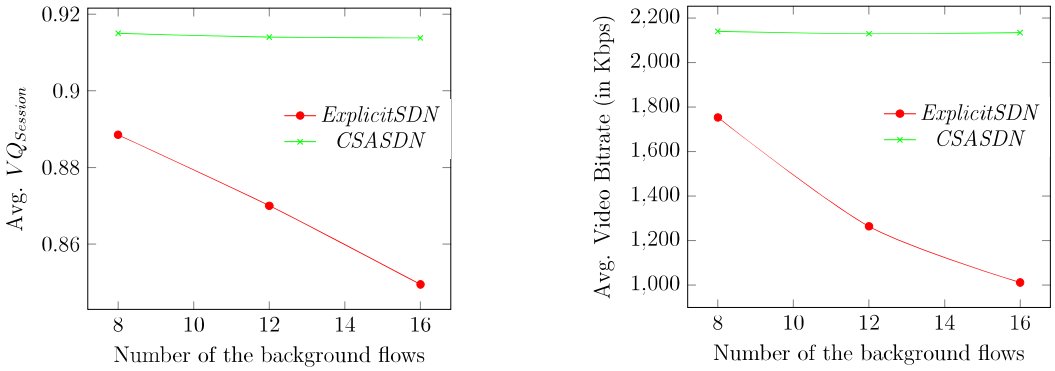


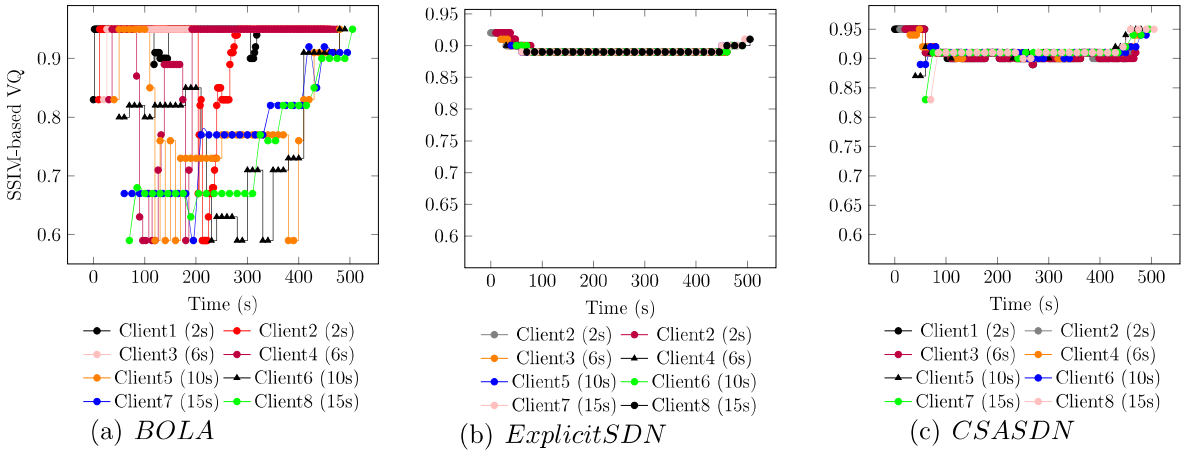
Figure 4.11. The impact of the number of the background flows on VQ.

Figure 4.11 shows the effect of the number of the background flows on the average video bitrate and the corresponding average $VQ_{Session}$. It obviously reflects that *CSASDN* is not affected by the increase in the number of the flows whereas *ExplicitSDN* is proportionally choosing a lower quality even though the total background traffic is not changed. It also proves that *CSASDN* is successful in detecting background flows, following the total traffic generated by them and computing the

available capacity for the DASH players. Moreover, *ExplicitSDN* explicitly causes a network under-utilization problem while the number of flows increases because it does not track the diversity of the background flows in terms of the amount of the generated traffic. Furthermore, we collect the other QoE metrics from the experiments. *CSASDN* has the similar QoE metrics with the previous experiments shown in Table 4.2 among all different number of the flows. In other words, our proposed approach provides a better quality while not inducing any video freezes as the number of flows increases.

4.4.6. The applicability to arbitrary arrival/departure times

DASH clients in the previous experiments started and ended streaming at the same time just to be able to easily repeat the same setup configuration for the sake of simplicity. In the real-world scenarios, clients mostly join and leave the system at different times. In this subsection, we conduct experiments to show that our proposed approach is applicable to the real-world cases by handling arbitrary arrival and departure times of the clients.



(a) *BOLA* (b) *ExplicitSDN* (c) *CSASDN*

Figure 4.12. Video quality during 7.5-min *Of Forest And Men* session among 8 clients with different arrival times in the existence of heavy background traffic (*rand*(6..10) Mbps).

We use *Of Forest And Men* content with the setup configuration explained in Section 5.1. In each experiment, there exists 16 clients as 8 DASH clients and 8 background flows competing for the 30Mbps total shared capacity on the same bottleneck link. The total background traffic amount is capped to 10Mbps with a lower bound of 6Mbps. We commonly apply a different ABR mechanism per each experiment to all 8 DASH clients with various chunk duration. Streaming on each client is started 10s later than the previous one. We follow the SSIM-based video quality values over time for each session streaming under a heavy background traffic randomly generated by 8 background flows. At the end of each experiment, we also compute eMOS of the playback session based on the mean and standard deviation of video quality, and video freezes.

Figure 4.12 reflects the video quality values of all 8 sessions over time in both approaches. The gradual video quality drop in the first 70 seconds is common in all the approaches because clients arrive one by one with 10-second intervals and the available bandwidth per each client decreases over time. As of 70th second, all clients have started playback sessions and then, SDN-based approaches achieve a more stable video whereas *BOLA* has highly fluctuating and lower video quality. The results also show that *CSASDN* is slightly better than *ExplicitSDN* in terms of the SSIM-based

video quality because *ExplicitSDN* does not follow the active background traffic and it evenly distributes the available bandwidth among all background and DASH flows. Moreover, the gradual quality increase in the last 70 seconds due to the ended playbacks proves that our proposed approach successfully detects leaving clients and updates target bitrate levels of the continuing clients.

The computed eMOS values of each playback session in all the approaches are used to generate the box plots in Figure 4.13. Because eMOS in *BOLA* varies 0.2 to 4.2 with the average of 2.51, the figure does not include the eMOS values of *BOLA* for the better visualization. The average eMOS in our proposed approach is 4.38 while it is 4.15 in *ExplicitSDN*. To sum up, it shows *CSASDN* outperforms the competitors in terms of QoE in the case of arbitrarily arriving and leaving clients.

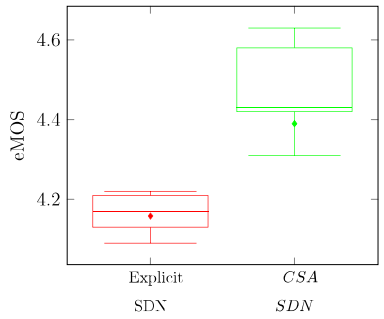


Figure 4.13. Performance Comparison in terms of the eMOS between *ExplicitSDN* and *CSASDN* for the arbitrary arrival/leaving experiments.

4.5. Discussion

We presented an SDN-assisted, chunk duration-aware DASH system for VoD use cases, by assuming that multimedia content is encoded and packaged offline as ready to be served on demand. Hence, we did not take into account the impact of real-time encoding and packaging process on users' QoE and the available throughput measurement. In the next chapter, we address this gap when the content is being prepared while streaming, and explain how to address those challenges in live event streaming scenarios through our proposed SDN-assisted solution in this chapter.

5. DASH FOR LIVE EVENTS USING SDN ASSISTANCE

In this chapter, we introduce a low-latency live streaming system over DASH using SDN assistance after highlighting the challenges of live streaming with CMAF and HTTP 1.1 Chunked Transfer.

5.1. Introduction

The demand for video streaming services is driven by the growing investment of over-the-top (OTT) service platforms not only on Video-on-Demand (VoD) use cases but also live streaming. Amazon Prime's procuring broadcasting rights of worldwide premium sports events such as English Premier League, ATP Tour Tennis, and American National Football League [65] perfectly exemplifies a huge interest of OTT platforms on live streaming. One of the success factors for this penetration is the wide adoption of DASH. However, DASH has been originally developed for VoD use cases to provide a high QoE with uninterrupted video streaming based on adaptive video bitrates over highly varying network conditions and heterogeneous devices. Hence, OTT services face challenges to deliver low-latency live streaming over DASH due to large playback buffer and segment duration, whereas it is expected to beat satellite and terrestrial cable latencies in the broadcast world characterized by a 5 to 10 second latency [66]. In the conventional DASH systems, players buffer a few segments to start playback (e.g., three 10-second segments in Apple HLS which causes a 40-second playout delay between the user and the live signal). It brings about a spoiler effect. How would you feel if you had experience of hearing from a neighbour or receiving a message from a friend cheering about the goal that you cannot see for the next 30 seconds?

Is simply combining the conventional standard of HTTP 1.1 chunked encoding transfer with the new packaging approach of CMAF enough to reduce the live latency without causing any QoE degradation? The answer is no. The core logic of the stan-

dard adaptive bitrate algorithms, throughput estimation logic, estimates the available bandwidth as almost equals to the encoded bitrate in live streaming over chunked encoded transfer because they simply divide the segment size by the download time as pointed out in [51] when the chunks of the segment are being prepared during the segment download. Therefore, it clearly prevents players from switching up the quality due to under-estimation of the available bandwidth because of the idle times between chunks. There is theoretically no exact way in the standard to determine the total idle times to be able to calculate the accurate download rate by subtracting idle times from the total download time of the segment because the HTTP protocol does not provide the beginning time of each chunk download while the response for the relevant segment is progressing.

In this chapter, we address the ABR problems due to the under-estimation of the available bandwidth in HTTP 1.1 chunked encoding CMAF by leveraging a central controller that has a global view of the network and DASH players in the shared network at the edge. We benefit from the Software-Defined Networking (SDN) concept which enables a standardized traceability and programmability of the network elements. We propose an SDN-supported adaptive bitrate mechanism with a central module which communicates with both the SDN controller and DASH players via a separate communication channel to provide the highest possible video quality without causing any video stalls. It resolves the network underutilization issue in the HTTP chunked encoding CMAF by centrally monitoring all the background flows to calculate their capacity consumption on the bottleneck link. Then, it accurately measures the remaining bandwidth for the DASH players and returns the optimal bitrates to the players based on the target live latency, available video bitrate list and segment duration shared by the players at the beginning of the live streams. Hence, it can be considered as a lightweight SDN integration as it does not require any active network programmability to avoid any scalability issues. In case of any failure in connecting to the SDN-aided central controller, DASH clients rely on their purely client-based conventional ABR mechanisms as in line with the distributed nature of DASH systems. While providing the low-latency live streaming, we achieve a better QoE without sac-

rificing perceptual video quality and coexistence with the legacy players in a reliable manner.

Our work differs from all the studies in the literature survey in Section 3.2 in many essential aspects, such as utilizing CMAF, considering background traffic, working on the HTTP 1.1-based pull models of the legacy players over the existing CDN infrastructure, and not sacrificing QoE to reduce latency. Our work in this chapter is also not the first to propose leveraging SDN for video streaming systems to improve QoE. Previously we introduced an SDN-assisted DASH mechanism for VoD use cases over the legacy systems with the long segment duration over the standard HTTP 1.1 GET and POST requests [4]. In the same manner, Bentaleb et al. [31] implemented an external SDN-based network management module to resolve fairness and quality stability problems in the existence of multiple DASH clients in the same shared network. With a similar approach, using the active network programming with the bandwidth slicing per player, Kleinrouweler et al. [29] proposed a target video bitrate signaling from a central module to reduce the number of switches in video quality. Furthermore, Go et al. [37] implemented an SDN-based framework for IP Video Surveillance networks to centrally execute video bitrate adjustments by utilizing the network statistics messages available in OpenFlow, a standard protocol for the communication between the data plane and the SDN controller. Moreover, Jiang et al. [38] introduced an SDN-based dynamic network resources allocation to guide the video bitrate selection of clients through rate limiting in OpenFlow. However, none of them considers the low-latency requirement for live streaming and CMAF packages over DASH.

5.2. SDN-supported Low-Latency (LL) streaming over DASH

5.2.1. System Overview

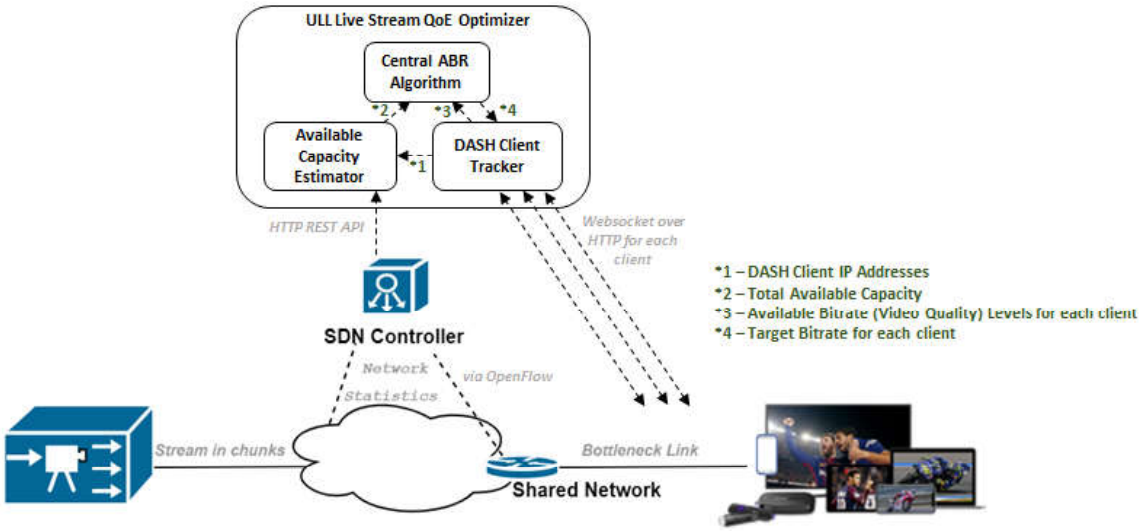


Figure 5.1. System Overview of SDN-supported Low-latency (LL) Streaming over DASH.

The system of SDN-supported LL live streaming is depicted in Fig. 5.1. It mainly consists of DASH clients, OTT live stream provider and a central QoE optimizer on top of the SDN Controller. All communication between DASH players and OTT content providers are based on HTTP REST APIs in the standard. Being different from the legacy systems, it exposes a bidirectional communication channel between the clients and the central QoE optimizer. To keep one of the main advantageous of the DASH that enables clients to reach the DASH server sitting behind the firewall and NAT, it is implemented using websocket that is a widely implemented web technology that uses Port 80 to make the connection appear to the HTTP server.

As the same with the conventional DASH systems, the OTT live stream provider serves the content at different quality levels and a manifest file called multimedia presentation description (MPD) for the content which includes available quality levels, segment duration and URLs for each quality level and each segment to enable clients

to switch video quality at segment boundaries. All the details about the MPD data model and the initialisation flow are detailed in [67]. Segments are carried in CMAF chunks using HTTP 1.1 chunked encoding transfer. Video quality level for segments is signaled by the central algorithm on top of DASH Client Tracker and Available Capacity Estimator. The live latency target is also set by the central controller. Each player is subscribed to DASH Client Tracker and informs available quality levels. The capacity estimator continuously monitors all the flows leveraging SDN Northbound APIs and getting the network statistics. Then, it estimates the available network capacity for DASH players in each shared network. The available capacity and all the DASH client information such as quality levels and segment duration of the live stream are inputted to the central ABR heuristic. We periodically run the heuristic for each shared network to calculate the possible highest quality level of each DASH client. Finally, the target quality level is returned to each client via the DASH Client Tracker.

The key design principle of the system is simplicity to avoid any blocking issue in possible real-life deployment scenarios. First, it does not require any active network programming for explicit per-client bandwidth allocation on bottleneck links. It is helpful for not only a simple design but also the overall network utilization, background and legacy DASH client flows. Second, it just collects the video bitrate list, and chunk duration from players once at the beginning of each session and does not require client feedback messages anymore during the playback. From the central controller to a particular DASH player, the target video bitrate is sent only when the output of the QoE optimization heuristic for that player is changed due to any considerable network fluctuations or arrival/exit of DASH players. Hence, it does not induce any noticeable overhead because of the out-band bidirectional communication channel between the players and the central QoE optimizer. Third, it does not expose any deep packet inspection approaches to sniff manifest files to extract video bitrate alternatives and the relevant video metadata. Therefore, it is totally compatible with encrypted manifest files over HTTPS. Fourth, DASH players keep working with their purely client-based conventional ABR mechanisms as a fallback mechanism in case of any failure in the

communication with the central QoE optimizer. It is a kind of recovery mode as in line with the distributed nature of DASH systems. Finally, there is no modification on the legacy DASH servers, content and manifest files.

5.2.2. Central QoE Optimization Heuristic

The central QoE optimizer aims at finding the available capacity for all the DASH clients on the bottleneck link to calculate the highest available video quality level per client while achieving the low live latency and avoiding video stalls.

The live delay refers to the difference between the time when the event happens, and the user sees it. Our system allows the target value of the live delay to be configured. It can be considered as a kind of service-level agreement (SLA) value. By low latency, we mean a typical value under 10s. This target limits the maximum duration of the buffered data on the player. Hence, such a small buffer complicates ABR tasks due to the risk of video stall events in case of even a single suboptimal decision under fluctuating background traffic. If players conservatively chose the lowest video quality to avoid the risk of draining the buffer in this challenging context, an undesirable QoE for end-users with network under-utilization issues would be inevitable.

The primary enabler in providing a high QoE during the live event without condemning users to the lowest quality or frequent and prolonged video stall events is to calculate the background traffic accurately to determine the available capacity for the clients. Fig. 5.2 illustrates the sequence to find the available capacity periodically. As each client connects to the central module via a separate communication channel, IP addresses of connected DASH players are known and kept in a hash set of the DASH clients (DC). When each player stops streaming, it closes the connection, and it is removed from the DC store. Additionally, statistics of each flow are retrieved with the help of SDN. By accepting these values, the total background traffic is calculated by summing the used bandwidth of all flows, not in the hash set of DASH player IP addresses at any time. The historical values of used bandwidth by background flows

are kept in a sliding window. It stores the last K values. The algorithm relies on the average of these values in the window to smooth the estimated background traffic to avoid a higher number of quality switches.

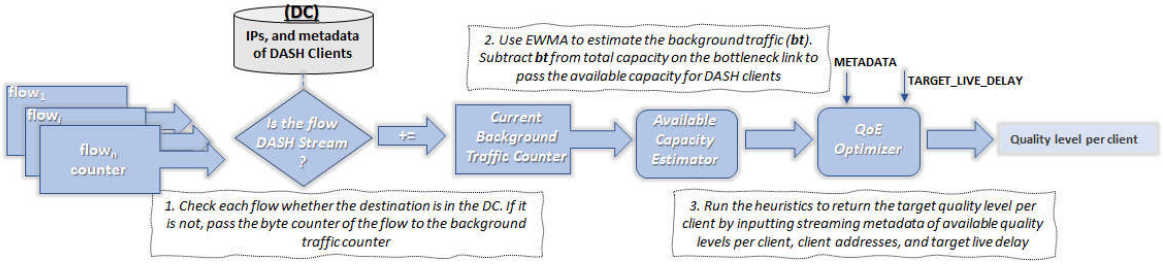


Figure 5.2. Flow in Central Adaptive Bitrate Heuristic to optimize QoE in live streaming.

The smoothed background traffic is based on Exponential Weighted Moving Average (EWMA) as

$$SBT(i) = (1 - \epsilon) \times SBT(i-1) + \epsilon \times BT(i), \tag{5.1}$$

where $BT(i)$ and $SBT(i)$ are the measured background throughput, and its smoothed version at the iteration i respectively, and ϵ is the smoothing factor. For ϵ, we select the most used value, $\frac{2}{K+1}$ as the simple moving average. This value is helpful to minimize fluctuations in video quality, although it causes reacting with more latency to network fluctuations, especially for the larger K values. To be able to react much faster to highly varying background traffic, alternatively, an adaptive smoothing factor can be used depending on the current trend of the measured throughput value. A sample application is presented to select a smoothing factor dynamically in the study introduced by Sobhani et al. [59]. A comprehensive evaluation study among various alternatives of the throughput prediction strategies can also be found in [68].

As the final step before the QoE-aware heuristic, the available capacity for DASH players is calculated by subtracting the smoothed background traffic from the total capacity of the bottleneck link. Then, the central mechanism with a QoE-aware heuristic accepts it, the target live delay, and all the DASH clients metadata of segment duration

and available quality levels (i.e., video bitrates) explained in Algorithm in Figure 5.3 as the input. The algorithm starts by selecting the lowest quality for the current fragment. It then increments its quality level by 1 until the total download time exceeds the sum of the segment duration and the previously buffered data duration. It provides to choose the desired video quality level so that the total download time per segment is not greater than the segment duration plus the buffer duration. Since players can also buffer data less than the target live latency at the beginning, it relaxes the download time for half the target live delay to be able to achieve the highest video bitrate without causing any video stalls. The algorithm repeats this procedure over all the DASH clients in each run. It is periodically triggered. The previous video quality levels of all the clients in each run are also cached. Therefore, if the next run of the algorithm returns the same quality level per a particular client, it is not shared with the client again as the client continues with the latest update from the central module. Hence, the communication channel between players and the central module is efficiently used to minimize the communication overhead in the system.

Input: Available capacity for DASH streams ($availableCapacity$), target live delay ($targetLatencyFromSLA$), list of n DASH clients with available video quality levels [0,...,$maxQualityLevel$], and segment duration

Output: Next quality level for each DASH player

$i = 0$;

$perDeviceCapacity = availableCapacity$ / n;

repeat

$currLevel_i = 0$;

$estimatedDownloadTime_i = 0$;

while $estimatedDownloadTime_i < segmentDuration_i$ **do**

$currLevel_i = currLevel_i + 1$;

if $currLevel_i > maxQualityLevel$ **then**
break

end

$estimatedSegmentSize_i =$
$segmentDuration_i \times videoBitrate[currLevel_i]$;

$estimatedDownloadTime_i =$
$estimatedSegmentSize_i / perDeviceCapacity$;

end

$currLevel_i = currLevel_i$ - 1 ;

if $(client_i[currLevel]$!= $currLevel_i)$ **then**

$client_i[currLevel] = currLevel_i$;

Send $currLevel_i$ to $client_i$ as the target video bitrate level;

end

$i = i + 1$;

until $i >= n$;

Figure 5.3. Central QoE-aware heuristic.

5.3. Experimental Evaluation

In this section, we conduct extensive experiments to confront our proposed SDN-supported LL Streaming mechanism with the purely client-based approaches in terms of QoE in the context of CMAF packages and HTTP 1.1 Chunked Encoding transfer.

5.3.1. Testbed and Implementation Details

We leverage DASH.js [27] for the players, Floodlight [60] for the SDN controller, Mininet [56] for the simulation of the network plane, and $iPerf$ [62] for generating background traffic. The testbed includes 4 Mininet-based DASH players competing for the 20 Mbps total shared capacity under the same bottleneck link as shown in Fig. 5.4. Under the same bottleneck link, it also consists of 5 hosts to generate background traffic. Latency on the local links in the Mininet is set to 10ms with 20 Mbps down-link capacity. We run each DASH player in a new instance of the Firefox browser on an Ubuntu machine with a quad-core CPU running at 2.60 GHz and 16 GB RAM. We also run our $iPerf$-based background traffic generator script in Mininet hosts for highly varying time intervals with a given maximum traffic amount. Both traffic amount and time are randomized to mimic fluctuating background traffic.

We use the test live-streaming content served by the live simulator running on the DASH-IF server [27]. It generates content with CMAF chunks and pushes them over HTTP 1.1 Chunked Encoding. Each segment is eight seconds long and encoded into four quality levels for video at {0.3, 0.6, 1.2, 2.4} Mbps and one bitrate level for audio at 37 kbps. Chunk duration can be set to any floating number. Our local Mininet environment with lots of hosts is connected to the test server via the last-mile ISP with the minimum guaranteed capacity of 50 Mbps.

In this setup, we compare the SDN-supported LL Live-streaming mechanism with $ACTE$, and the low-latency mode enabled DASH.js [69], which has various ABR heuristics. Our evaluation is based on the QoE metrics explained in Section 6.3.2. In all

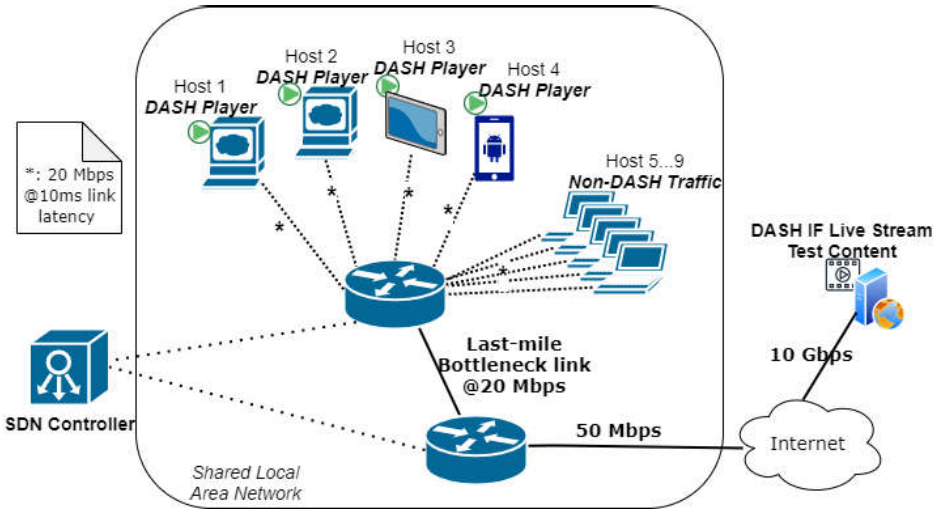


Figure 5.4. Experiment Setup.

the experiments, players under test have a catch-up feature with the adapting playback rate to pull the player back to the optimum live edge using the default support in DASH.js. In other words, the player can keep itself pretty tightly to a target latency by controlling the playback rate relying on the assumption that variations in the playback rate of 10% or less are not perceptually noticeable enough to end-users.

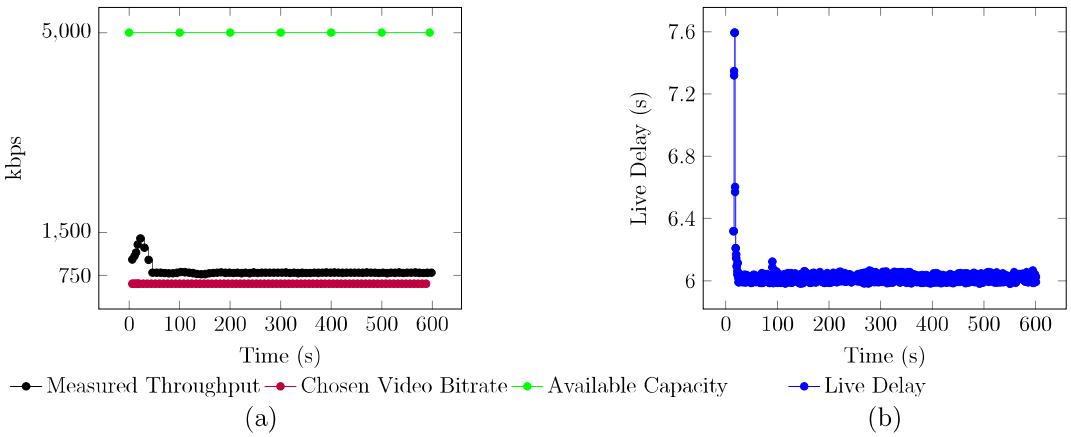


Figure 5.5. QoE Metrics in the baseline low-latency enabled DASH player for CMAF.

5.3.2. The Metrics

We first collect the average live latency for each session. It represents the total time from capturing to rendering. The relevant field in the dash.js reference player per each second is periodically retrieved. Then, the average live latency is simply calculated

in Equation 5.2 as follows,

$$D_{avg} = \frac{\sum_{t=1}^{T} d_t}{T} \quad \text{where } d_t \text{ indicates the live delay at time } t \text{ during the total } T\text{-second live stream.} \tag{5.2}$$

As the other factors with the negative impact on QoE, we follow the duration and number of freezes and the number of quality level switches during the session. The number of video freezes is simply the number of occurrences when the playback buffer is empty while the number of the switches is the total number of quality drops or jumps during the session. The duration of the video freeze is the total time of a particular stall from the freeze to re-starting. In our performance evaluations, we also calculate the estimated Mean Opinion Score (eMOS) [19] to better reflect the overall QoE perceived by end users. It is calculated by combining the different QoE metrics into a single metric in the linear model provided in Equation (2.1).

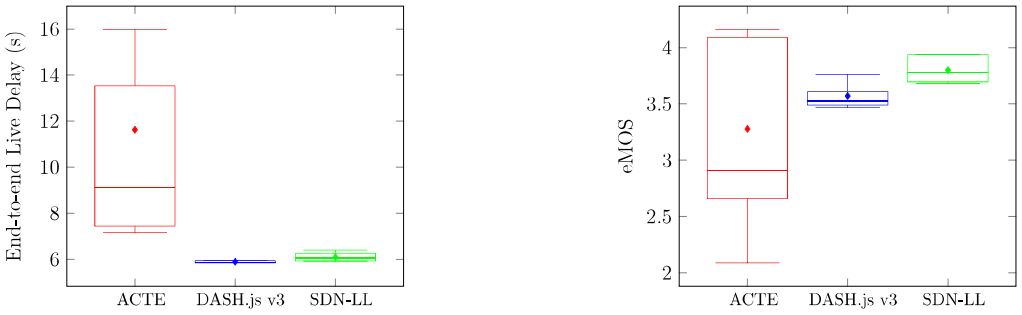
5.3.3. Results

We conduct three categories of the experiments in the context of low-latency streaming with CMAF and HTTP Chunked Transfer to see:

(i) Network under-utilization issue in purely client-based legacy ABRs,
(ii) Comparison between our proposed SDN-supported LL mechanism and the legacy ABRs over 3 s target live delay,
(iii) Comparison between our proposed SDN-supported LL mechanism and the legacy ABRs over 6 s target live delay.

First, we run experiments with one client in 5 Mbps available capacity by setting the live latency target to 6 s, and using 0.5 s chunks. The player on the client has DASH.js 2.3 with the default configuration which estimates the throughput over the full segment size divided by the download time, that is the difference between the times when the segment was fully received and the request was sent. Figure 5.5 reflects the

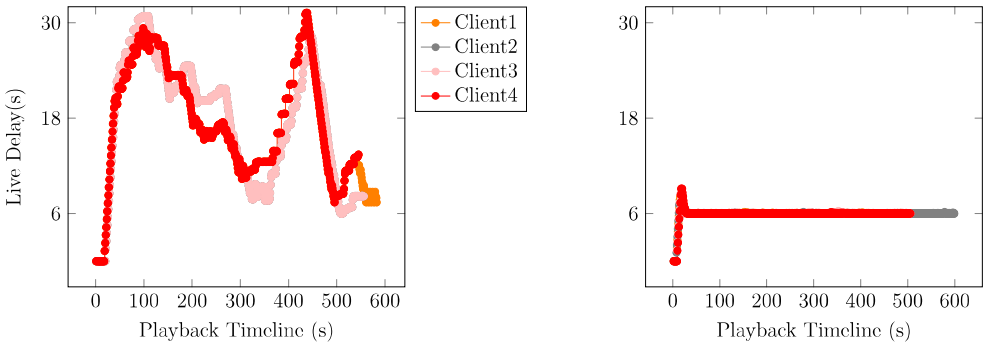
measured throughput, chosen video bitrate by the player, and available capacity in addition to the live delay during a 10-minute live stream. Although it achieves the low latency target, it cannot switch up the video quality by sticking to the quality level of 604 kbps video bitrate despite having higher-quality options of 1.2 Mbps and 2.4 Mbps, and a dedicated 5 Mbps link without any other background flow. It underestimates the available capacity about 800 kbps as almost equal to the encoded bitrate because the download time from the client point of view includes lots of idle times as the chunks of the segment are being prepared during the segment download. This case clearly shows the network under-utilization problem in the default ABR behavior on the legacy players for HTTP 1.1 Chunked Encoded CMAF by condemning users to a lower quality in spite of the existence of available resources for a better video quality.



(a) Avg. Live Delay values

(b) eMOS-based QoE

Figure 5.6. Live delay values and QoE scores for different ABR algorithms over 10 repetitions for 4-client setup (**Target live delay of 6 s**).



(a) ACTE (b) SDN-LL DASH

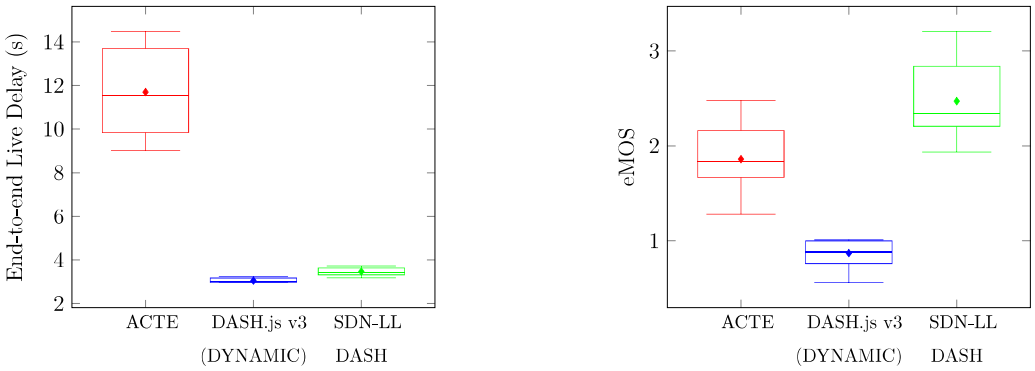
Figure 5.7. Live latency variations during a 10-minute live session with the target delay of 6 s from a sample run.

Second, we compare our approach with the competitors over 0.5 s chunk and 6 s target delay in the existence of background traffic. As the competitors, we choose $ACTE$ and the default dynamic strategy ($DYNAMIC$) [70] of DASH.js v3 as the combination of the mechanisms of insufficient buffer rule, BOLA [20] and its throughput-based ABR for low-latency mode which ignores the download times of the chunks higher than the average download time of the previous chunks in the same segment. The buffer-based decision mechanisms in ($DYNAMIC$) are also similar to the bitrate adaptation method for seamless on-demand video streaming introduced by Le et al. in [71]. In each 10-minute live-streaming experiment, there exist four homogeneous players with the same behavior under the shared bottleneck link of 20 Mbps by generating random traffic between 5 Mbps and 14 Mbps for random intervals. Each interval duration with a random traffic amount capped to 14 Mbps is within the range of 15 s to 30 s, i.e., background traffic flows change in periods from 15 s to 30 s. We calculate eMOS as a proxy of the combined QoE and the average end-to-end live delay at the end of each session.

Given that we have four clients in one run, and there exist 10 repetitions, there are 40 different 10-minute live streaming sessions for each ABR mechanism. The box plots in Figure 5.6 depict the statistical distribution of the eMOS and the average live delay values from 40 sessions per ABR strategy. Figure 5.6(a) reflects that our proposed

Table 5.1. All QoE metrics in different ABR mechanisms (Target live delay of 6 s).

		Avg.Live Delay (s)	**Avg. Duration per Stall (s)**	**Avg. Num. Stalls**	**Avg. Num. Switches**	**Avg. Video Quality Level (1-4)**	**Avg. eMOS (0-5.84)**
DASH.js v3 (DYNAMIC)	Client 1	**5.91**	0	0	8.4	2.91	**3.47**
	Client 2	**5.95**	0	0	7.9	2.97	**3.58**
	Client 3	**5.89**	0	0	8.8	3.01	**3.61**
	Client 4	**5.96**	0	0	9.1	2.96	**3.56**
ACTE	Client 1	**9.44**	0.52	13.3	2	3.91	**3.32**
	Client 2	**10.2**	0.55	15.2	2	3.95	**3.38**
	Client 3	**8.28**	0.64	11.6	2	3.87	**3.31**
	Client 4	**8.77**	0.59	12.3	2	3.85	**3.40**
SDN-LL DASH	Client 1	**6.01**	0.25	0.8	11.5	3.13	**3.84**
	Client 2	**6.04**	0.43	0.7	11.1	3.15	**3.82**
	Client 3	**6.03**	0.27	0.8	12.3	3.36	**3.85**
	Client 4	**6.08**	0.41	1.1	10.8	3.24	**3.78**



(a) Avg. Live Delay values

(b) eMOS-based QoE

Figure 5.8. Live delay values and QoE scores for different ABR algorithms over 10 repetitions for 4-client setup (**Target live delay of 3 s**).

mechanism and *DYNAMIC* on DASH.js can approach to the target live delay of 6 s, whereas *ACTE* cannot get closer to the target delay due to frequent and long video freezes because it ignores low throughput measurements in case of highly fluctuating background traffic. Therefore, it cannot prevent draining buffer. In the same manner, Figure 5.7 includes all live latency snapshots during an example live stream session from the four clients when *ACTE* and SDN-LL DASH are applied. It is clear that SDN-LL DASH consistently provides the target latency, although *ACTE* suffers from a high variance of the end to end latency values. In some periods, as observed in Figure 5.7 (e.g., from 100 s to 300 s), *ACTE* can gradually decrease the live delay to catch up the target delay after video stalls by leveraging the adapting playback rate

feature. Figure 5.6(b) shows that our proposed SDN-LL DASH approach is better than the competitors in terms of the eMOS as well. It also points out that *ACTE* has fairness issues, as seen in the broad range of the average values, whereas the other two approaches do not manifest such an issue.

Table 5.2. All QoE metrics in different ABR mechanisms (Target live delay of 3 s).

		Avg.Live Delay (s)	**Avg. Duration per Stall (s)**	**Avg. Num. Stalls**	**Avg. Num. Switches**	**Avg. Video Quality Level (1-4)**	**Avg. eMOS (0-5.84)**
DASH.js v3 (DYNAMIC)	Client 1	**2.96**	1.02	0.5	7.9	1.26	**0.96**
	Client 2	**3.22**	0.81	1	9.3	1.15	**0.82**
	Client 3	**3.24**	0.97	0.4	8.1	1.22	**0.97**
	Client 4	**2.99**	0.80	0.6	9.1	1.16	**0.88**
ACTE	Client 1	**12.42**	0.53	26.3	2	3.92	**1.91**
	Client 2	**11.31**	0.66	20.2	2	3.87	**1.78**
	Client 3	**11.75**	0.47	22.8	2	3.86	**1.71**
	Client 4	**11.38**	0.54	20.3	2	3.87	**1.86**
SDN-LL DASH	Client 1	**3.45**	0.27	6.1	10.5	3.34	**2.49**
	Client 2	**3.58**	0.11	5.7	10.7	3.27	**2.63**
	Client 3	**3.39**	0.15	5.8	11.0	3.32	**2.52**
	Client 4	**3.40**	0.22	5.1	10.2	3.21	**2.47**

Table 5.1 represents all QoE metrics for each client when a different mechanism is applied. The average values are calculated over 10 repetitions per client. SDN-LL DASH provides a better overall QoE based on the average eMOS score even though *DYNAMIC* can avoid any video stall event using its insufficient bandwidth rule and choosing lower video bitrates. SDN-LL DASH enables a better video quality on average at the expense of one short video stall in a few cases. In summary, the results conclude that our proposed mechanism outperforms in terms of the combined QoE metric eMOS while, at the same time, achieving the SLA live target delay.

Finally, we conduct the previous experiment configuration in a more aggressive target delay by setting it to 3 s. Table 5.2 shows all QoE components among all three approaches. As a combined QoE metric calculated over all the different QoE factors such as the number and the duration of video stalls, the number of quality switches, the average video quality, the average eMOS scores over all the experiments conclude that our proposed mechanism outperforms the competitors. While *ACTE* has more dramatic video freezes and consequently it does not manage to reach the live latency

target of 3 s, DASH.js v3.0 conservatively keeps the video bitrate low and achieves the target delay. As shown in Figure 5.8(a), our SDN-supported LL approach provides 3.45 s live latency on average without sacrificing the video bitrate and the other QoE metrics over all the clients during all the sessions. Figure 5.8(b) indicates that the eMOS value at the first quartile of the 40 sessions in our approach is better than even the third quartile eMOS scores of the competitors.

5.4. Discussion

We presented how we utilized our SDN-assisted solution previously detailed in Chapter 4 for live event streaming use case over HTTP chunked transfer of CMAF packages. Although results demonstrated it provides a better QoE compared to the state-of-the-art solutions, it would require a dedicated SDN northbound application, which does not exist in the legacy systems. Deploying such an extra central component is not perfectly in line with the nature of the distributed design of DASH systems. Though it does not rely on active network programming of SDN and just uses the network statistics collection through a lightweight integration, it may still make the real-life applicability questionable. To address this concern around the real-life deployment challenges with SDN integration, in the next chapter we propose a purely client-based approach and eliminate the need for SDN assistance, while achieving a high QoE and low latency for live event streaming scenarios.

6. LOW-LATENCY HAS CLIENT USING RULE-BASED ABR HEURISTICS

In this chapter, we present a HAS client with bandwidth measurement heuristics for live streaming without any central component assistance.

6.1. Introduction

The popularity of video streaming services is increasing with the growing investment of over-the-top (OTT) service platforms, considering both Video-on-Demand (VoD) and live streaming use cases. One of the success factors for this growth is the wide adoption of HTTP adaptive streaming (HAS). HAS eases the traversal through NATs and firewalls and allows the use of existing caching infrastructure (e.g., content delivery networks) to scale in a cost-effective manner. In HAS, the multiple versions of the same content at different quality levels are kept at the OTT back-end side. It allows video players to choose the appropriate video quality based on the client-side decision to adapt to the varying network conditions. Each encoded quality version is divided into small segments, including short-duration content (e.g., 1-10 seconds of video). Segment duration and boundaries are the same among all versions of the same content. Therefore, segments are aligned between one version and other versions of quality levels in the video timeline so that the player can smoothly switch to different quality levels at the segment boundary when network conditions change [43].

HAS clients have been originally designed for VoD use cases without any latency requirements. Hence, OTT services face challenges to deliver low-latency live streaming with HAS due to large playback buffer and segment duration, whereas it is expected to beat satellite and terrestrial cable latencies in the broadcast world characterized by a 5 to 10-second latency [66]. The encoder in the legacy system waits for a full segment to be encoded to be able to push to the network making at least one segment duration delay in the content generation. Additionally, the legacy players buffer a few segments

to start playback (e.g., three 10-second segments in Apple HLS). When adding the delay of the buffered segments in players to the encoder-side delay, the time difference between the moment when the live event is rendered at the client device, and when it happens can go to above four times the segment duration. It causes a spoiler effect as an unpleasant user experience (e.g., hearing from a neighbor cheering about a goal in a football match that you cannot watch for the next 40 seconds).

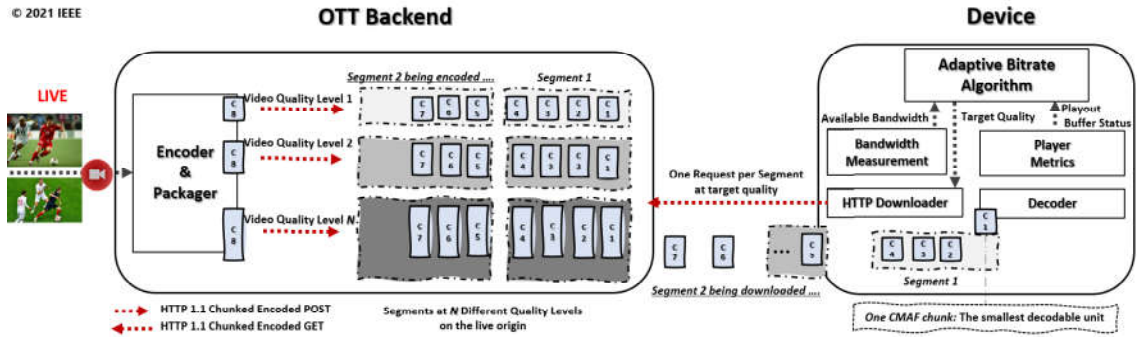


Figure 6.1. End-to-end Flow of HTTP Chunked Encoded Transfer of CMAF segments.

One solution to reduce the live delay is to shorten segment duration. However, it has numerous drawbacks, such as a dramatic increase in the number of HTTP requests and responses, a few multiples of the relevant round-trip times, and excessive rates of switches between different video quality levels [72]. To avoid these drawbacks, HTTP 1.1 chunked encoding transfer and the MPEG Common Media Application Format (CMAF) as the recently introduced media container standard are presented [50]. A segment in CMAF consists of multiple small pieces called chunks, i.e., the smallest decodable units. With the HTTP chunked transfer, it enables the distribution of segments by chunks (e.g., even 100 ms content) while keeping all main advantages of HAS systems. Fig. 6.1 depicts the end-to-end flow of HTTP chunked encoding transfer of CMAF packages in which chunks are encoded and packaged by the adaptive bitrate encoder and packager to different quality levels. Then, the packager output is immediately transferred to the live origin chunk-by-chunk for the distribution. So, chunks can be posted to the network without waiting for the full segment. Players can be simultaneously pulling chunks of the CMAF segment at a specific video quality

chosen by their ABR algorithms in return to one HTTP 1.1 Chunked Get request for the whole segment.

While HTTP chunked encoding transfer of CMAF packages reduces live latency, it brings about QoE degradation. The pillar of the ABR algorithms, bandwidth measurement logic, calculates the available bandwidth as almost equal to the encoded bitrate in live streaming over chunked encoded transfer because they divide the segment size by the download time. The download time is affected by not only the current network status but also idle times at the live encoder end as the chunks of the segment are being prepared during the segment download. However, there is theoretically no exact way in the standard to determine the total idle times to be able to calculate the accurate download rate because the HTTP protocol does not provide the beginning time of each chunk download. Hence, it stops players from switching up the video quality due to the bandwidth underestimation. To mitigate this problem, ABR for Chunked Transfer Encoding (*ACTE*) [51] measures the available bandwidth implementing a sliding window and disregarding the download rate of chunks not close to the moving average. In this chapter, we propose a novel bandwidth measurement heuristic and implement a HAS client which accurately measures the available capacity and chooses the optimal video quality level while, at the same time, avoiding video freezes in low-latency live streaming of CMAF packages. We take into account all chunks in measuring the available bandwidth after approximating the idle times at the encoder side. It allows us to calculate the active download time of the segment limited by only the network status. Then, we exploit a throughput-based ABR algorithm fed with the accurate bandwidth measurement. It is also supported by a conservative mode, which immediately goes to the lowest quality in case of any video stall. Using our real HAS client implementation and live streaming testbed, we perform extensive experiments with various network conditions. The experiments enable us to compare our heuristic with the existing approaches in terms of live latency, average video quality, the number and the duration of video freezes, and the number of the video quality switches during live streaming events. Moreover, this testbed proves that our client can achieve down to one-second live latency without sacrificing QoE.

6.2. Live Streaming HAS Client

6.2.1. Bandwidth Measurement Heuristic

We keep the sizes and the download timings of all the N chunks for a segment in a list as follows: $L = \{cd_1, cd_2, ..., cd_k, ..., cd_N\}$, where L denotes the list of the chunks in the same segment. For each chunk, our knowledge is limited to the download end time and the size of the downloaded bytes by the protocol, represented as $et(cd_k)$ and $s(cd_k)$, respectively. To calculate the network throughput while downloading a segment, we cannot purely use the total segment size and the total download time since some of the chunks are not network-limited as there exist idle times at the live encoder side.

We first calculate the average of the interarrival times between the consecutive chunks in the segment as follows:

$$t_{avg} = \frac{et(cd_N) - et(cd_1)}{N - 1}. \tag{6.1}$$

Second, we consider the chunks downloaded faster than the average time as the network-limited chunks and add them to a separate list (i.e., $L^{'}$), as

$$L^{'} = \{cd_i\},\ s.t.\ et(cd_i) - et(cd_{i-1}) \leq t_{avg}\ ,\ \forall cd_i \in L. \tag{6.2}$$

So, the chunks in $L^{'}$ are assumed as not affected by any considerable idle time at the encoder side. Then, we calculate an approximation coefficient over the chunks in $L^{'}$ to be later used to estimate the time passed in the network for the other chunks, not in this list but L. It is denoted as ρ. It is calculated by dividing the total download time of the chunks in $L^{'}$ by the total size of those chunks as

$$\rho = \frac{T'}{\sum_{cd_i \in L'} s(cd_i)}, \text{where } T' = \sum\nolimits_{cd_i \in L'} et(cd_i) - et(cd_{i-1}). \tag{6.3}$$

Third, we estimate the time spent in transmission for the other chunks using ρ and their sizes as

$$T_{others} = \sum\nolimits_{cd_i \in (L/L')} s(cd_i) \times \rho. \tag{6.4}$$

Finally, we approximate the throughput using the effective download time spent in the transmission and the total segment size, as

$$BW = \frac{\sum_{cd_i \in L} s(cd_i)}{T' + T_{others}}. \tag{6.5}$$

6.2.2. ABR Rules

ABR mechanism in our implemented HAS client relies on two modes: throughput-based video bitrate selection and a conservative mode. With the motivation of achieving the highest video quality without causing any video stalls, we choose the highest video bitrate among the available bitrates less than the available throughput. In case of any video stall, we go to the conservative mode to quickly recover the video freeze by choosing the lowest available video bitrate regardless of the measured bandwidth.

We keep the historical values of the available bandwidth calculated after each segment download. It stores the last K values. In our implementation, we use the default K value in the DASH.js, which is 4. The algorithm uses the average of these values in the window to smooth the available capacity information to avoid a higher number of quality switches, hard and sudden quality drops/jumps. In the throughput-based selection mode, the available capacity for the next video segment download is based on Exponential Weighted Moving Average (EWMA) in Equation 5.1.

6.2.3. Adaptive Playback Rate and Target Live Latency

Our HAS client has a catch-up feature with the adapting playback speed to pull the player back to the target live edge. The player can keep itself tightly to a target latency by controlling the playback rate relying on the assumption that variations in the playback rate of 25% or less are not perceptually noticeable enough to end-users [73]. So, it speeds up or slows down the playback rate within the range of (0.75, 1.25) based on the delta between the target latency and the current live latency. To determine the actual value within this range, we relied on the default implementation in the DASH.js player, which uses a sigmoid function.

6.3. Performance Evaluation

6.3.1. Setup

We leverage DASH.js [27] for the players, and Mininet [56] for the simulation of the network plane. We fit our bandwidth measurement implementation into DASH.js with its throughput-based ABR. We use the test live-streaming content served by the live simulator running on the DASH-IF server [27]. It generates content with CMAF chunks and pushes them over HTTP 1.1 Chunked Encoding. Each segment is eight seconds long and encoded into four quality levels for video at 0.3, 0.6, 1.2, 2.4 Mbps and one bitrate level for audio at 37 kbps. Chunk duration can be set to 0.1, 0.5, and 1 second. Our local Mininet environment with four hosts is connected to the remote test server via the last-mile ISP with the minimum guaranteed capacity of 50 Mbps. To be able to conduct the experiments with various bottleneck capacities, we change the last-hop capacity on the bottleneck node being shared by the four hosts in the Mininet environment. We also run our iperf-based background traffic generator script in one of four hosts for varying time intervals to mimic real-life network fluctuations. We randomize time interval of each background flow within the range of 30 to 60 seconds by getting inspired by the test set in ACM MMSys'20 Grand Challenge [74]. In each interval, we set a random traffic amount between given minimum and maximum values

for the iperf flow. Each network profile used is detailed in the evaluation categories below. In this setup, we compare our approach with the bandwidth measurement heuristics of *ACTE*, *LoL*, and the low-latency mode enabled DASH.js.

6.3.2. Evaluation Metrics

We track the average live latency for each session. The live delay represents the total time from capturing to rendering. We also collect all the QoE experiments used in the evaluation of a conventional HAS client introduced in [75]. We calculate the normalized mean video quality (MVQ) during a live streaming session with m segments, k available quality levels and the played quality level Q_i for segment i. In our case, the played quality level Q_i is mapped to the video quality index in the DASH manifest within the range of available quality levels. MVQ is formulated as

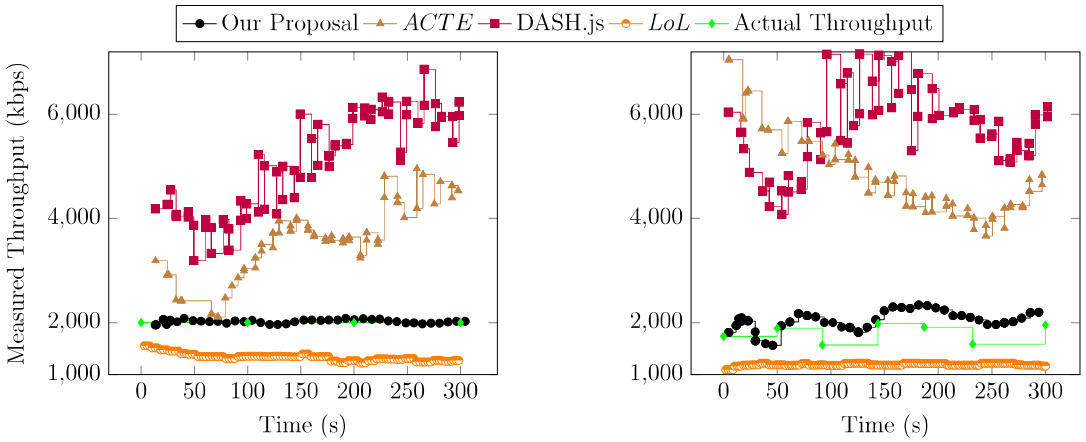
$$\text{Normalized } MVQ = \frac{\sum_{i=1}^{m} \frac{Q_i}{k}}{m}. \tag{6.6}$$

As the other factors with the negative impact on QoE, we follow the duration and number of freezes and the number of quality level switches during the session. The number of video freezes is the number of occurrences when the playback buffer is empty while the number of switches is the total number of quality drops or jumps during the session. The video freeze duration is the total time of a particular stall from the freeze to restarting. To obtain those metrics systematically, we implemented a metric collector script on top of the existing DASH.js player logs.

6.3.3. Comparative Results with State-of-the-art Solutions

In this section, we confront our live streaming HAS client with *ACTE*, *LoL*, and DASH.js in terms of the accuracy in the bandwidth measurements. For a fair comparison of the bandwidth measurement heuristics over QoE implications, we apply the same ABR algorithm and the adaptive playback rate mechanism with the bandwidth measurement modules of *ACTE* and DASH.js under the same conditions, i.e., the

same network characteristics and same live video parameters. We cannot include *LoL* to QoE-based experiments because its bandwidth measurement module is not open-source from their publicly available obfuscated player bundle to apply it to the same setup.



(a) 2-Mbps Link without background traffic

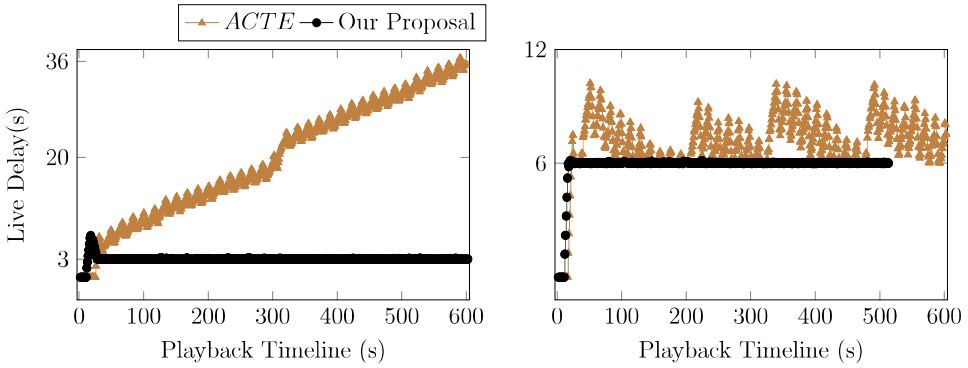
(b) 3-Mbps Link with 1..1.5 Mbps background traffic

Figure 6.2. Measured Throughput by all four approaches in the limited capacity of 2 Mbps and 3 Mbps.

6.3.3.1. Evaluation in measuring the available bandwidth. First, we set the link capacity to 2 Mbps on the last hop in the Mininet environment to mimic the bandwidth-limited networks without any background traffic. Then, we separately run each approach and archive the throughput measured by all the four approaches per each segment download. Each 8-second video segment includes 500ms of chunks. As shown in Fig. 6.2(a), our proposed heuristic measures the available throughput accurately, whereas *ACTE* and DASH.js reference client overestimate it. *ACTE* and DASH.js reference clients aggressively filter out chunks in the throughput measurement process by assuming that their greater download time is due to the encoder-side idle times. Since they can ignore the chunks affected by a limited network capacity, it turns out a throughput overestimation problem. Conversely, *LoL* manifests an underestimation problem as it can over-calculate the download duration. Second, we set the last-hop link capacity to 3 Mbps by adding background traffic, which fluctuates between 1 Mbps

and 1.5 Mbps. So, the available capacity is varying between 1.5 Mbps and 2 Mbps. In this experiment, the same background traffic pattern is applied to each approach separately. As shown in Fig. 6.2(b), the measurement in our proposed heuristic closely follows the available throughput, whereas *ACTE* and DASH.js reference client highly overestimate it. While *LoL* does not have any overestimation problems, it is not as close as our approach to the actual available throughput.

6.3.3.2. QoE implications under 3-second and 6-second target live latency. We repeat the previous experiment ten times for each approach by setting the target live latency to 3 seconds and 6 seconds in the available throughput of 2 Mbps. To investigate the impact of the bandwidth measurement heuristics of our approach, *ACTE* and DASH.js on the QoE, we collect all the metrics explained in Section 6.3.2 for a 10-minute live session.



(a) The target delay of 3 s (b) The target delay of 6 s

Figure 6.3. Live latency variations during a 10-minute live session.

So, in total, we have the metrics from sixty sessions as combinations of ten repetitions, two different target latency values, and three mechanisms.

Table 6.1. All QoE metrics in all three approaches under the target live delays of 3 s and 6 s (over 10 repetitions).

	Target Live Latency (s)	Avg. Live Delay (s)	Avg. Duration per Stall (ms)	Avg. Num. Stalls	Avg. Num. Switches	Normalized MVQ
DASH.js	**3 s**	16.2 ± 1.93	315 ± 5	34.8 ± 0.28	70.4 ± 0.34	0.62 ± 0.002
	6 s	6.71 ± 0.08	320 ± 11	24.4 ± 0.42	69.4 ± 1.12	0.62 ± 0.008
ACTE	**3 s**	18.6 ± 2.34	260 ± 81	34.2 ± 0.27	69.4 ± 0.33	0.62 ± 0.002
	6 s	7.13 ± 0.87	272 ± 95	25.2 ± 0.54	72.6 ± 2.24	0.63 ± 0.01
Our Proposal	**3 s**	3.03 ± 0.02	151 ± 78	2.4 ± 0.33	4 ± 0.17	0.73 ± 0.001
	6 s	5.87 ± 0.03	0	0	4.4 ± 0.43	0.73 ± 0.004

Table 6.1 summarizes all the QoE data collected from those sessions. It reflects that our proposal outperforms the others in terms of all the QoE metrics in both cases. Note that the player waits for one-second content to rebuffer to be able to continue to the playback in each video stall as the DASH manifest file requires 1 second of minimum buffer duration. Because the others cause a high number of video stalls due to the bandwidth overestimation and the selection of the higher qualities, they cannot get closer to the target live latency. Similarly, the conservative mode is triggered more frequently in the others to recover stalls. So, as they select the lowest quality more often, the mean video quality during a session is less than our approach. Furthermore, the reason their performance is worse in the target latency of 3 seconds than the 6-second target latency is that since the buffered video data is less in smaller target values, it is more likely to run out of the video buffer in case of the bandwidth overestimation. Fig. 6.3 presents live latency values during a typical session for $ACTE$ and our approach. As similar to the results in Table 6.1, it indicates that our approach can consistently achieve the target latency during the session, whereas $ACTE$ cannot keep the player tightly to the target. Although the adaptive playback speed feature attempts to pull the player back to the target by slightly increasing the playback rate after each video stall, numerous stalls refrain the player from maintaining the live delay target in $ACTE$. DASH.js results are not included in the figure for clarity because they are almost identical to $ACTE$.

6.3.3.3. Multiple players with arbitrary arrival times. We conduct experiments by using three live streaming players and fluctuating background traffic in the same shared network. We set the last-hop link capacity to 10 Mbps, and add a random background traffic within the range of 5 to 5.5 Mbps. Streaming on each player is started 30 s later than the previous one. We also set the target live latency to 6 s. Then, we collect all the metrics from the three players for 10-minute live sessions. In each 10-minute experiment, all the players apply the same approach. Table 6.2 summarizes all the QoE data collected from those sessions with the mean and 95% confidence interval values. Results are in line with the previous tests that our proposal outperforms *ACTE* and DASH.js reference client. The others are about 30 seconds far away from the live event on average because of higher video freeze rates. In comparison, our proposal can achieve about 8 s live latency even in the existence of heavy background traffic and multiple players. Furthermore, the average normalized video quality is still better in our approach because the others go to the lowest quality more frequently to recover from video freeze events.

Table 6.2. All QoE metrics from three simultaneous players with a fluctuating background traffic (over 10 repetitions).

	Avg.Live Delay (s)	**Avg. Duration per Stall (ms)**	**Avg. Num. Stalls**	**Avg. Num. Switches**	**Avg. Normalized Video Quality**
DASH.js	29.49 ± 1.49	272 ± 18	27.5 ± 0.49	52 ± 1.5	0.66 ± 0.01
ACTE	30.46 ± 1.68	246 ± 26	29.2 ± 0.34	54 ± 1.3	0.66 ± 0.01
Ours	7.97 ± 0.29	242 ± 31	16.1 ± 0.92	33 ± 2.4	0.76 ± 0.03

Table 6.3. The impact of chunk duration on QoE in our approach under 1s target live delay (over 10 repetitions).

Chunk Duration	**Avg.Live Delay (s)**	**Avg. Duration per Stall (ms)**	**Avg. Num. Stalls**	**Avg. Num. Switches**	**Avg. Normalized Video Quality**
0.1s	1.17 ± 0.01	51 ± 16	1.8 ± 0.01	7.8 ± 0.89	0.73 ± 0.002
0.5s	1.46 ± 0.02	172 ± 56	26.1 ± 1.10	21 ± 2.9	0.78 ± 0.01
1s	2.57 ± 0.35	241 ± 59	36.6 ± 1.85	27 ± 2.4	0.76 ± 0.01

6.3.3.4. Ultra-low latency down to one second and impact of various chunk duration. We push the limits by setting the target latency to 1s, and perform new experiments to observe the impact of chunk duration in our approach. We set the total capacity of the bottleneck link to 5 Mbps by generating a random background traffic between 2 Mbps and 3 Mbps. We use three different chunk durations. Table 6.3 highlights the mean and 95% confidence interval values of all the QoE metrics. Despite fluctuating available capacity, our HAS client achieves 1.17 s live delay on average during 10-minute live streaming events, while the chunk duration is 100 ms. As the chunk duration decreases, the number of stalls is dropping and consequently the live delays are going down. This is because fewer data suffices for players to be able to start rendering when the smallest decodable units are getting shorter.

6.3.3.5. Evaluation with real 4G network traces and higher video bitrates. We apply real 4G traces using traffic control command *tc* [76]. After merging all different trace logs with the same-type vehicle into common network profiles [72], we generate three network profiles: 25-min, 57-min, and 84-min sessions with an average bandwidth of {22.8, 31.9, 29} Mbps and standard deviation of {14.6, 17.2, 17.3} Mbps from a train, tram, and bus respectively. The histograms of the distribution of the available capacity for each network profile are depicted in Fig. 6.4. Similar to the existing CMAF-based live streaming services, we also set up our DASH and content servers. We use the Big Buck Bunny with a segment duration of 8 s, and a chunk duration of 200 ms encoded with FFmpeg [77] (through NVENC H.264 on Nvidia GTX 950M GPU) into five quality levels of {1, 2, 4, 8, 15} Mbps at 30 fps with the minimum buffer duration of 1 s in the manifest file. We loop 10-min original content nine times and stream to cover the longest network profile of 84 minutes. We fix the bandwidth to 100 Mbps at the server side and set the target live latency to 3 s at the player side.

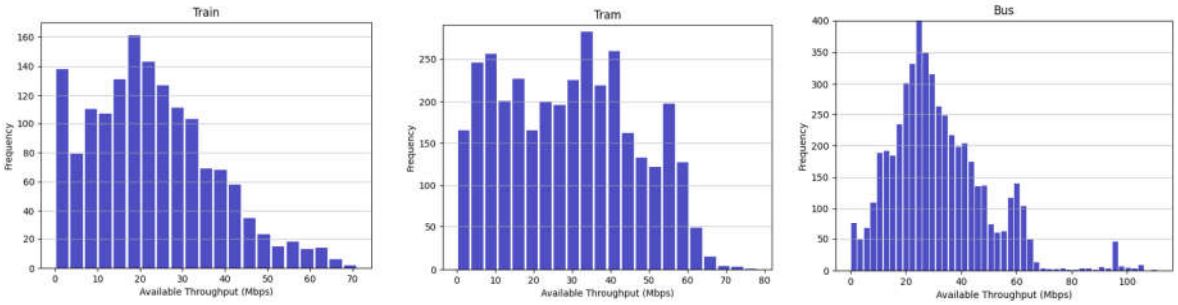
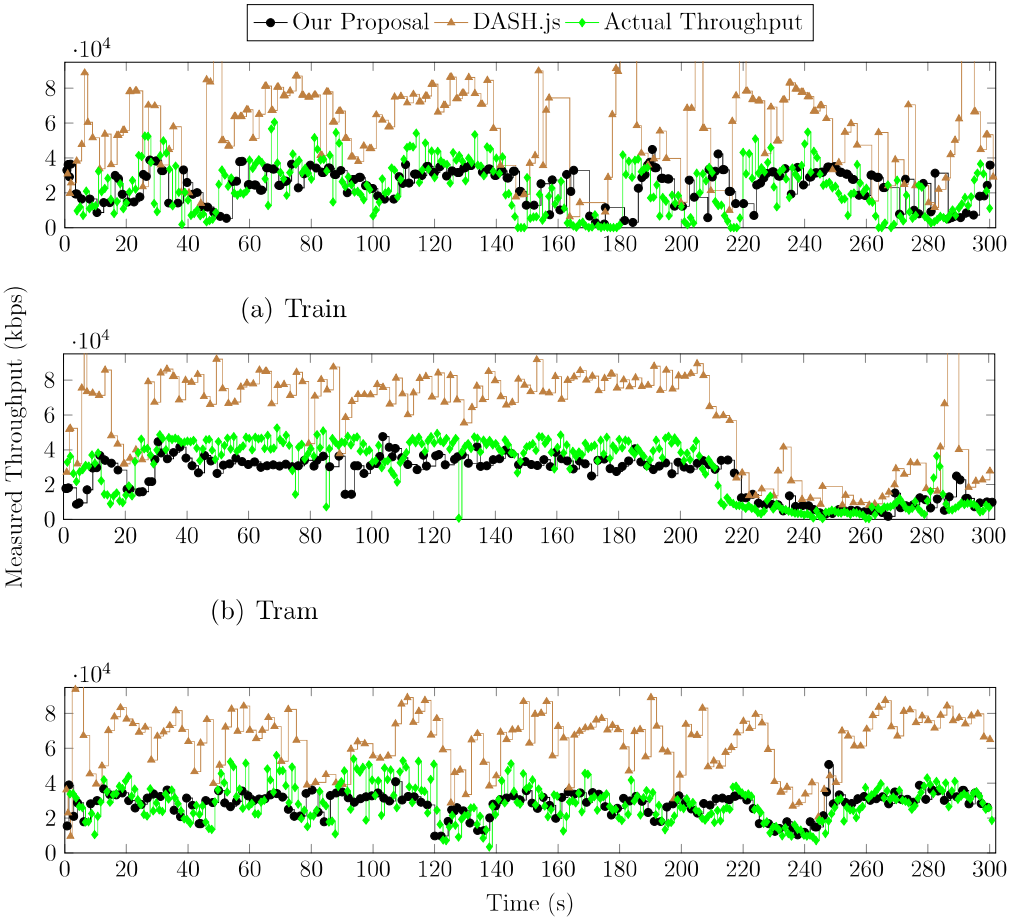


Figure 6.4. Distribution of the available capacity in each network profile.

First, we compare all three approaches in terms of accuracy in bandwidth measurement. Fig. 6.5 depicts the bandwidth measurements in DASH.js and our approach and the real throughput during the first 5 minutes under each network profile. It clearly shows that our approach closely follows the available capacity in all three different network profiles, whereas the default measurement module in DASH.js overestimates. *ACTE* is not included in the figure for clarity as it estimates higher than DASH.js.

Second, we investigate how these bandwidth measurements affect QoE metrics under three network profiles. As shown in Table 6.4, the others underperform our approach in terms of live latency as the bandwidth overestimation problem induces more frequent and longer lasting video stalls in the others. So, our approach gets closer to the target live latency under highly variable network conditions at the expense of a few more quality level switches without considerably sacrificing video quality. It is also reflected in the lower standard deviation of the latency in our approach. Due to some temporary connection interruptions in the real 4G network traces, the maximum latency can reach up to about 50 seconds, while it is recovered down to the target latency with the help of an adaptive playback rate by speeding up the playback in all three approaches.



(a) Train

(b) Tram

(c) Bus

Figure 6.5. Measured Throughput in real 4G traces.

6.4. Discussion

We provided a purely client-based solution to achieve both low latency and high QoE for live events streaming over HTTP. Our extensive experiments through a real player under various network traces showed that our approach allowed reducing the live delay down to 1 s without losing the user's QoE. Furthermore, it is easily applicable to real-life deployments as it is aligned with the stateless design of DASH systems and it does not require any change in the traditional OTT and network backbone.

Table 6.4. QoE metrics from different types of real 4G traces.

	Trace Type	Max. Live Delay (s)	Avg. Live Delay (s)	Std. Live Delay (s)	Num. Stalls	Total Stall Duration (s)	Num. Switches	Normalized MVQ
DASH.js	**bus**	12.01	3.07	0.57	87	36.6	40	0.97
	train	43.74	4.96	5.26	127	90.8	47	0.93
	tram	34.61	5.37	5.96	971	248.6	69	0.85
ACTE	**bus**	19.26	3.16	1.15	205	68.3	15	0.96
	train	46.22	5.27	6.55	345	137.3	22	0.88
	tram	54.54	6.56	9.61	933	269.6	58	0.91
Ours	**bus**	5.09	3.02	0.09	42	18.4	141	0.96
	train	42.14	4.68	5.05	136	76.7	72	0.89
	tram	9.78	3.08	0.46	98	47.6	155	0.91

7. ADAPTIVE LIVE STREAMING USING REINFORCEMENT LEARNING

In this chapter, we present a reinforcement learning framework that learns a strategy to choose adaptive playback speed and video quality for low-latency live event streaming over DASH.

7.1. Introduction

Live event streaming is growing in popularity as over-the-top (OTT) service providers are procuring broadcasting rights of worldwide premium sports events like English Premier League, ATP Tour Tennis, Major League Baseball, American National Football League with 5G infrastructure rolling out more and more. At that point, DASH is the most promising approach for the rapid deployment of the infrastructure of those live events due to its existing wide adoption for Video-on-Demand (VoD) use cases.

In DASH, the multiple quality versions of the same content at different encoding bitrates are kept at the OTT back-end side. Each version is split into small segments. Segment duration and boundaries are the same among all versions of the same content. It enables players to choose the appropriate video quality and switch based on the client-side decision to adapt to the varying network conditions [78].

DASH has been initially designed for VoD to serve previously-stored videos. Hence, legacy DASH players struggle to achieve low latency and a high QoE simultaneously in live sports events. The live latency in this context means the time difference from capturing to rendering a particular moment of the event. The challenge of DASH players is to catch up terrestrial cable latencies in the broadcast world characterized by a 5 to 10-second latency [66] while maximizing viewers' QoE. They buffer a few segments to start playback (e.g., three 10-second segments in Apple HLS, which causes

about a 40-second lag between a viewer's screen and the event). Any stall during the stream also adds to this delay. Such delays ruin the viewer's QoE with a high risk of the spoiler effect. To mitigate, adaptive playback speed, in addition to adaptive video bitrates, is used by relying on the assumption that variations in the playback speed of 10% or less are not perceptually noticeable to viewers [73]. So, the player speeds up or slows down the playback rate within the range of (0.9, 1.1) to keep itself close to a target latency.

In this chapter, we consider adaptive playback speed and video quality decision as a joint optimization problem. We implement Deep Reinforcement Learning (DRL) framework to learn video bitrate and playback speed adaptation strategy to maximize QoE for live video streaming without any assumption about the environment or fixed rule-based heuristics. We also aim to keep live latency low without skipping any content. We compare our approach with state-of-the-art solutions under real 4G traces.

The MPEG CMAF packaging through HTTP 1.1 chunked encoding transfer, the recent advancement in content packaging and delivery, allows DASH to achieve low latency in live streaming. A segment in CMAF consists of multiple small pieces called chunks, i.e., the smallest decodable units. With the HTTP chunked transfer, it enables to distribute segments by chunks (e.g., even 100 ms content) while keeping all main advantages of DASH systems such as quality switching at segment boundaries, leveraging the caches in content delivery networks (CDNs), only one request for each segment and firewall friendliness.

Numerous studies in the literature propose ABR mechanisms to choose the appropriate video quality level and update it depending on changing network conditions to maximize viewers' QoE. For example, Pensieve [79] uses DRL to select video bitrates to achieve a high QoE in DASH after training a neural network model with the observations collected by players in VoD use cases. In [80], Mao et al. deploy enhanced version of Pensieve into Facebook's web player and runs more than 30 million video streaming sessions that have their RL-based ABR mechanisms. Inspired by Pensieve, TCLiVi [81]

uses the same DRL framework for live streaming to decide the video quality level and the target buffer level that players aim to maintain as the minimum content duration required by the player. The buffer level is a proxy for the slow and fast playback speeds, which slows down the payback to avoid a video freeze and speeds up the video player to reduce latency, respectively. CBLC [82] also uses DRL to decide the latency limit to skip content to catch up to the live edge while choosing the video quality and the target buffer level. However, none of them consider CMAF packaging over short-duration chunks and directly choose playback speed as the output of the DRL process. Instead, they select playback speed via hand-crafted thresholds of the delta between the target buffer and the current buffer duration. Furthermore, DASH.js [27] introduces adaptive playback speed heuristics independent from the video bitrate selection in their ABR algorithms. For example, STALLION [83] benefits from DASH.js for playback speed adaption as it drives only video bitrate selection to strive for low-latency live streaming. So, it does not solve a joint decision problem of playback speed and video quality level.

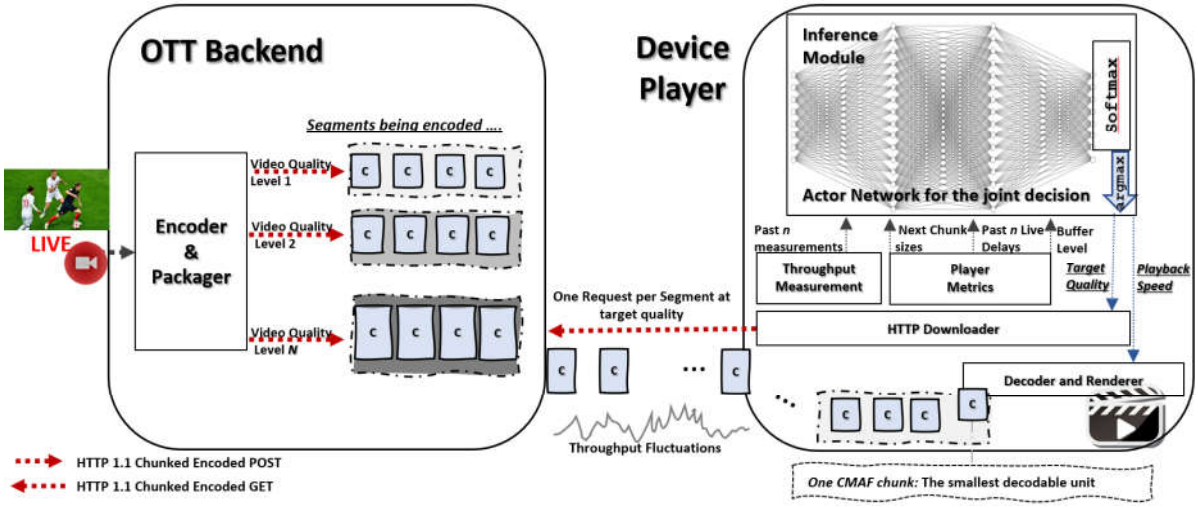


Figure 7.1. The Proposed System of DRL-based Adaptive Live Streaming over HTTP Chunked Encoded Transfer of CMAF segments.

Our main contribution is to perform a joint inference of playback speed and video bitrate selection after training the system to learn a strategy to obtain a high QoE with low latency via our DRL framework. To achieve this, we make the following

contributions: 1) we represent the state space from the key observations of player, the action space of the available video bitrates and three playback speed levels, and a reward function in the form of a combined formula of QoE and live latency; 2) we construct a neural network to provide the best action for a given state to map the state space to a joint decision of playback speed and video bitrate levels for the next video segment. To train the model, we utilize Asynchronous Actor-Critic algorithm (A3C), a state-of-the-art reinforcement learning (RL) algorithm, by introducing another neural network called the critic network, which receives the action taken by the actor and the state space observations to estimate the maximum future award (i.e., the action value). As the value estimator, the critic network is later used to evaluate the action to accelerate to train the actor network; 3) we implement a playback simulator to emulate the adaptive playback speed and video quality selection in live events over DASH. We leverage this simulator in the training process to experience 8 hours playback in only 10 seconds using real 4G traces in one epoch; 4) we perform extensive experiments to confront our approach with both state-of-the-art DRL-based and rule-based solutions.

7.2. System Model and Formulation of The Problem

In this study, the considered multimedia delivery system between DASH clients and servers supports HTTP chunked encoding transfer of CMAF packages. Chunks are encoded and packaged to different quality levels at the server-side. The packager output is immediately transferred to the live origin chunk-by-chunk for distribution. So, chunks can be posted to the network without waiting to encode the whole segment. Note that the content representation and data flow are fully compatible with legacy DASH systems as players are informed about the available video quality levels via the manifest files fetched at the beginning of each session. At any time, players download only one quality representation and they can switch the quality level at the fragment boundaries.

Players can be simultaneously pulling chunks of the CMAF segment at a specific video quality chosen by the inference module as shown in Fig. 3.2, while the content is

being encoded at the OTT backend side. Our objective during this real-time process is to maximize QoE subject to the live latency target by choosing the played video quality level and playback speed for each video fragment. As the contributors to QoE during a live event streaming process are the played video quality, video quality fluctuations, video freezes, and end-to-end latency between capturing and rendering the event moment, we define QoE model as the weighed sum of these four sub-objectives. This objective can be mathematically expressed as the following:

$$\max_{p_t, q_t} \quad \sum_{t=0}^{T} QoE_t,$$
$$\text{where } QoE_t = (c_0 * q_t - c_1 * max(0, (\frac{(buffer_t + FD)}{p_t}) - (FD * \frac{q_t}{BW_t})) \tag{7.1}$$
$$-c_2 * (|q_t - q_{t-1}|) - c_3 * live\_delay_t)$$

s.t.

$$p_t \in \{0.9, 1, 1.1\}, q_t \in \{\text{available video bitrates}\}, \tag{7.2}$$

$$buffer_t < T_{live\_target}, \tag{7.3}$$

where p_t, q_t, $buffer_t$, and BW_t represent the playback speed, the played video quality level, the remaining video duration in the playout buffer, and the measured available throughput at time step t, respectively. $c_{0,1,2,3}$ are the coefficients to set the impact of each sub-objective on the overall QoE in Equation 7.1. FD is the fragment duration as a static value during T-second live streaming. T_{live_target} is the target live latency as a kind of service-level agreement value assigned by the OTT application provider. Constraint (7.2) limits the range of the available playback speed and video quality values, while Constraint (7.3) makes sure players can only buffer content shorter than the target live latency.

The inference module in Fig. 7.1 outputs the adaptive video quality q_t and playback speed p_t given an input set of observations (e.g., the past bandwidth measure-

ments, the remaining buffer size, the current live latency, the next available video segment sizes). The decision mechanism within the inference module relies on a neural network with the policy parameters. The neural network's training process and its design to generate those parameters are elaborated in the proposed RL framework as discussed in the next section.

7.3. The Proposed RL Framework

7.3.1. RL Problem Definition

We model the interactions between an agent and the environment in time-varying network and player conditions as a Markov Decision Process (MDP) represented by a sequence of *state*s, *action*s, and *reward*s. State s_t at a specific time t is the observation of the environment which is enough to determine the next state s_{t+1}. The *action* is taken by the agent to lead to a *state* transition. In our problem, the agent is our inference module that controls adaptive playback speed and video quality. The *reward* r_t is the immediate feedback signal of the environment as a result of the a_t at time step t.

In our RL task, given a MDP we aim to find a *policy*, i.e., a mapping from states to actions, that maximizes the expected sum of future rewards. The policy function $\pi : S \mapsto A$ chooses the action $a_t \in A$ given the current state $s_t \in S$, that gains r_t. The expected sum of future rewards is represented as

$$G_t = \sum_{t=0}^{\infty} \gamma^t r_t, \tag{7.4}$$

where $\gamma \in (0, 1]$ is the discount factor to trade-off present and future rewards and $t = 0$ is the current time. We explain our *state* and *action* spaces, and reward function as follows:

State space: We use the playing video bitrate, bandwidth measurement, the download time of the last segment, the remaining buffer size, the current live latency, the next available video segment sizes, and the number of the remaining segments until the end of the live events as the observation signals to determine the next state.

Action space: It includes $k * P$ combinations, where k is the number of the available video quality levels, and P is the number of playback speed levels.

Reward Function: We represent r_t in the form of a combined QoE expression used by [81]. It is aimed to balance four sub-objectives relying on the coefficients of each QoE contributor such as the played video bitrate, video stall duration, the quality switch compared to the quality of the previous fragment, and latency between capturing and rendering the segment of live event as

$$\begin{aligned} r_t = & c_0 * VQ_t - c_1 * stall\_duration_t \\ & - c_2 * |VQ_t - VQ_{t-1}| - c_3 * live\_delay_t, \end{aligned} \tag{7.5}$$

where VQ_t is the video bitrate at time step t, and $c_{0,1,2,3}$ are the coefficients to set the impact of each sub-objective on the overall QoE.

Environment: The system model of live streaming via DASH in time-varying network conditions in Section II serves as the environment of the proposed RL framework.

7.3.2. The Proposed Actor-Critic Algorithm

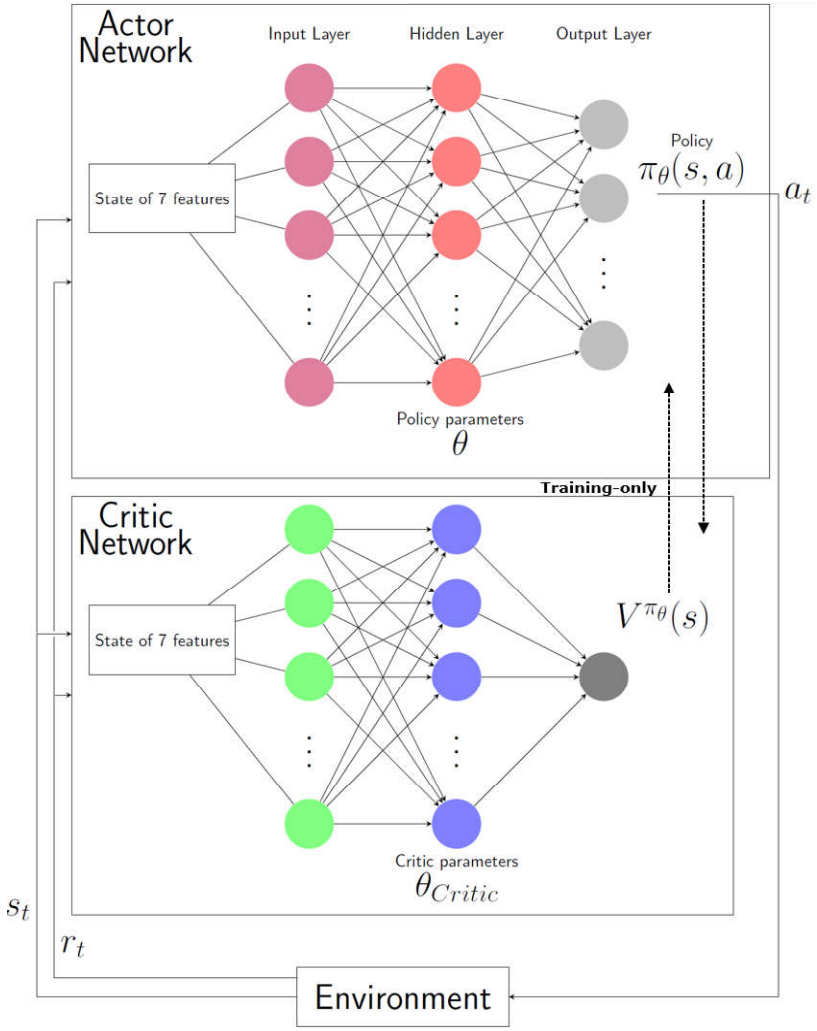


Figure 7.2. The Proposed Actor-Critic Architecture.

As shown Fig. 7.2, we use an actor-critic architecture. The actor outputs the policy π for any state s, a vector of probabilities of each action alternative across $k * P$ combinations. As the state space is at a tremendous scale due to continuous discrete values of input features, we utilize a neural network to approximate the policy π_θ with a limited number of adjustable policy parameters represented as θ. First, using the policy gradient theorem [84], we train the policy for the optimal θ parameters to maximize the expected future reward expressed in Equation 7.4. Then, we follow the gradient

$\nabla_\theta J(\theta)$ of the expected cumulative reward with respect to the policy parameters as

$$\begin{aligned} \nabla_\theta J(\theta) &= \nabla_\theta \mathbb{E}_{\pi_\theta}\left[G_t\right] \\ &= \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s,a) A^{\pi_\theta}(s,a) + \beta \nabla_\theta H(\pi_\theta(s))\right], \end{aligned} \tag{7.6}$$

where $A^{\pi_\theta}(s,a) = Q^{\pi_\theta}(s,a) - V^{\pi_\theta}(s)$. Note that $V(s)$ is the expected future reward before any action is taken in state s, whereas $Q(s,a)$, the Q-value, is the expected reward in state s after action a is performed. $A^{\pi_\theta}(s,a)$, the difference between them, is an indicator of how bad or good is a particular action given a particular state. It is also called the *advantage* value. Inspired by Pensieve [79], we also use an entropy component $H(\cdot)$ with the exploration factor β in Equation 7.6 to trade off exploitation against exploration to obtain better policies as a standard approach in RL. During the training process, the policy parameters of the actor network are updated in each step in the direction of the gradient as

$$\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta), \text{ where } \alpha \text{ is the learning rate.} \tag{7.7}$$

The learning agent empirically calculates the Q-value in our simulated environment after performing a sampled action of playback speed and video quality in the actor network output. As Equation 7.7 needs the estimation of the *advantage* value that depends on $V(s)$, we use the critic network to approximate $V(s)$. To train the critic network, we use a multi-step Temporal Difference learning by getting inspired by [85]. So, the critic network gets the action taken by the actor network and estimates $V(s)$ that is later used to update the policy parameters depending on the *advantage* value.

Once we generate the policy parameters after training two neural networks, we deploy only the actor network with the policy parameters to players. In the inference module, we continuously feed the instantaneous *state* space information to decide the video quality per each segment and playback speed level over time.

7.4. Simulation Results

7.4.1. Experiment Setup

We extend Pensieve's simulator by adding new features to emulate adaptive playback speed and the real-time nature of content streaming in which the player is not allowed to download future content even if the network capacity and the play-out buffer capacity allow. To mimic the environment, we use real network traces [79] collected in Norway from different-type vehicles such as train, tram, metro, bus, car and ferry. Our test video is encoded to $k = 6$ different quality levels at {300, 750, 1200, 1850, 2850, 4300} kbps, and each video fragment is four seconds long, while each live event has a total duration of 20 minutes. Our player simulator supports $P = 3$ playback speed levels at {0.9, 1, 1.1}. So, in our action space, we have $k * P = 18$ discrete alternatives for the joint decision of playback speed and video quality. We set the player buffer limit to one fragment. So, our target live latency is 4 s.

7.4.2. Training Details and QoE Coefficients

Table 7.1. Training Parameters.

Parameter	***Value***
α_{actor} (learning rate)	0.001
α_{critic} (learning rate)	0.0001
β (entropy weight)	4 to 0.2
β_{EPS} (entropy constant)	0.000001
γ (discount factor)	0.99
batch size	48

All the hyper-parameters used in training are shown in Table 7.1. All of them are kept static during the training process except for β. We start with $\beta = 4$ and gradually reduce it to 0.2. The entire training process took about 200,000 epochs to converge. In each step, seven features of the state space explained in Section III are forwarded to the input layer. At the same time, eight past throughput measurements and the next available video segment sizes are first passed into a standard 1D-CNN.

Then, we have a hidden layer of 128 neurons where the results of the input layer go through a fully connected network with the activation function of $ReLU$. Input and hidden layers have the same structure in the actor and critic networks, as depicted in Fig. 7.2, whereas the output layer differs. We use the softmax function in the output layer of the actor network that gives the probability distribution over 18 combinations of playback speed and video bitrate. In contrast, the critic network has a linear neuron in the output layer for the expected future reward. We use TensorFlow to implement and train these networks. We use 80% of the network traces for the training data and keep the rest as an independent validation set.

In the training process, the coefficients of each QoE contributor in the reward function in Equation 7.5 are chosen as the same with TCLiVi [81] for a fair comparison, while c_0, c_1, c_2, and c_3 are set to 1, 1.5, 0.005, and 0.02 respectively. These can be straightforwardly adapted to reflect the different QoE preferences, and the training process can be repeated using the updated reward function and the same parameters in Table 7.1. E.g., if reducing live delay is more critical, the penalty factor of the relevant component (i.e., c_3) can be increased. Rather than a generic QoE formula among users, a personalized QoE metric can be also applied as detailed in [86].

7.4.3. Results

Our approach is compared to the following state-of-the-art algorithms: (1) $TCLiVi$ uses DRL-based adaptive video bitrate and target buffer selection that is later used to choose the playback speed. (2) *Pensieve* uses DRL-based adaptive video bitrate with a fixed playback speed for VoD use case. For a fair comparison, we re-train its model by replacing the reward function with the same QoE formula as in Equation 7.4 used by TCLiVi and our approach. (3) $DASH.js$ is a commercial player developed by DASH Industry Forum that supports low-latency mode with adaptive playback speed. We use a throughput-based ABR version in our comparative analysis. (4) *Baseline* uses a throughput-based ABR with a fixed playback speed.

Table 7.2. Comparison of all five approaches in terms of QoE and live latency.

	5-minute Live Sessions		**20-minute Live Sessions**	
	Avg. Total Reward (QoE)	**Avg. Live Latency (s)**	**Avg. Total Reward (QoE)**	**Avg. Live Latency (s)**
Our Approach	35.8 ± 5.6	6.7 ± 0.2	154.9 ± 12.7	6.3 ± 0.1
TCLiVi	22.7 ± 5.4	5.5 ± 0.2	93.1 ± 12.3	6.1 ± 0.1
Pensieve	21.0 ± 4.3	6.9 ± 2.1	81.5 ± 32.3	7.8 ± 1.0
DASH.js	29.2 ± 9.4	5.9 ± 0.2	119.4 ± 26.2	6.1 ± 0.1
Baseline	30.1 ± 9.4	11.3 ± 0.4	132.3 ± 32.8	13.9 ± 0.3

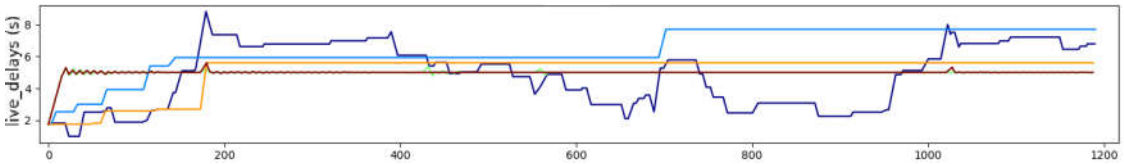
We run each approach under the same network traces in 15 5-min and 20-min live sessions. Table 7.2 summarizes the QoE scores and live delays with the mean and 95% confidence interval values over all the sessions. It clearly shows that our approach outperforms the others in terms of the total reward in the form of a QoE score without sacrificing the live latency considerably. *Baseline* gives a better QoE score across the others despite a higher live latency due to the fixed adaptive playback speed because the QoE formula favors more on the video quality compared to the live delay. As *DASH.js* uses an adaptive playback speed on top of *Baseline* to catch up to the live event, it achieves a lower live latency. *TCLiVi* follows the live events more closely at the expense of 40% less QoE than our approach because it is more conservative in video quality selection. As the total event duration increases, our approach also catches up with the live latency achieved by *TCLiVi*.

We investigate live latency changes over time in each session. Fig. 7.3 depicts representative sessions from each trace group to dive deep into the performance over 20-min sessions. It confirms that our policy learned how to adapt the playback speed as a catch-up feature to pull the player back to the target live edge. It also highlights the impact of adaptive playback speed in *TCLiVi*, *DASH.js* and our approach. After video stalls, they can recover the live latency, whereas *Baseline* and *Pensieve* get stuck at higher live latency values. Fig. 7.3(c) and 7.3(f) perfectly exemplify this situation

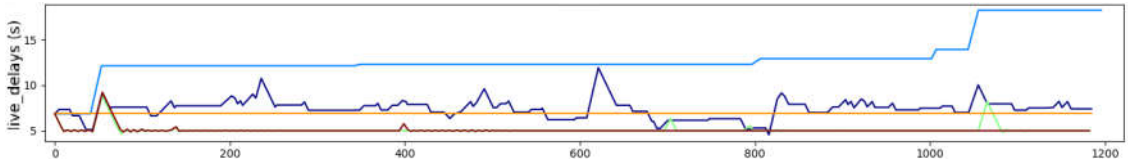
in which *Baseline* and *Pensieve* cannot recover the live latency once they go away because of any video stall.

7.5. Discussion

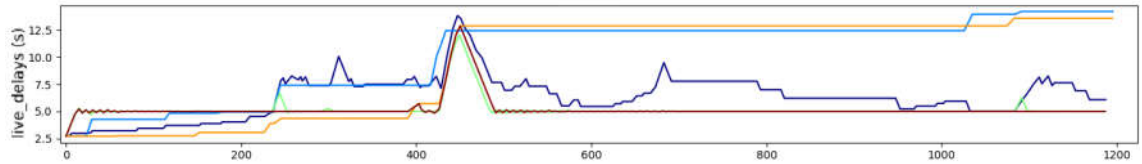
We introduced a novel deep reinforcement learning framework to learn the optimal playback speed and video bitrate decisions from experience. Our comparative evaluation showed that it achieves a better QoE (up to 68%) than both state-of-the-art rule-based and reinforcement learning-based solutions. In future work, while deploying the training policy to real-time players, our bandwidth measurement proposal explained in Chapter 6 could be embedded into the players to improve the overall performance.



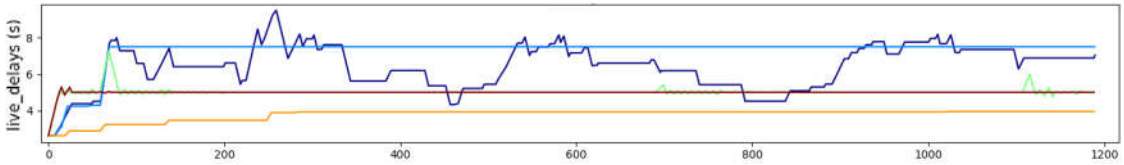
(a) Bus



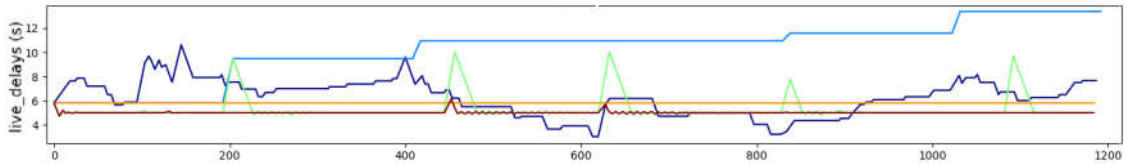
(b) Train



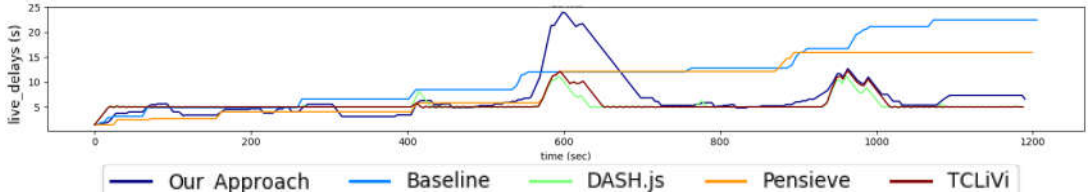
(c) Car



(d) Metro



(e) Ferry



(f) Tram

Figure 7.3. Live latency changes over time.

8. CONCLUSION

This thesis focuses on dynamic adaptive video streaming over HTTP for both on-demand video services and low-latency event streaming. Our motivation is to maximize the QoE of end-users through video quality and playback speed adaptations under highly varying network conditions.

We started by introducing the current state-of-the-art adaptive video bitrate mechanisms for the on-demand video services and low-latency live video streaming over HTTP. We presented an extensive survey of ABR algorithms by classifying taxonomy in terms of their deployed components and input signals after explaining the background information about DASH standard and QoE considerations in video streaming.

As our first contribution, we implemented a chunk-size aware SDN-assisted DASH system without reserving network resources to provide a fair and better QoE in the presence of multiple DASH clients, and background flows in a last-mile bottleneck shared network. It was the first SDN-enabled DASH mechanism in the literature, which takes the diversity of the chunk duration among the various clients into account. It enabled meeting the SLA values for the start-up delay in spite of various chunk sizes among highly varying content catalogs from different content providers. Unlike the state-of-the-art solutions, it supported monitoring the background traffic amount effectively before deciding the adaptive video quality level to improve the perceived video quality and network utilization while avoiding video stalls. The results of our extensive experiments revealed that our proposal outperformed the purely client-based and state-of-the-art SDN-based ABR mechanisms in terms of the overall QoE. Compared to the conventional SDN-based approach, it increased the average video bitrate by over 90% especially in the case with a high number of background flows while, at the same time, guaranteeing a fair start-up delay and not causing video freezes even though it does not require any active network programming for bandwidth slicing. Additionally, the experiments reflected that it decreased video quality oscillations by over

84% compared to the purely client-based ABR approach. In other words, it ensured the QoE stability in the existence of multiple DASH flows in the same shared network, whereas the purely client-based ABR mechanisms had a huge problem in the video quality stability.

As another novel contribution to the literature, we presented a low-latency streaming solution to pragmatically enable dynamic adaptive HTTP live streaming over the existing infrastructure based on CMAF and HTTP 1.1 Chunked Encoding Transfer. With the help of lightweight SDN assistance, our system achieves a better QoE while, at the same time, achieving a low latency within the range of three to six seconds. In contrast, traditional ABR algorithms either condemn users to low video quality or struggle with lots of video stall events because the nature of the HTTP chunked transfer does not allow them to measure the available throughput accurately.

Different from our SDN-assisted live streaming system, we implemented low-latency live streaming HAS client with a novel bandwidth measurement heuristic to ease real-life deployment without introducing any additional component to the legacy DASH system. We conducted experiments on a real player in bandwidth-limited networks. We showed that our approach followed the live target more closely while achieving a lower video freeze rate (-94%) and higher video quality (+16%) compared to the existing approaches. Moreover, it allowed reducing the live delay down to 1 s without losing the user's QoE. Furthermore, it is easily applicable to real-life deployments because it does not require any change in the traditional OTT and network backbone.

As a starting point for our future research direction, we proposed a deep reinforcement learning-based adaptive live streaming solution with two control knobs of playback speed and video bitrate. We also conducted a comparative evaluation, and our results concluded that our approach achieves a better QoE (up to 68%) than both state-of-the-art rule-based and reinforcement learning-based solutions.

As future work, we will investigate the results on the different QoE preferences

by testing with various coefficients of each QoE contributor (e.g., increasing penalty factor for live latency or decreasing the penalty for quality fluctuations). It will also be interesting to deploy our DRL-based solution into a real DASH player as we did in our previous proposed systems.

REFERENCES

1. Quartz, *Amazon's streaming audience is almost as big as Netflix's*, 2021, `https://qz.com/2003812/amazon-prime-has-almost-as-many-streaming-subscribers-as-netflix/`, accessed in June 2021.

2. Cisco, *White Paper: Cisco VNI Forecast and Methodology, 2015-2020, Cisco,San Jose, CA, USA*, Mar. 2016.

3. Corp., E., *Ericsson Mobility Report*, 2019, `https://www.ericsson.com/4acd7e/assets/local/mobilityreport/documents/2019/emr-november-2019.pdf`, accessed in June 2021.

4. Ozcelik, I. M. and C. Ersoy, "Chunk Duration–Aware SDN-Assisted DASH", *ACM Transactions on Multimedia Computing, Communications, and Applications*, Vol. 15, No. 3, pp. 82:1–82:22, Aug. 2019.

5. Ozcelik, I. M. and C. Ersoy, "Low-Latency Live Streaming over HTTP in Bandwidth-Limited Networks", *IEEE Communications Letters*, pp. 1–1, 2020.

6. Kua, J., G. Armitage and P. Branch, "A Survey of Rate Adaptation Techniques for Dynamic Adaptive Streaming Over HTTP", *IEEE Communications Surveys Tutorials*, Vol. 19, No. 3, pp. 1842–1866, thirdquarter 2017.

7. Swaminathan, V., "Are We in the Middle of a Video Streaming Revolution?", *ACM Transactions on Multimedia Computing, Communications, and Applications*, Vol. 9, No. 1s, pp. 1–6, Oct. 2013.

8. Sodagar, I., "The MPEG-DASH Standard for Multimedia Streaming Over the Internet", *IEEE MultiMedia*, Vol. 18, No. 4, pp. 62–67, 2011.

9. Silverlight, M., *Microsoft Smooth Streaming*, 2012, `https://www.microsoft.com/`

silverlight/smoothstreaming/, accessed in June 2021.

10. Apple, *HTTP Live Streaming*, 2011, https://developer.apple.com/streaming/, accessed in June 2021.

11. Adobe, *Adobe Http Dynamic Streaming (Hds) Technology Center*, 2012, https://www.adobe.com/devnet/hds.html, accessed in June 2021.

12. Barman, N. and M. G. Martini, "QoE Modeling for HTTP Adaptive Video Streaming–A Survey and Open Challenges", *IEEE Access*, Vol. 7, pp. 30831–30859, 2019.

13. Seufert, M., S. Egger, M. Slanina, T. Zinner, T. Hoßfeld and P. Tran-Gia, "A Survey on Quality of Experience of HTTP Adaptive Streaming", *IEEE Communications Surveys Tutorials*, Vol. 17, No. 1, pp. 469–492, Firstquarter 2015.

14. Liu, T.-J., Y.-C. Lin, W. Lin and C.-C. J. Kuo, "Visual Quality Assessment: Recent Developments, Coding Applications and Future Trends", *APSIPA Transactions on Signal and Information Processing*, Vol. 2, p. e4, 2013.

15. Wang, Z., A. C. Bovik, H. R. Sheikh and E. P. Simoncelli, "Image Quality Assessment: from Error Visibility to Structural Similarity", *IEEE Transactions on Image Processing*, Vol. 13, No. 4, pp. 600–612, April 2004.

16. Winkler, S. and P. Mohandas, "The Evolution of Video Quality Measurement: from PSNR to Hybrid Metrics", *IEEE Transactions on Broadcasting*, Vol. 54, No. 3, pp. 660–668, 2008.

17. Garcia, M., P. List, S. Argyropoulos, B. Feiten and A. Raake, *ITU-T Rec. P. 1201: Standardized Parametric Packet-based Model for Audiovisual Quality Assessment in IPTV and Progressive Download Services*, 2012.

18. Juluri, P., V. Tamarapalli and D. Medhi, "Measurement of Quality of Experience of Video-on-Demand Services: A Survey", *IEEE Communications Surveys Tutorials*,

Vol. 18, No. 1, pp. 401–418, 2016.

19. Claeys, M., S. Latre, J. Famaey and F. De Turck, "Design and Evaluation of a Self-Learning HTTP Adaptive Video Streaming Client", *IEEE Communications Letters*, Vol. 18, No. 4, pp. 716–719, April 2014.

20. Spiteri, K., R. Urgaonkar and R. K. Sitaraman, "BOLA: Near-optimal Bitrate Adaptation for Online Videos", *IEEE INFOCOM 2016 - The 35th Annual IEEE International Conference on Computer Communications*, pp. 1–9, April 2016.

21. Huang, T., C. Zhou, R.-X. Zhang, C. Wu, X. Yao and L. Sun, "Stick: A Harmonious Fusion of Buffer-based and Learning-based Approach for Adaptive Streaming", *IEEE INFOCOM 2020 - IEEE Conference on Computer Communications*, pp. 1967–1976, 2020.

22. Huang, T.-Y., R. Johari, N. McKeown, M. Trunnell and M. Watson, "A Buffer-Based Approach to Rate Adaptation: Evidence from a Large Video Streaming Service", *SIGCOMM Comput. Commun. Rev.*, Vol. 44, No. 4, p. 187–198, Aug. 2014.

23. Jiang, J., V. Sekar and H. Zhang, "Improving Fairness, Efficiency, and Stability in HTTP-Based Adaptive Video Streaming with FESTIVE", *Proceedings of the 8th International Conference on Emerging Networking Experiments and Technologies*, CoNEXT '12, p. 97–108, Association for Computing Machinery, New York, NY, USA, 2012.

24. Li, Z., X. Zhu, J. Gahm, R. Pan, H. Hu, A. C. Begen and D. Oran, "Probe and Adapt: Rate Adaptation for HTTP Video Streaming At Scale", *IEEE Journal on Selected Areas in Communications*, Vol. 32, No. 4, pp. 719–733, 2014.

25. Qiao, C., G. Li, Q. Ma, J. Wang and Y. Liu, "Trace-driven Optimization on Bitrate Adaptation for Mobile Video Streaming", *IEEE Transactions on Mobile*

Computing, pp. 1–1, 2020.

26. Spiteri, K., R. Sitaraman and D. Sparacio, "From Theory to Practice: Improving Bitrate Adaptation in the DASH Reference Player", *Proceedings of the 9th ACM Multimedia Systems Conference*, MMSys '18, pp. 123–137, ACM, New York, NY, USA, 2018.

27. Forum, T. D. I., *DASH Ref. Player*, Dec. 2017.

28. Kleinrouweler, J. W., S. Cabrero and P. Cesar, "Delivering Stable High-quality Video: An SDN Architecture with DASH Assisting Network Elements", *Proceedings of the 7th International Conference on Multimedia Systems*, MMSys '16, pp. 4:1–4:10, ACM, New York, NY, USA, 2016.

29. Kleinrouweler, J. W., S. Cabrero and P. Cesar, "An SDN Architecture for Privacy-Friendly Network-Assisted DASH", *ACM Transactions on Multimedia Computing, Communications, and Applications*, Vol. 13, No. 3s, pp. 44:1–44:22, Jun. 2017.

30. Bentaleb, A., A. C. Begen and R. Zimmermann, "SDNDASH: Improving QoE of HTTP Adaptive Streaming Using Software Defined Networking", *Proceedings of the 24th ACM International Conference on Multimedia*, MM '16, pp. 1296–1305, ACM, New York, NY, USA, 2016.

31. Bentaleb, A., A. C. Begen, R. Zimmermann and S. Harous, "SDNHAS: An SDN-Enabled Architecture to Optimize QoE in HTTP Adaptive Streaming", *IEEE Transactions on Multimedia*, Vol. 19, No. 10, pp. 2136–2151, Oct 2017.

32. Cofano, G., L. De Cicco, T. Zinner, A. Nguyen-Ngoc, P. Tran-Gia and S. Mascolo, "Design and Experimental Evaluation of Network-assisted Strategies for HTTP Adaptive Streaming", *Proceedings of the 7th International Conference on Multimedia Systems*, MMSys '16, pp. 3:1–3:12, ACM, New York, NY, USA, 2016.

33. Cofano, G., L. D. Cicco, T. Zinner, A. Nguyen-Ngoc, P. Tran-Gia and S. Mas-

colo, "Design and Performance Evaluation of Network-assisted Control Strategies for HTTP Adaptive Streaming", *ACM Transactions on Multimedia Computing, Communications, and Applications*, Vol. 13, No. 3s, pp. 42:1–42:24, Jun. 2017.

34. Bagci, K. T., K. E. Sahin and A. M. Tekalp, "Compete or Collaborate: Architectures for Collaborative DASH Video Over Future Networks", *IEEE Transactions on Multimedia*, Vol. 19, No. 10, pp. 2152–2165, Oct 2017.

35. Georgopoulos, P., Y. Elkhatib, M. Broadbent, M. Mu and N. Race, "Towards Network-wide QoE Fairness Using Openflow-assisted Adaptive Video Streaming", *Proceedings of the 2013 ACM SIGCOMM Workshop on Future Human-centric Multimedia Networking*, FhMN '13, pp. 15–20, ACM, New York, NY, USA, 2013.

36. Bhat, D., A. Rizk, M. Zink and R. Steinmetz, "SABR: Network-Assisted Content Distribution for QoE-Driven ABR Video Streaming", *ACM Transactions on Multimedia Computing, Communications, and Applications*, Vol. 14, No. 2s, pp. 32:1–32:25, Apr. 2018.

37. Go, S. J. Y., C. A. M. Festin and W. M. Tan, "An SDN-based framework for improving the performance of underprovisioned IP Video Surveillance networks", *Journal of Network and Computer Applications*, Vol. 132, pp. 49–74, 2019.

38. Jiang, J., L. Hu, P. Hao, R. Sun, J. Hu and H. Li, "Q-FDBA: İmproving QoE Fairness for Video Streaming", *Multimedia Tools and Applications*, Vol. 77, No. 9, pp. 10787–10806, May 2018.

39. Thomas, E., "Enhancing MPEG DASH Performance via Server and Network Assistance", *IET Conference Proceedings*, pp. 8 .–8 .(1), January 2015.

40. Lu, Z., S. Ramakrishnan and X. Zhu, "Exploiting Video Quality Information With Lightweight Network Coordination for HTTP-Based Adaptive Video Streaming", *IEEE Transactions on Multimedia*, Vol. 20, No. 7, pp. 1848–1863, July 2018.

41. Altamimi, S. and S. Shirmohammadi, "QoE-Fair DASH Video Streaming Using Server-Side Reinforcement Learning", *ACM Transactions on Multimedia Computing, Communications, and Applications*, Vol. 16, No. 2s, Jun. 2020.

42. El Marai, O., T. Taleb, M. Menacer and M. Koudil, "On Improving Video Streaming Efficiency, Fairness, Stability, and Convergence Time Through Client–Server Cooperation", *IEEE Transactions on Broadcasting*, Vol. 64, No. 1, pp. 11–25, 2018.

43. Bentaleb, A., B. Taani, A. C. Begen, C. Timmerer and R. Zimmermann, "A Survey on Bitrate Adaptation Schemes for Streaming Media over HTTP", *IEEE Communications Surveys Tutorials*, Vol. 21, No. 1, pp. 562–585, 2019.

44. Bentaleb, A., A. C. Begen and R. Zimmermann, "QoE-Aware Bandwidth Broker for HTTP Adaptive Streaming Flows in an SDN-Enabled HFC Network", *IEEE Transactions on Broadcasting*, Vol. 64, No. 2, pp. 575–589, June 2018.

45. El Essaili, A., T. Lohmar and M. Ibrahim, "Realization and Evaluation of an End-to-End Low Latency Live DASH System", *2018 IEEE International Symposium on Broadband Multimedia Systems and Broadcasting (BMSB)*, pp. 1–5, IEEE, 2018.

46. Van Der Hooft, J., S. Petrangeli, T. Wauters, R. Huysegems, T. Bostoen and F. De Turck, "An HTTP/2 push-based approach for low-latency live streaming with super-short segments", *Journal of Network and Systems Management*, Vol. 26, No. 1, pp. 51–78, 2018.

47. Yahia, M. B., Y. L. Louedec, G. Simon, L. Nuaymi and X. Corbillon, "HTTP/2-Based Frame Discarding for Low-Latency Adaptive Video Streaming", *ACM Transactions on Multimedia Computing, Communications, and Applications*, Vol. 15, No. 1, pp. 1–23, Feb. 2019.

48. Wang, C., J. Guan, T. Feng, N. Zhang and T. Cao, "BitLat: Bitrate-Adaptivity and Latency-Awareness Algorithm for Live Video Streaming", *Proceedings of the*

27th ACM International Conference on Multimedia, MM '19, pp. 2642–2646, Association for Computing Machinery, New York, NY, USA, 2019.

49. Yi, G., D. Yang, A. Bentaleb, W. Li, Y. Li, K. Zheng, J. Liu, W. T. Ooi and Y. Cui, "The ACM Multimedia 2019 Live Video Streaming Grand Challenge", *Proceedings of the 27th ACM International Conference on Multimedia*, MM '19, pp. 2622–2626, Association for Computing Machinery, New York, NY, USA, 2019.

50. Akamai, *Ultra-Low-Latency Streaming Using Chunked-Encoded and Chunked-Transferred CMAF*, Mar. 2019.

51. Bentaleb, A., C. Timmerer, A. C. Begen and R. Zimmermann, "Bandwidth Prediction in Low-latency Chunked Streaming", *Proceedings of the 29th ACM Workshop on Network and Operating Systems Support for Digital Audio and Video*, NOSSDAV '19, pp. 7–13, ACM, New York, NY, USA, 2019.

52. Lim, M., M. N. Akcay, A. Bentaleb, A. C. Begen and R. Zimmermann, "When They Go High, We Go Low: Low-Latency Live Streaming in Dash.Js with LoL", *Proceedings of the 11th ACM Multimedia Systems Conference*, MMSys'20, pp. 321–326, Association for Computing Machinery, New York, NY, USA, 2020.

53. Bentaleb, A., M. N. Akcay, M. Lim, A. C. Begen and R. Zimmermann, "Catching the Moment with LoL+ in Twitch-Like Low-Latency Live Streaming Platforms", *IEEE Transactions on Multimedia*, 2021.

54. Sun, L., T. Zong, S. Wang, Y. Liu and Y. Wang, "Tightrope Walking in Low-Latency Live Streaming: Optimal Joint Adaptation of Video Rate and Playback Speed", *Proceedings of the 12th ACM Multimedia Systems Conference*, MMSys '21, p. 200–213, Association for Computing Machinery, New York, NY, USA, 2021.

55. Xia, W., Y. Wen, C. H. Foh, D. Niyato and H. Xie, "A Survey on Software-Defined Networking", *IEEE Communications Surveys Tutorials*, Vol. 17, No. 1, pp. 27–51,

Firstquarter 2015.

56. Team, M., *Mininet: An Instant Virtual Network on Your Laptop (or Other PC)*, 2012, "http://www.mininet.org", accessed in April 2021.

57. Fette, I. and A. Melnikov, *The Websocket Protocol*, RFC 4180, 2011.

58. Brown, R. G., "Exponential Smoothing for Predicting Demand", *Operations Research*, Vol. 5, pp. 145–145, 1957.

59. Sobhani, A., A. Yassine and S. Shirmohammadi, "A video Bitrate Adaptation and Prediction Mechanism for HTTP Adaptive Streaming", *ACM Transactions on Multimedia Computing, Communications, and Applications*, Vol. 13, No. 2, pp. 1–25, 2017.

60. Community, T. P. F., *Floodlight is an Open-Source SDN Controller*, Dec. 2012.

61. Mozilla, *Firefox*, 2012, "https://www.mozilla.org/firefox/new/", accessed in May 2021.

62. Tirumala, A., *Iperf: The TCP/UDP Bandwidth Measurement Tool,*, Dec. 2017.

63. Foundation, L., *Open vSwitch: An Open Virtual Switch,*, Mar. 2016, http://openvswitch.org.

64. Lederer, S., C. Müller and C. Timmerer, "Dynamic Adaptive Streaming over HTTP Dataset", *Proceedings of the 3rd Multimedia Systems Conference*, pp. 89–94, ACM, 2012.

65. Lindholm, J., "The Netflix-ication of Sports Broadcasting", *The International Sports Law Journal*, Vol. 18, No. 3, pp. 99–101, Mar 2019.

66. Petrangeli, S., J. V. D. Hooft, T. Wauters and F. D. Turck, "Quality of Experience-

Centric Management of Adaptive Video Streaming Services: Status and Challenges", *ACM Transactions on Multimedia Computing, Communications, and Applications*, Vol. 14, No. 2s, pp. 31:1–31:29, May 2018.

67. Sodagar, I., "The MPEG-DASH Standard for Multimedia Streaming Over the Internet", *IEEE MultiMedia*, Vol. 18, No. 4, pp. 62–67, April 2011.

68. Wei, B., H. Song, S. Wang, K. Kanai and J. Katto, "Evaluation of Throughput Prediction for Adaptive Bitrate Control Using Trace-Based Emulation", *IEEE Access*, Vol. 7, pp. 51346–51356, 2019.

69. Öztürk, E., D. Silhavy, T. Einarsson and T. Stockhammer, "Low-latency DASH-more than just Spec: DASH-IF Test Tools", *Proceedings of the 11th ACM Multimedia Systems Conference*, pp. 353–356, 2020.

70. Spiteri, K., R. Sitaraman and D. Sparacio, "From Theory to Practice: Improving Bitrate Adaptation in the DASH Reference Player", *ACM Transactions on Multimedia Computing, Communications, and Applications*, Vol. 15, No. 2s, pp. 67:1–67:29, Jul. 2019.

71. Le, H. T., N. P. Ngoc and C.-T. Truong, "Bitrate Adaptation for Seamless On-demand Video Streaming over Mobile Networks", *Signal Processing: Image Communication*, Vol. 65, pp. 154 – 164, 2018.

72. van der Hooft, J., S. Petrangeli, T. Wauters, R. Huysegems, P. R. Alface, T. Bostoen and F. De Turck, "HTTP/2-Based Adaptive Streaming of HEVC Video Over 4G/LTE Networks", *IEEE Communications Letters*, Vol. 20, No. 11, pp. 2177–2180, 2016.

73. Kalman, M., E. Steinbach and B. Girod, "Adaptive Media Playout for Low-delay Video Streaming over Error-prone Channels", *IEEE Transactions on Circuits and Systems for Video Technology*, Vol. 14, No. 6, pp. 841–851, 2004.

74. TwitchTv, *ACM MMSys 2020 Grand Challenge*, 2020, `https://github.com/twitchtv/acm-mmsys-2020-grand-challenge`, accessed in June 2021.

75. Claeys, M., S. Latre, J. Famaey and F. De Turck, "Design and Evaluation of a Self-Learning HTTP Adaptive Video Streaming Client", *IEEE Communications Letters*, Vol. 18, No. 4, pp. 716–719, 2014.

76. LDP, *Introduction to Linux Traffic Control*, 2010, `https://tldp.org/HOWTO/Traffic-Control-HOWTO/intro.html`, accessed in November 2020.

77. Bellard, F., *FFmpeg*, 2001, `https://www.ffmpeg.org/`, accessed in December 2020.

78. Mondal, A. and S. Chakraborty, "Does QUIC Suit Well With Modern Adaptive Bitrate Streaming Techniques?", *IEEE Networking Letters*, Vol. 2, No. 2, pp. 85–89, 2020.

79. Mao, H., R. Netravali and M. Alizadeh, "Neural Adaptive Video Streaming with Pensieve", *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, pp. 197–210, 2017.

80. Mao, H., S. Chen, D. Dimmery, S. Singh, D. Blaisdell, Y. Tian, M. Alizadeh and E. Bakshy, "Real-world Video Adaptation with Reinforcement Learning", *arXiv preprint arXiv:2008.12858*, 2020.

81. Cui, L., D. Su, S. Yang, Z. Wang and Z. Ming, "TCLiVi: Transmission Control in Live Video Streaming Based on Deep Reinforcement Learning", *IEEE Transactions on Multimedia*, 2021.

82. Hong, R., Q. Shen, L. Zhang and J. Wang, "Continuous Bitrate & Latency Control with Deep Reinforcement Learning for Live Video Streaming", *Proceedings of the 27th ACM International Conference on Multimedia*, pp. 2637–2641, 2019.

83. Gutterman, C., B. Fridman, T. Gilliland, Y. Hu and G. Zussman, "Stallion: Video Adaptation Algorithm for Low-latency Video Streaming", *Proceedings of the 11th ACM Multimedia Systems Conference*, pp. 327–332, 2020.

84. Xu, J., B. Ai, L. Wu and L. Chen, "Handover-Aware Cross-Layer Aided TCP With Deep Reinforcement Learning for High-Speed Railway Networks", *IEEE Networking Letters*, Vol. 3, No. 1, pp. 31–35, 2021.

85. Sutton, R. S., A. G. Barto *et al.*, *Introduction to Reinforcement Learning*, Vol. 135, MIT press Cambridge, 1998.

86. Gao, Y., X. Wei and L. Zhou, "Personalized QoE Improvement for Networking Video Service", *IEEE Journal on Selected Areas in Communications*, Vol. 38, No. 10, pp. 2311–2323, 2020.

APPENDIX A: COPYRIGHT INFORMATION

In reference to IEEE copyrighted material which is used with permission in this thesis, the IEEE does not endorse any of Bogazici University's products or services. Internal or personal use of this material is permitted. If interested in reprinting/republishing IEEE copyrighted material for advertising or promotional purposes or for creating new collective works for resale or redistribution, please go to *http://www.ieee.org/publications_standards/publications/rights/rights_link.html* to learn how to obtain a License from RightsLink.