DYNAMIC ADAPTIVE VIDEO STREAMING WITH MINIMAL BUFFER SIZES

DISSERTATION

zur Erlangung des Grades des

Doktors der Ingenieurwissenschaften (Dr.-Ing.)

der Naturwissenschaftlich-Technischen Fakultäten

der Universität des Saarlandes

vorgelegt von

Yongtao Shuai

Saarbrücken, 2018

Tag des öffentlichen Kolloquiums:	19. September 2018
Dekan der Fachrichtung:	Prof. Dr. Sebastian Hack
Prüfungsausschuss	
Vorsitzender:	Prof. Dr. Christian Rossow
Gutachter der Dissertation:	Prof. Dr.-Ing. Thorsten Herfet
	Prof. Dr. José Manuel Menéndez
Akademischer Mitarbeiter:	Dr. Pascal Peter

Dynamic Adaptive Video Streaming with Minimal Buffer Sizes

ABSTRACT

Recently, adaptive streaming has been widely adopted in video streaming services to improve the Quality-of-Experience (QoE) of video delivery over the Internet. However, state-of-the-art bitrate adaptation achieves satisfactory performance only with extensive buffering of several tens of seconds. This leads to high playback latency in video delivery, which is undesirable especially in the context of live content with a low upper bound on the latency. Therefore, this thesis aims at pushing the application of adaptive streaming to its limit with respect to the buffer size, which is the dominant factor of the streaming latency.

In this work, we first address the minimum buffering size required in adaptive streaming, which provides us with guidelines to determine a reasonable low latency for streaming systems. Then, we tackle the fundamental challenge of achieving such a low-latency streaming by developing a novel adaptation algorithm that stabilizes buffer dynamics despite a small buffer size. We also present advanced improvements by designing a novel adaptation architecture with low-delay feedback for the bitrate selection and optimizing the underlying transport layer to offer efficient real-time streaming. Experimental evaluations demonstrate that our approach achieves superior QoE in adaptive video streaming, especially in the particularly challenging case of low-latency streaming.

KURZFASSUNG

In letzter Zeit setzen immer mehr Anbieter von Video-Streaming im Internet auf adaptives Streaming um die Nutzererfahrung (QoE) zu verbessern. Allerdings erreichen aktuelle Bitrate-Adaption-Algorithmen nur dann eine zufriedenstellende Leistung, wenn sehr große Puffer in der Größenordnung von mehreren zehn Sekunden eingesetzt werden. Dies führt zu großen Latenzen bei der Wiedergabe, was vor allem bei Live-Übertragungen mit einer niedrigen Obergrenze für Verzögerungen unerwünscht ist. Aus diesem Grund zielt die vorliegende Dissertation darauf ab adaptive Streaming-Anwendung im Bezug auf die Puffer-Größe zu optimieren da dies den Hauptfaktor für die Streaming-Latenz darstellt.

In dieser Arbeit untersuchen wir zuerst die minimale benötigte Puffer-Größe für adaptives Streaming, was uns ermöglicht eine sinnvolle Untergrenze für die erreichbare Latenz festzulegen. Im nächsten Schritt gehen wir die grundlegende Herausforderung an dieses Optimum zu erreichen. Hierfür entwickeln wir einen neuartigen Adaptionsalgorithmus, der es ermöglicht den Füllstand des Puffers trotz der geringen Größe zu stabilisieren. Danach präsentieren wir weitere Verbesserungen indem wir eine neue Adaptions-Architektur für die Datenraten-Anpassung mit geringer Feedback-Verzögerung designen und das darunter liegende Transportprotokoll optimieren um effizientes Echtzeit-Streaming zu ermöglichen. Durch experimentelle Prüfung zeigen wir, dass unser Ansatz eine verbesserte Nutzererfahrung für adaptives Streaming erreicht, vor allem in besonders herausfordernden Fällen, wenn Streaming mit geringer Latenz gefordert ist.

EXTENDED ABSTRACT

As video streaming rapidly gains on popularity and quality, it already represents a dominant share of Internet traffic and its share will constantly grow in the near future. To be effective, video needs to be presented to users in high quality and without interruptions. However, the delivery of such video over best-effort packet-switched networks is challenging due to unknown and time-varying network conditions, including available throughput, delay and losses. A state-of-the-art approach to tackle this challenge is adaptive streaming, in which the video stream is divided into small segments and encoded using multiple quality levels with respect to video characteristics (e.g., resolution and bitrate). This allows streaming applications to respond to varying network by selecting and retrieving the most suitable quality level for each segment to obtain a seamless playback. Adaptive streaming is designed to deliver the highest possible Quality-of-Experience (QoE) — e.g., maximizing the bitrate while minimizing the likelihood of playback stalls (interruptions) and avoiding frequent bitrate switches. Hence, nowadays, it has become ubiquitous in video delivery, especially in Internet-based video distribution.

Despite the facility of dealing with varying network conditions, one of open issues with adaptive streaming is that existing bitrate adaptation only reaches acceptable video quality when buffering several tens of seconds. This leads to high playback latency in video streaming, which is undesirable especially in the context of live content with a low upper bound on the latency. In this thesis, we present an efficient algorithm and architecture to support low-latency streaming. Our goal is to improve the user's QoE in adaptive streaming while reducing the latency. Several contributions to this area of the research are outlined as follows.

In order to cope with efficient streaming systems with respect to low latency, it will be necessary to determine a reasonable lower bound of the latency for adaptive streaming. In our first contribution, we focus on the analytic study of streaming latency. We analyze the components affecting the latency and determine the buffering size as the main component. We present an analytical model of adaptive streaming for buffer size, which can generalize various streaming scenarios concerning the latency. Given the characteristics of a network (e.g., throughput and delay) and the available video profiles (e.g., segment duration and video bitrates), we derive a theoretical minimum of the buffering size required by adaptive streaming and an approximation of the minimum buffering size. This allows us to determine a reasonable low latency for streaming applications.

With the guidelines of the minimum buffering delay, our second contribution addresses the challenge to design an adaptation algorithm which approaches the lower bound requirement of streaming latency. The main challenge is the stabilization of client buffer dynamics, because a small buffer lacks resilience to the buffer fluctuation imposed by the mismatch between the network throughput and the video bitrate of the selected video segment. We tackle this fundamental challenge by developing a novel algorithm that minimizes buffer deviation from the desired level despite a small client buffer and jointly reduces the frequency of bitrate switching.

The proposed algorithm is the most suitable adaptation for low-latency streaming compared to other state-of-the-art algorithms. It achieves the best performance with a buffering size as small as a single segment duration.

The majority of bitrate adaptations do not explicitly take the network delay into consideration during their design. However, the network delay is critical to low-latency streaming. First, it introduces a significant delay for client feedback and thus limits the effectiveness of the adaptation. Second, it incurs the reception delay of consecutive segments, which directly influence the overall latency budget for the streaming. In our third contribution, we present an adaptive streaming architecture designed to address this issue. The key advantage of our architecture is a server-side adaptation based on the throughput and buffer information, which provides a low-delay feedback for the video bitrate selection and near-zero values for the reception delay. Furthermore, we optimize the underlying transport layer of the streaming architecture in order to support predictable delay variation and a high throughput utilization for video streaming. With these refinements, our approach exhibits advanced improvements in terms of user-perceived video quality.

Silent gratitude

— Xiaomei

Gratitude is the memory of the heart.

— Jean Massieu

ACKNOWLEDGMENTS

Many people helped me in this challenging endeavor I took up. To all of them I would like to express my *silent gratitude*!

The greatest gratitude goes to my supervisor, Prof. Dr.-Ing. Thorsten Herfet, for providing me the opportunity, the scientific and monetary foundation to accomplish this research work. Definitely, this work was significantly shaped due to his in-depth comments and the numerous inspiring discussions we had. I also express my gratefulness to all my committee members for their time and willingness being a part of my dissertation defense, especially to Prof. Dr. José Manuel Menéndez for agreeing to serve as an external reviewer and to Prof. Dr. Christian Rossow for being the chair of the committee. This work was further financially supported by the Intel Visual Computing Institute.

I also express my deep gratitude to a number of colleagues and friends at Saarland University, whose research excellence has served as a role model for me and whose wonderful personalities have made a long-lasting impact on me. In particular, thank you, Dr.-Ing Manuel Gorius, who has shown tremendous patience in guiding me and bootstrapping me at the early stage of my video streaming research; thank you, Dr. Goran Petrovic, who has advised me a lot in research and other things, and whose enormous support and encouragement have brought me to the end of the Ph.D. process; thank you, Tobias Lange, for his extensive feedback and detailed comments that significantly improved the presentation quality of my thesis and earlier papers. I would like to thank Dr. Michelle Carnell and other members in the Graduate School of Computer Science for the great event organizations throughout my Ph.D. years and for the financial support during the final thesis write-up stage.

I had the pleasure to work with a number of colleagues and students at the TC Lab: Dr.-Ing Manuel Gorius, Dr.-Ing. Jochen Miroll, Dr.-Ing. Michael Karl, Dr.-Ing. Christopher Haccius, Dr. Goran Petrovic, Prof. Dr.-Ing. Guoping Tan, Dr.-Ing. Zhao Li, Tobias Lange, Jochen Grün, Nasimi Eldarov, Andreas Schmidt, Pablo Gil Pereira, Ramakrishna Mundugar, Frank Waßmuth, Kelvin Augustin Chelli, Harini Priyadarshini Hariharan, Su-A Kim, Manpreet Kaur, Muhammad Arhum Gulzar, Santhosh Nayak, Vinayak Hegde, Dimitrina Krasteva, Vassilena Slaveva, and Praharsha Sirsi. I am glad that we have shared interests, scientific and personal discussions. I also want to give credit to their fruitful cooperation. I would additionally like to thank Diane Chlupka at the lab for providing mission-critical administrative support. Likewise, I owe Zakaria Keshta for keeping the computing infrastructure

running smoothly, as well as for his friendly suggestions for the work and the life. Taking this opportunity, I would like to thank all those people not being listed so far that directly and indirectly contributed to my research work. These also include the anonymous reviewers whose comments helped me considerably in improving both the results and the presentation of the topic.

Without some of the people in my life, I would never have been able to reach the finish line. My gratitude also goes to all my friends accompanying me on the way and my family for their encouragement, support, and love. One needs a lot of them for the long-distance march of pursuing a Ph.D. I dedicate this work to all of you.

CONTENTS

LIST OF FIGURES

LIST OF ALGORITHMS

LIST OF EXCURSIONS

ACRONYMS

3GPP	Third Generation Partnership Project
A3C	Asynchronous Advantage Actor-Critic
ABS	Adaptive Bitrate Streaming
AIMD	Additive-Increase / Multiplicative-Decrease
ANSI	American National Standards Institute
AP	Access Point

AR	Augmented Reality
ARQ	Automatic Repeat reQuest
ATSC	Advanced Television Systems Committee
AVC	Advanced Video Coding
BBR	Bottleneck Bandwidth and Round-trip propagation time
CBR	Constant Bitrate
CDF	Cumulative Distribution Function
CDN	Content Delivery Network
CMAF	Common Media Application Format
CPU	Central Processing Unit
DASH	Dynamic Adaptive Streaming over HTTP
DASH-IF	DASH Industry Forum
DCCP	Datagram Congestion Control Protocol
DVB	Digital Video Broadcasting
ETSI	European Telecommunications Standards Institute
EWMA	Exponentially Weighted Moving Average
FEC	Forward Error Correction
FTV	Free Viewpoint Television
FVV	Free Viewpoint Video
GOP	group-of-picture
GPS	Global Positioning System
HAS	HTTP-based Adaptive Streaming
HbbTV	Hybrid Broadcast Broadband TV
HD	High-Definition
HDS	HTTP Dynamic Streaming
HEC	Hybrid Error Coding
HEVC	High Efficiency Video Coding
HLS	HTTP Live Streaming
HMM	Hidden Markov Model
HSDPA	High Speed Downlink Packet Access

HTTP	Hypertext Transfer Protocol
HVS	Human Visual System
IPTV	Internet Protocol television
ISO	International Organization for Standardization
ISP	Internet Service Provider
ITU	International Telecommunication Union
KAMA	Kaufman's Adaptive Moving Average
MOS	Mean Opinion Score
MPC	Model Predictive Control
MPD	Media Presentation Description
MPEG	Moving Picture Experts Group
MPEG-TS	MPEG Transport Stream
MSE	Mean Squared Error
MSS	Microsoft Smooth Streaming
MVC	Multi-view Video Coding
MVV	Multi-View Video
PCR	Program Clock Reference
PI	Proportional-Integral
PLL	Phase Locked Loop
PRRT	Predictably Reliable Real-time Transport
PSNR	Peak Signal-to-Noise Ratio
QoE	Quality-of-Experience
QoS	Quality-of-Service
QUIC	Quick UDP Internet Connections
RL	Reinforcement Learning
RTCP	Real-Time Control Protocol
RTP	Real-Time Transport Protocol
RTSP	Real-Time Streaming Protocol
RTT	Round Trip Time
SAND	Server and Network Assisted DASH

SCTP	Stream Control Transmission Protocol
SSIM	Structural SIMilarity
SVC	Scalable Video Coding
SVR	Support Vector Regress
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
UHD	Ultra-High-Definition
URL	Uniform Resource Locator
VBR	Variable Bitrate
VoD	Video-on-Demand
VQEG	Video Quality Expert Group
VQL	Video Quality Level
VQM	Video Quality Metric
VR	Virtual Reality

NOTATIONS

a_i	an estimate of the available network throughput experienced during the reception of segment i.
$\tilde{a_i}$	a measured sample of the available network throughput experienced during the reception of segment i.
$avgRTT$	the average of the round trip time.
$b(t)$	the buffer level in seconds at time t.
b_i^e	the buffer level in seconds when the reception of segment i ends.
b_i^s	the buffer level in seconds when the reception of segment i starts.
B_{min}	the minimum of the buffering size for a streaming session without playback stalls, i.e., the minimum buffer level required prior to the playback of a streaming session, such that the client can avoid playback stalls during the entire streaming session.
B_{min}^*	the theoretical minimum of B_{min}.

$\widetilde{B_{min}}$	an approximate value of B_{min}.
B^d	the buffering size, i.e., the amount of video data (in seconds) collected in the buffer before the start of the first video playback.
B_{min}^{deg}	the minimum buffer level required prior to a network degradation, such that the client can avoid playback stalls during the network degradation.
B_{min}^{md}	the minimum buffer level required for the progressive buffer reduction within a time period having multiple degradation events with insufficient degradation intervals.
B_{max}^{sd}	the maximum of all B_{min}^{deg} in a streaming session.
$\overline{B_{min}^{sdi}}$	the average of the minimum buffer levels required for each network degradation in $\mathcal{D}^{sdi}$.
$baseRTT$	the minimum of the round trip time.
$C(t)$	the available network throughput at time t.
C	the average network throughput over the entire streaming session.
C_i	the average network throughput during the entire reception of segment i.
C^{deg}	the average network throughput during a network degradation event.
C_a^{deg}	the average network throughput during the time period within which the remaining reception of a segment succeeding a network degradation event deg is completed.
C_n^{idi}	the average throughput over the n-th insufficient degradation interval.
d_i	the reception duration for segment i, i.e., $d_i = t_i^e - t_i^s$.
d_i^d	the request-response delay for segment i, i.e., the time elapsed from the moment the client requests segment i until the client starts the reception of segment i.
d_i^n	the network delay of segment i.
d_i^r	the delivery time of the request to segment i.
deg	an event of network degradation.
D^{deg}	the duration of a network degradation event, in which the average network throughput over each segment duration is lower than the guaranteed throughput.
D_{min}^{di}	the shortest interval between two network degradation events within the entire streaming session.
D_n^{idi}	the duration of the n-th insufficient degradation interval.
$I(v)$	the index of the video segment to which the video packet v corresponds.
I_{LV}	the impairment function of quality variation.

I_{SD}	the impairment function of startup delay.
I_{ST}	the impairment function of playback stalls.
L	the streaming latency.
M_i	the value of the video quality metric of segment i.
N	the number of segments contained in a video.
N_i	the number of consecutive segments preceding segment i that have the same bitrate as segment i
N_{ST}	the number of rebuffering events.
N^{idi}	the number of insufficient degradation intervals in a streaming session.
OPT	the optimal solution of bitrate selection.
QoE	the Quality-of-Experience function.
r	the average video bitrate of a segment.
r_i	the average video bitrate of segment i.
R	the nominal bitrate of the video (the video quality level).
R_i	the nominal video bitrate of segment i.
$S(v)$	the size of a video packet v in bytes.
S_i^R	the size (in *kbit*) of segment i encoded at the nominal video bitrate R.
t_i^a	the available time of segment i, i.e., the time at which segment i is available at the server and is ready for the delivery.
t_i^d	the time at which the server starts to deliver segment i.
t_i^e	the time at which the reception of segment i ends.
t_i^r	the request time of segment i, i.e., the time at which the client issues a request for segment i.
t_i^s	the time at which the reception of segment i starts.
T	the duration of a video streaming session.
T_c	video segment duration in seconds.
T_d	the delay time of the first video playback.
T_S	the startup delay
T_{ST}	the total rebuffering duration, i.e., the total time of all playback stalls events.
w	the most recent estimate of the congestion window.
w'	the previous estimate of the congestion window.
α	the parameter of the congestion control equation in FAST TCP, which reflects the number of packets queued in the network.

β_l	the lower threshold level in seconds of the buffer region for not switching the video bitrate.
β_{max}	the maximum client buffer level (i.e., the client buffer size) in seconds.
β_{ref}	the desired buffer level in seconds.
β_u	the upper threshold level in seconds of the buffer region for not switching the video bitrate.
γ	the smoothing parameter of the congestion control equation in FAST TCP.
Δt	the reception delay between two consecutive segments.
Δt_{i+1}	the reception delay between segment $i+1$ and segment i, i.e., the time that elapses from the moment the client finishes receiving segment i until the client starts to receive segment $i+1$.
Δt_i^r	the request-response time of receiving segment i, i.e., the time that elapses from the moment the client requests the reception of segment i until the client starts to receive segment i.
Δt_{i+1}^w	the waiting time for which the client waits before the reception of segment $i+1$, if a full buffer event occurs during the reception of segment i.
η	the parameter of the congestion control equation to control the aggressiveness in the throughput acquisition.
λ	the weighting factor for quality variations in an Quality-of-Experience function.
μ	the weighting factor for the startup delay in an Quality-of-Experience function.
ν	the weighting factor for rebuffering duration in an Quality-of-Experience function.
$\Phi(t)$	the accumulated amount of received data (in seconds of video) in the buffer at time t.
$\Phi'(t)$	the accumulated amount of playable received data (in seconds of video) in the buffer at time t.
$\Omega(t)$	the accumulated amount of playout data at time t.
$\mathcal{D}$	the set of all network degradation events in a streaming session.
$\mathcal{D}^{idi}$	the set of sequential degradation events with N^{idi} insufficient degradation intervals.
$\mathcal{D}^{sdi}$	the set of network degradation events which are connected with the shortest interval between two events in the entire streaming session.
$\mathcal{R}$	the set of available nominal bitrates of the video.
$\mathcal{V}_t$	the set of all video packets sent within the duration of t.

1

INTRODUCTION

Video has been an integral part of communications and entertainment applications for many years. With the growth and popularity of the Internet, video delivery over best-effort packet-switched networks such as the Internet has become an important research area. To be effective, video needs to be presented to users in high quality and without interruptions. So the term "streaming" is used to refer to the process of delivering video as a continuous stream of audio-visual data. Nowadays, video streaming has become a dominant traffic type in the Internet and its share will constantly increase in the near future. By 2021, Cisco's Visual Networking Index [1] estimates that video traffic will globally be 82% of all consumer Internet traffic and live Internet video will account for 13% of Internet video traffic. According to Sandvine's data [2], the company found that the combination of streaming audio and video was responsible for over 71% of all evening traffic on North American fixed networks. By 2020, Sandvine predicts an increase to 80%.

1.1 RISING OF INTERNET VIDEO

As a result, the Internet is morphing into a video distribution network for digital entertainment, which competes with traditional television broadcasting for consumers' attention. A new emergent form of television services aims to replicate the traditional television experience offered by cable, terrestrial, and satellite providers. Today, people are watching a wide range of Internet video — from online television or subscription services such as Netflix[1] to free video from platforms like YouTube[2]. In the U.S., 64% of all Internet households had an online subscription service from Netflix, Amazon Prime[3], and/or Hulu[4] in 2017, and 51% of this group had more than one of these services [3]. In addition to television-like services, applications such as surveillance, Free Viewpoint Television (FTV), telepresence, and tele-immersion are leveraging the Internet as the communication platform to an ever increasing extent. People see video virtually every day while scrolling through recent social media such as Facebook[5] and WeChat[6]. User-generated video — including personal live streaming — introduces even more exciting possibilities for them. The appetite for video — both on-demand and live — will further increase with innovative technologies such as Ultra-High-Definition (UHD), Multi-

1 Netflix - Watch TV Shows Online, Watch Movies Online. https://www.netflix.com/
2 https://www.youtube.com/
3 https://www.primevideo.com/
4 Hulu: Stream TV and Movies Live and Online. https://www.hulu.com/
5 https://www.facebook.com/
6 WeChat - Free messaging and calling app. https://www.wechat.com/

View Video (MVV), and Virtual and Augmented Reality (VR and AR) video. VR and AR traffic will increase 20-fold between 2016 and 2021 [1]. Indeed, video is the future of media on the Web and is competing with scheduled linear television content for consumers' attention.

1.2 CHALLENGES IN INTERNET VIDEO

The Internet, however, was not designed to stream video. Traditional television providers exclusively use a communication channel such as a radio frequency to broadcast the video content to everyone within reach. For instance, Digital Video Broadcasting (DVB) is such a well-known system distributing video using a variety of approaches including: cable [4], satellite [5], and terrestrial television [6]. In contrast, the Internet is a best-effort packet-switched network that does not provide any Quality-of-Service (QoS) guarantees. Also, the network for the data transmission is shared among all the users. With video streaming over the Internet, a packet-wise data stream is transmitted to every single receiver. The Internet is thus a very challenging network for delivering delay-constraint data such as video, since, prior to the start of streaming, the state of the network is unknown and hardly predictable. In particular, Transmission Control Protocol (TCP), which governs the Internet traffic, is conventionally regarded as inappropriate for video streaming. The reason is that the congestion control and the retransmission mechanism in TCP can lead to undesirable end-to-end delays, which violate the timeliness requirement for video streaming. Overall, video over best-effort packet-switched network faces a number of challenges including: unknown and time-varying throughput, delay, and losses. A study shows that TCP-based streaming requires the available network throughput to be roughly twice as high as the video bitrate and a startup delay in tens of seconds [7]. Worse still, if the throughput is not sufficient, the video playback will stall due to the buffer running empty — video data is not received in time for its playback. Then, the client needs to re-buffer a sufficient amount of data so as to resume the playback. Furthermore, the demand, the usage, and the connectivity of wireless and mobile networks recently grow significantly with the proliferation of smartphones and tablets. In 2012–2017, the increase of mobile data traffic was recorded with nearly 70% per year, primarily due to increased viewing of video content [8]. The rapid growth and the ubiquity of wireless and mobile networks make the challenges and issues in video streaming even more technically difficult. A mobile user will always experience varying degrees of connectivity, and in some cases the variations can be extreme and long lasting. According to a recent global study [9], 65% of respondents using mobile broadband experience rebuffering problems, 62% experience delayed playback start, and 57% experience low video bitrate problems.

1.3 RISING OF ADAPTIVE STREAMING

Thus, on one hand, traditional fixed-quality video streaming technologies fail to deliver acceptable Quality-of-Experience (QoE), which is a subjective measure from the user's perspective of the overall quality of multimedia services [10], e.g., less playback stalls and better throughput utilization translate to an improved QoE. On the other hand, supporting such an enormous amount of video traffic with an ap-

propriate QoE places a huge burden on the communication network technology, and requires novel solutions in the areas of content distribution, wireless and mobile networking, as well as video streaming. So all significant streaming technologies developed since 2008 have been based on Adaptive Bitrate Streaming (ABS). In ABS, a video stream is divided into small segments, each of which contains a specific duration of video and is encoded with multiple quality levels with respect to video characteristics such as video bitrate and resolution. This enables streaming applications to dynamically adjust the characteristics of the streamed video to a varying network state. Thereby, ABS lessens the playback stalls due to buffer underflow and improves the utilization of the available network throughput. The most prominent examples of ABS include Apple HTTP Live Streaming (HLS) [11], Microsoft Smooth Streaming (MSS) [12, 13], Adobe HTTP Dynamic Streaming (HDS) [14], and MPEG Dynamic Adaptive Streaming over HTTP (DASH) [15, 16]. In particular, DASH has been adopted as a true standard [17] by the International Organization for Standardization (ISO) since 2012. With the demotion of Silverlight[7] and Flash[8] plug-ins in browsers, HLS and DASH dominate the ABS format landscape. The Media Source Extensions [18] framework enables HLS and DASH playback via the HTML5 [19] video tag. Recently, MPEG (Moving Picture Experts Group) issued a standard — Common Media Application Format (CMAF [20]) — which aims to bridge HLS and DASH around a shared segment format. This uniform format reduces deployment overhead and complexity of Internet-based video distribution. The broadcast industry has also adopted ABS for its need. DVB consortium and Advanced Television Systems Committee (ATSC) issued DVB-DASH [21] and ATSC 3.0 [22] specifications, which use DASH for both broadband and broadcast delivery as well as the Hybrid Broadcast Broadband TV (HbbTV [23]).

1.4 CHALLENGES IN ADAPTIVE STREAMING

As of today, ABS is the most prevailing and significant technology in video streaming, especially over the Internet. A huge number of techniques, algorithms and systems have been centered around this concept which aims to offer users the best possible viewing experience on their networks and local hardware. Nevertheless, the viewing experience is not comparable to traditional broadcast: frequent rebuffering events, high latency of video playback, as well as low and/or heavily varying image quality are the most dominant impairments. Extensive modeling and evaluations of bitrate adaptation have revealed that the state-of-the-art adaptive solutions reach satisfactory performance only under buffering of several tens of seconds [24, 25, 26]. For live events this implies a high playback latency for video streaming, which is undesirable especially in the context of the services with a low upper bound on the latency. This performance bottleneck is mainly a result of the underlying TCP transport layer and the biased bitrate adaptation. Moreover, with the increased adoption rate of broadband Internet access — global broadband adoption above 10 *Mbps* was 45% in 2017, a 29% increase compared with one year prior [8] — and the upcoming 5G rollout — across Europe and China by 2020 [27, 28] — we are seeing that users' streaming demand for high-quality viewing experiences will be constantly increas-

7 https://www.microsoft.com/silverlight/
8 https://www.adobe.com/products/flashplayer.html

ing. Likewise, we are seeing the improvement potential of ABS's performance, since users will be expecting the viewing experience to match their Internet connectivity. Consequently, in contrast to existing work, this thesis focuses on the research of a low-latency streaming service with adaptive solutions that fulfill the requirements of live video broadcast. To this end, we design and develop a novel approach for adaptive streaming that minimizes the required video buffers on clients.

1.5 RESEARCH CONTRIBUTIONS AND THESIS OUTLINE

In the following we outline the research problems and the main contributions.

TOWARDS REDUCED LATENCY IN ADAPTIVE VIDEO STREAMING

Although ABS has been a very active research area, a lot of research (e.g., [24, 25, 26, 29, 30, 31, 32]) focuses on the adaptation algorithms, few researchers (e.g., [33]) are interested in how low the latency can be. To fill this gap, the first goal in this thesis is to determine a reasonable lower bound on the latency for video delivery. Such a low bound is absolutely necessary for the potential improvement of streaming solutions in scenarios with low-latency requirements, such as the streaming of live events and augmented vision, and video conferencing. Moreover, it provides us with baselines to design more efficient streaming systems with respect to the latency.

In Chapter 3, our objective is to find out how low a reasonable latency for streaming systems can be, by deriving the lower bound for the buffering size. In particular, we first analyze the latency in ABS and identify the buffering size as the key component affecting streaming latency; then we develop an analytical model of ABS for buffer size, which can accommodate wide range of streaming scenarios, especially in the context of low latency; last, we derive the theoretical lower bound of the buffering size, and introduce an approximation of the minimum buffering size. The related publication can be found in [34].

ADAPTATION ALGORITHM FOR BUFFER STABILIZATION

The allowed latency of streaming systems limits the size of the client buffer in time units (*seconds of video*). State-of-the-art adaptation algorithms (e.g., [24, 25, 26, 29, 30, 31, 32]) are not suitable for *low-latency* ABS due to a lack of explicit stabilization of client buffer dynamics. Specifically, an empty client buffer leads to playback stalls (rebuffering); when the client buffer is at its maximum level, a biased feedback of network throughput is induced that results in suboptimal adaptation decisions. Accordingly, existing streaming systems with a small client buffer size suffer from frequent video playback stalls and/or low image quality.

Chapter 4 tackles the stabilization issue in low-latency ABS by developing a novel algorithm that effectively supports streaming with a small buffering size. The contribution consists of two major lines of work: i.) we present a baseline algorithm for the bitrate adaptation that minimizes the buffer deviation from the desired level despite a buffer as small as a single segment duration, and ii.) further improve the algorithm by applying a simple-but-effective modification that reduces

the heavy variation of image quality. Our algorithm outperforms the state-of-the-art algorithms with at least 6% higher QoE. The research results of this chapter are covered in the following publications: [35, 36, 37].

ADVANCED IMPROVEMENTS IN LOW-LATENCY ADAPTIVE VIDEO STREAMING

ABS performs video bitrate selection based on streaming client's local information such as throughput estimate and buffer occupancy. Most adaptation controllers are deployed at the client side. In practice, the accuracy of bitrate adaptation is limited due to feedback delay and unawareness of the dynamics of the underlying transport layer. In addition, the TCP transport layer introduces severe packet delay variation (packet jitter) and unstable throughput due to its loss- and window-based congestion control, as well as error control with retransmission mechanism [7]. As a result, streaming adaptation is often erroneous and causes client buffer instability such that conventional streaming applications require an extensive buffering on the order of tens of seconds (e.g., [24, 25, 26, 29, 30, 31, 32]).

In Chapter 5, we propose a server-side architecture for ABS, which removes feedback delay from the control loop of bitrate adaptation and provides a hybrid adaptation controller based on throughput and buffer information. Moreover, we deploy the proposed architecture over a new transport-layer protocol in [38] and employ a modified delay-based congestion control from Fast TCP [39] into the transport layer. This enables predictable packet jitter, explicit throughput estimates, and a high throughput utilization for ABS. We evaluate the performance of our architecture with respect to impairment functions that model user-perceived video quality and compare against the well-known streaming architectures. Through intensive experiments, we demonstrate significant improvements with at least 64% lower impairment of stalls and at least 20% lower impairment of quality variations in ABS with buffering sizes as small as the segment duration. Related publications are [40, 41, 42, 43, 37].

2

BACKGROUND

This chapter will briefly go through the prerequisite information on the work presented in this thesis. It starts with an overview of the technology in Internet-based video streaming in Section 2.1. This is followed by the introduction of adaptive video streaming in Section 2.2, including quality switching, playback buffer, streaming behavior, video bitrate, segmentation, standards, and adaptation controller. This chapter ends with the quality assessment of streaming services with respect to quality of experience in Section 2.3.

2.1 VIDEO STREAMING

TV broadcasting (such as cable [4], satellite [5], and terrestrial television [6]) and Internet Protocol television (IPTV) services operate over managed networks for video distribution, which fulfill certain QoS aspects required by these services and support multicast deployment. In contrast, the prevalent streaming technologies are dedicated to delivering the video content over mostly unmanaged networks, i.e., best-effort packet-switched networks such as the Internet. With these streaming technologies, video services such as Dailymotion[1] and Youtube, are usually managing a unicast connection and deliver video streams to the client by using the standard Hypertext Transfer Protocol (HTTP) which relies on the TCP. Other technologies operate with a proprietary streaming protocol running on top of an existing transport protocol, mostly TCP and occasionally User Datagram Protocol (UDP). One successful example of such a streaming protocol is the protocol suite of Real-Time Transport Protocol (RTP) / Real-Time Control Protocol (RTCP) / Real-Time Streaming Protocol (RTSP) [44, 45], which is usually built upon UDP. It provides the means to establish and control the streaming session, as well as to monitor the QoS so that streaming applications are able to deal with delay variations (jitters) and packet reordering/losses. TCP is widely used in commercial streaming systems due to the ease of deployment on existing Content Delivery Network (CDN) architecture for web services via HTTP, and the inherent control mechanisms for network congestion and packet losses, while UDP-based streaming is typically focused on live services, by providing mechanisms for TCP-friendliness and loss recovery [46, 47, 48].

Any Internet-based streaming technology, however, faces an inherent challenge. On one hand, the Internet is a so-called best-effort packet-switched network; i.e., it was not designed to provide any QoS guarantees, such as specified throughput bitrate, network delay, and packet loss rate. On the other hand, the entity of stream-

1 Dailymotion: What to Watch. http://www.dailymotion.com/

ing (i.e., video) by nature has strict timing constraint for its playback. Also, due to users' demand on excellent image quality, video streams are expected to have the highest possible video bitrate and limited packet losses such that visual artifacts are minimized. This conflict of the inherent properties between videos and the Internet leads to a major problem: namely, Internet-based streaming experiences frequent rebuffering events (playback stalls) in the course of video playback. Because when the video data to be played back cannot arrive in time at the client and the data in the buffer is not sufficient to be played back, the client has to pause the playback and wait for sufficient data to be received and stored into the buffer, so as to resume the decoding and the playback. Such a process of pausing the playback and buffering the data is termed as *rebuffering* or *playback stall*[2].

Considerable research effort has addressed the challenge by developing networking architectures [49, 50, 51, 52, 53]. However, none of them has seen wide deployment yet. One reason lies in the complex implementation of QoS models as well as of the trade-off between resource reservation and over-provisioning. Another way to overcome the problem is to dynamically adapt the video characteristics of the streams to varying network states. This leads to the development of ABS, which will be described in more details in the sequel.

2.2 ADAPTIVE BITRATE STREAMING

The general idea of ABS is to allow the bitrate (and consequently the quality) of the video stream to change with respect to currently available resources (on a reasonable timescale). The resource in this context is usually known as network throughput, also known as network bandwidth[3], but other parameters can be taken into account such as screen resolution, battery capacity, and Central Processing Unit (CPU) usage of end devices.

2.2.1 *Dynamic Quality Switching*

First, a video source is divided into small segments, each of which can be processed independently from the others. The client sends requests to retrieve certain segments of the video from a server, and then renders the received video segments while the next segments are being downloaded. The Uniform Resource Locators (URLs) of all segments are collected in a manifest file, which describes the temporal and structural relationships between segments. Second, each of these segments only represents a small duration of the whole video, typically 2–10 seconds long. This enables the client to download only the necessary segments and employ trick modes such as fast-forward, rewind, or seek efficiently. So the process of delivering video is more like a continuous stream of audio-visual data, to which the term "streaming" is used to refer. Third, the small segments are encoded at multiple Video Quality Levels (VQLs) with respect to various video characteristics such as

2 Although playback stalls can be due to multiple reasons — such as system/player crash and playback failure — besides rebuffering, the term *playback stall* refers to the interruption of the playback due to rebuffering throughout this thesis. It is used interchangeably with the term *rebuffering*.

3 In this thesis, the term *bandwidth* will refer to the bitrate in bits per second (*bps*) that can be transmitted, not the signal bandwidth in hertz (*Hz*).

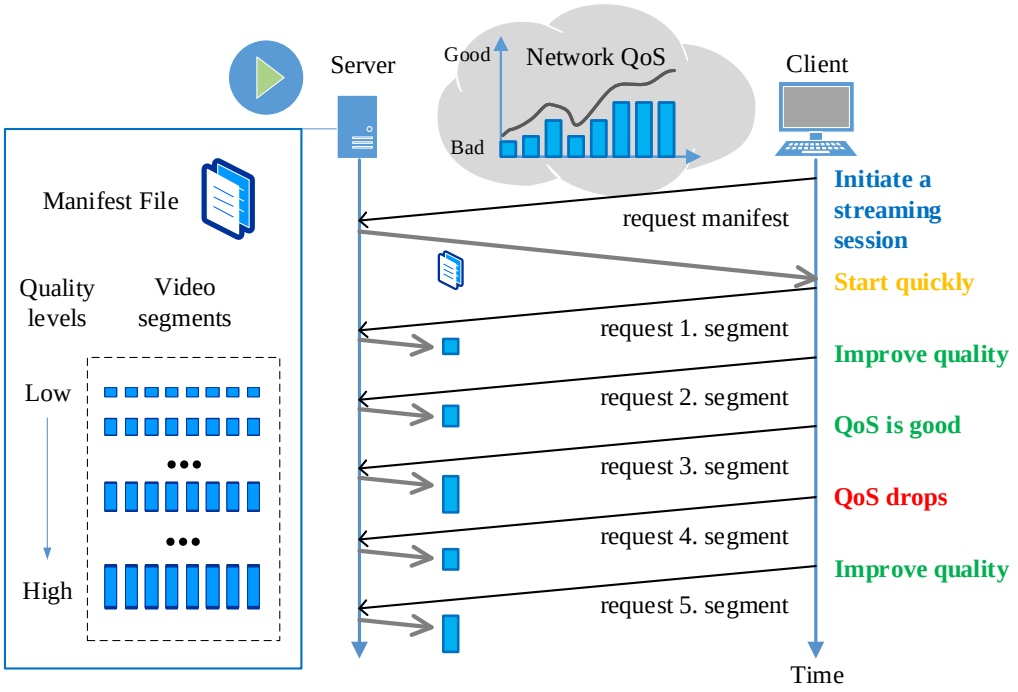


Figure 2.1: Illustration of segment requests and bitrate adaptation in adaptive streaming

the resolution, frame rate, and video (coding) format. Typically, the VQLs differ with respect to their nominal video bitrates, i.e., the average number of bits that are processed in a unit of time over the entire video. So, the quality of the video is often characterized by the nominal video bitrate. Given multiple VQLs for each segment, the client may seamlessly switch between different VQLs at each request. Because the segments are completely self-contained, the client can also perform seamless video playback across segments and VQLs. The streaming system strives to select the best segments to be delivered after examining a variety of parameters related to network QoS (e.g., available throughput and network delay), device capabilities (e.g., display resolution and CPU usage), as well as streaming states (e.g., the occupancy of the playback buffer and server workload). Therefore, the bitrate adaptation for ABS can be formulated as an optimization problem. Its goal is to provide the highest possible QoE, e.g., maximizing the achievable bitrate while minimizing the likelihood of playback stalls and bitrate switches. Figure 2.1 shows an example of ABS, which improves the QoE by dynamically adjusting the VQLs according to the network QoS.

2.2.2 *Client Playback Buffer*

Once video data is received at the client, the playback of the data may start immediately. However, such an immediate playback is infeasible to maintain a continuous playback, because the arrival time of video data is often unable to meet its playback deadline due to packet jitters and the mismatch between the video bitrate and the throughput bitrate of the network. Generally, the client employs a playback buffer (also called client buffer) to store the received data and attempts to keep the buffer occupancy at a safe level, in order to absorb the jitters and the bitrate mismatch.

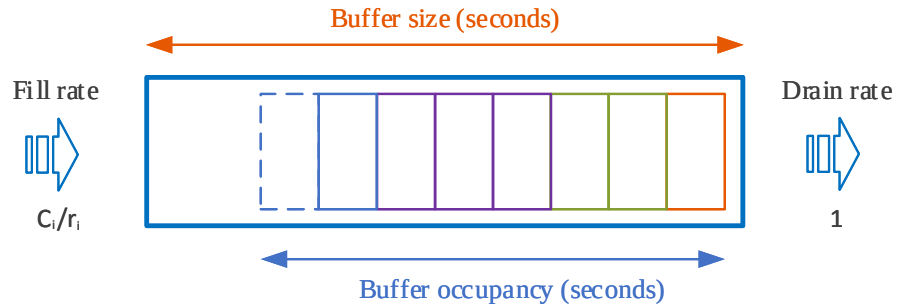


Figure 2.2: Illustration of the client playback buffer, tracked in seconds of video. While every second of video is taken out from the buffer for a continuous playback, a specific amount of video is received and filled into the buffer. The amount can be represented as a ratio between the network throughput and the selected video bitrate, $\frac{C_i}{r_i}$ seconds.

The occupancy of the playback buffer is primarily affected by the choice of video bitrate. Since ABS provides multiple individual VQLs with different video bitrates, the client is able to control the buffer occupancy to a certain extent by dynamically selecting an appropriate VQL for each requested segment. Figure 2.2 illustrates an example of the playback buffer occupancy with respect to the throughput and the video bitrate. The occupancy (level) of the buffer is typically expressed in *seconds of video*. Once the client completes the reception of a video segment, the playback buffer is filled with a duration of video equal to the video duration of the segment. For simplicity, we refer to the video duration of the segment as the segment duration, denoted by T_c. In the course of the *buffering phase*, the buffer level constantly increases along the time. Once the buffering finishes, the client starts the playback and enters to the *playback phase*. In the case of a continuous playback during the playback phase, every second, one second of video is taken from the buffer and displayed to the user, since exact one second of video should be played back every second of real time. At the same time, the client receives $\frac{C_i}{r_i}$ seconds of video and stores it into the buffer, where r_i denotes the video bitrate of the selected segment with index i and C_i the average network throughput during the entire reception of segment i. It implies that the buffer is filled at a rate $\frac{C_i}{r_i}$, while the buffer is drained at a unit rate for a constant playback. For instance, consider $r_i = 2\,Mbps$ and $C_i = 10\,Mbps$, then for each second, the buffer occupancy grows by 4 seconds.

If the ABS's client picks a video bitrate that is smaller than the network throughput, the fill rate of the buffer will exceed the drain rate (i.e., $\frac{C_i}{r_i} > 1$) and the buffer level will rise. If the increase lasts sufficiently long, the buffer level will reach its maximum level that corresponds to the size of the buffer. Generally, the buffer size is measured in a unit of physical memory size, e.g., bytes. In practice, it is expressed in seconds of video, because the buffer level is directly related to seconds of video and bounded to a specific value in seconds of video in the context of video streaming — for both live and on-demand. On one hand, in live streaming, the buffer level (in seconds of video) cannot exceed the value of the streaming latency, because the video content is being generated while being streamed and then the possibility to prefetch content is severely limited. On the other hand, with Video-on-Demand (VoD) services, the complete video content is available for delivery throughout the

streaming session. Consequently, the client may theoretically prefetch the complete rest of the content into its playback buffer while streaming, if the selected video bitrate is sufficiently low compared to the available network throughput. However, even with VoD, the maximum buffer level is generally limited. One reason is the fact that the service providers avoid resource waste, by preventing the client from downloading the content that will not be presented because e.g., the user quits the streaming session or switches to other channels or abandons the prefetched low-quality content if a throughput increase allows the client to request a higher-quality level. Therefore, throughout this thesis, we term the *buffer size* as the maximum buffer level in seconds of video. We define a *full buffer* or *buffer overflow* as the event when the buffer occupancy is at its maximum level.

However, when the fill rate of the buffer is higher than the drain rate, the ABS system probably does not maximize the video bitrate and cannot fully utilize the available network throughput. On the contrary, if the system chooses a video bitrate that is larger than the throughput of the underlying network, the buffer will be filled at a rate smaller than the drain rate (i.e., $\frac{C_i}{r_i} < 1$) and the buffer occupancy will drop. If this situation persists sufficiently long, the buffer will run empty and the occupancy will be at the zero seconds. We call such an event *empty buffer* or *buffer underflow*. If this event occurs during the playback phase, it will incur a so-called rebuffering event. Note that, if rebuffering happens because the network throughput is unable to sustain even the minimum video bitrate, there is nothing an ABS system can do to avoid it, except for buffering sufficient video prior to such an inadequate throughput. Generally, the service providers should ensure sufficiently good network QoS for their users in order to maintain at least the minimum video bitrate while streaming. In practice, the guarantee however does not hold anytime due to e.g., hardware failures, network congestion, or signal degradation on wireless networks. We name an event as *network degradation* in which the available throughput is lower than the minimum video bitrate. Multiple studies (e.g., [54, 55, 56]) show that the frequency and the duration of the rebuffering have the strongest impact on the QoE. Therefore, a well-designed ABS system should keep the buffer occupancy at a high enough level to mitigate the impact of network degradations and should be able to avoid rebuffering events caused by choosing a non-sustainable video bitrate.

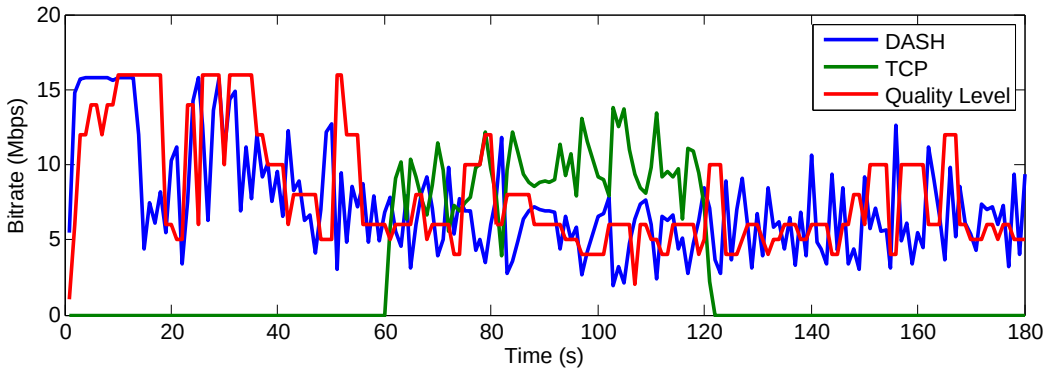
2.2.3 *ON-OFF Streaming Pattern*

While ABS struggles with buffer underflow for preventing playback stalls, it needs to take buffer control at near-maximum buffer levels into account. In case the playback buffer is at its maximum level, no video data can be received by the client and the streaming enters an OFF period. Once the OFF period is past, the streaming reenters an ON period and the client resumes receiving data. Here, one needs to distinguish between VoD and live streaming. With VoD streaming, in case of a full buffer, the client has to wait for the buffer occupancy to reduce to a level which allows further video data to be stored into the playback buffer. With live streaming, a full buffer implies that the streaming latency is near-zero. The unavailability of segments restricts the continuity of the streaming and also incurs the same effect of a full buffer in VoD streaming. The consumer of live streaming cannot receive video

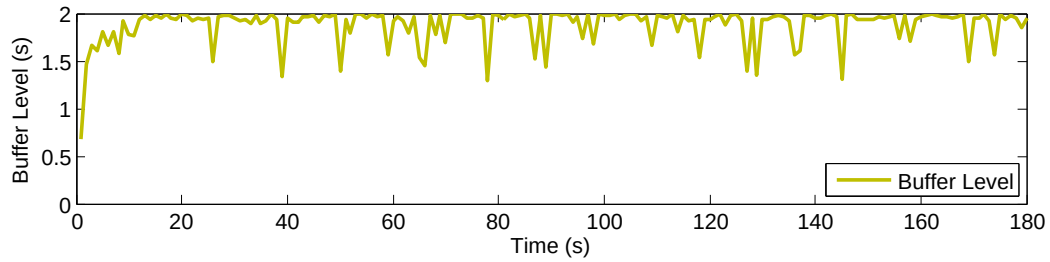
data until a segment is generated and is available for delivery. Therefore, a repeating alternation between ON and OFF periods happens in both streaming scenarios, as long as the network throughput is higher than the selected video bitrate and this condition lasts sufficiently long. Such a periodic phenomenon is called *ON-OFF streaming pattern*. Unfortunately, the ON-OFF streaming pattern poses problems on throughput estimation for ABS [57, 58, 59, 29, 60]. In general, the client estimates the available throughput of the underlying network by measuring the transport-layer throughput during every segment reception and computing a moving average of those measurements over time [61]. This moving average is then used to select the video bitrate for the next segment. Performing throughput estimation under the ON-OFF streaming pattern cause inaccuracies in the estimation.

Akhshabi *et al.* [57] identify that the ON-OFF streaming pattern can lead to quality instability and unfairness when multiple streaming clients compete over the same network bottleneck. Depending on the temporal overlap of the ON-OFF periods among competing clients, they may not estimate the available throughput correctly. For instance, suppose that the ON periods of two clients do not overlap during the reception of a segment. Both clients can perceive the maximum throughput (i.e., the capacity) of the bottleneck and will select a video bitrate that is close to the maximum throughput. If that video bitrate is higher than 50% of the maximum throughput, network congestion will occur while both clients receive a segment with that video bitrate. In this case, the clients will observe that their transport-layer throughput is less than the previous throughput estimate, and then they will switch back to a lower video bitrate. This oscillatory switching behavior can repeat, causing quality instability. Another example is that the ON period of one client spans the ON period of another one. This can take place if one client is receiving a segment with a higher bitrate than the other client. In this situation, the former client will measure a throughput that is more than 50% of the maximum throughput, while the latter one will measure only 50% of the maximum throughput. Suppose that the bitrates currently selected by both clients are appropriate to their throughput estimates, respectively. Then, both clients can converge to a stable but unfair equilibrium in which the client with the higher bitrate keeps selecting the higher bitrate but the client with the lower bitrate stays with the selection of the lower bitrate.

The root cause of the issues studied in [57] mainly lies in the application-layer behaviors initiated by the ON-OFF streaming pattern. In fact, such application-layer behaviors can disturb the transport-layer dynamics, which negatively impacts the application layer reversely. Accordingly, it can form a negative feedback loop. Huang *et al.* [58] observe that the ON-OFF streaming pattern can interfere with TCP dynamics, causing TCP to reenter the slow-start process, because the TCP congestion window may time out due to the inactivity within an OFF period. As a result, the client will acquire a lower throughput than the available throughput and thus suffer from the underestimation of the available throughput. Even worse, bitrate selection based on such low estimates can trigger a *downward spiral effect*, leading to a undesirably low video quality [58]. The worse performance is due to the fact that the underestimation of the throughput is combined with a conservative bitrate selection. A conservative bitrate selection typically picks a video bitrate which is equal to or lower than the throughput estimate. Then, a lower throughput leads the



(a) Throughput and quality level (nominal video bitrate) acquired by DASH and TCP



(b) Client buffer dynamics

Figure 2.3: Bitrate selection based on average HTTP throughput in DASH with 2 s client buffer size and 2 s segment duration. We demonstrate the performance issue of a bitrate algorithm in DASH under the ON-OFF streaming pattern while competing with a greedy TCP background traffic at a network bottleneck with 16 $Mbps$ maximum throughput.

client to select a lower bitrate. When selecting a lower bitrate, the segment would also be smaller in size (bytes). With a smaller segment size, the client becomes more vulnerable to acquire lower throughput. In the worst case, a vicious cycle is created, which brings the video bitrate down to its lowest value.

Figure 2.3 visualizes the behavior of a bitrate selection that is purely based on the measurement of the average throughput at the HTTP/TCP client. It clearly points out the random disturbance of the transport-layer throughput imposed by the adaptation decisions at the application layer. These disturbances cause an excessively poor bitrate selection. The observation reveals that the throughput acquired by the streaming client does not match the available throughput any more and is limited by the average bitrate of video segment instead, once the buffer keeps nearly full. The throughput measurement of the upper layer is not aware of the underlying transport dynamics and determines the available throughput based on the acquired throughput. This incurs fluctuating and biased bitrate selection. The effect is particularly visible during the competition with the greedy background traffic. After the background traffic is switched off, the streaming application significantly underutilizes the network by just acquiring a small share of the available throughput. The reason is that the throughput measurement at the application layer underestimates the available throughput of the network during the ON-OFF streaming behaviors.

2.2.4 *Video Bitrate*

As mentioned before, video streams in ABS are encoded at multiple VQLs, which are typically characterized by the *nominal video bitrate*. The video bitrate is the average rate at which a video is processed, measured in bits per second (*bps*) or more often in kilobits per second (*kbps*) and megabits per second (*Mbps*). The nominal bitrate of the video is the mean video bitrate over the entire video. The quality of the video is determined by various factors such as video resolution (e.g., 480p, 720p, and 1080p), frame rate (i.e., the number of consecutive images called frames that are displayed per second on the screen, i.e., frames-per-second, e.g., 24 *fps*, 30 *fps*, and 60 *fps*.), and video compression technique (also known as video coding technique, e.g., H.262 [62], H.264 [63], and H.265 [64], as well as VP8 [65], VP9 [66], and AV1 [67]). In general, a higher video bitrate will accommodate higher image quality in the video. Even though the image quality will be affected by other factors from above together with the bitrate. For example, at the same video bitrate, video encoded by an advanced codec such as H.264 will look substantially better than an older coder like H.262. Nevertheless, throughout this thesis, we use video bitrate and video quality both interchangeably, and use the nominal video bitrate to represent the VQL of a video (stream).

In ABS, a video stream is segmented into small parts with a specific duration of video. The average (video) bitrate of the segment may vary across the segments with the same VQL (nominal video bitrate), if Variable Bitrate (VBR) encoding is used to generate the video data. VBR encoding is commonly used on modern video coding techniques to produce superior visual quality with a lower nominal bitrate compared to Constant Bitrate (CBR) encoding, even though a video stream contains complex segments e.g., due to a large number of high motion scenes. The primary benefit of VBR encoding is that it allocates a higher bitrate (and therefore more storage space) to the more complex segments of video streams and lower bitrates to less complex segments. Accordingly, VBR encoding produces significantly higher quality at similar nominal bitrate for complex segments than CBR encoding, which maintains the same bitrate for all segments of the video stream.

Although VBR encoding provides a consistent quality throughout the video, it also poses challenges during streaming. The average bitrates of video segments may significantly vary around the nominal bitrate and may exceed the available network throughput. For instance, Figure 2.4 shows the variance of the video bitrate for different quality profiles of the video sequence *Big Buck Bunny* from [68], where the average bitrate of each segment may significantly differ from the nominal bitrate of the segment. In this example, at least 50% segments of each quality profile have an average bitrate of at least 5–45 % higher or lower than the nominal bitrate. The similar effects also happen to other video genres, e.g., sport (*Red Bull Playstreets*) and movie (*Valkaama*) from [68]. Therefore, handling such bitrate variations becomes one influential consideration while designing the ABS system.

2.2.5 *Segmentation*

Segmentation is essential to support streaming. To achieve the seamless playback and switching of segments, each segment is required to be able to be decoded

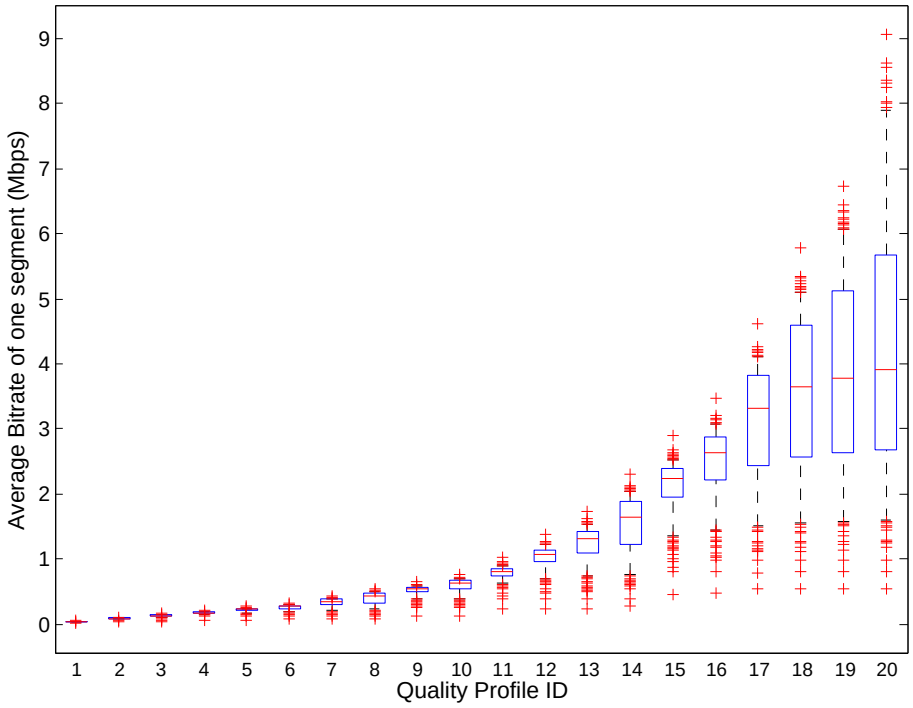


Figure 2.4: Average bitrate of segments in variable bitrate encoding with various quality profiles. The red plus marker denotes the outliers of the 5-th and 95-th percentiles, which are marked with black bars. The blue box contains the bitrate range from the 25-th to the 75-th percentile. The red bar in the center indicates the nominal video bitrate.

independently. Namely, the streaming format of the video must fulfill this requirement of the completely self-contained segment. Modern video encoding employs inter-frame compression: one or more neighboring frames are used to reconstruct a current frame due to the temporal redundancy, only the differences are transmitted. Therefore, it is necessary to ensure that there is an Intra-coded frame (I-frame) at the beginning of each segment, such that clients can seamlessly play back the segments and switch levels between different segments. Because I-frames do not reference any other frames or segments, the client can start the decoding of a segment once its reception is completed. Fixed I-frame positions can be achieved by aligning the group-of-picture (GOP) size of the encoding to the number of frames in a segment; e.g., for a video with a frame rate of 24 fps, a segment with two second video duration may have a GOP size of 24 frames or 48 frames. As a consequence, shorter segment durations result in a lower compression ratio because I-frames have more bits, and thus the overall video quality is worse than the one of longer segments encoded at the same bitrate. However, short segments allow quick adaptability. Therefore, this trade-off needs to be balanced in content generation for adaptive streaming. There are some research efforts (e.g., [69, 70, 71]) to optimize the segment duration. Typically, the segment duration varies between 2 seconds and 10 seconds.

The SP/SI-frame extension to H.264 [72] introduces two new frame types for an alternative to I-frame when applications perform functionalities such as stream switching, splicing, and random access. SP-frames utilize motion-compensated pre-

diction similar to Predicted frame (P-frame), while SI-frames apply only spatial prediction as I-frame and are used in conjunction with SP-frames. SP-frames have significantly higher compression ratio than I-frames while providing similar functionalities [72]. They are usually larger than P-frames by approximately 70% [73]. As a result, SP/SI-frames enable more switching points in a video stream without significantly sacrificing the compression ratio.

2.2.6 *MPEG-DASH*

Currently, one of the most successful applications of ABS — HTTP-based Adaptive Streaming (HAS) — runs on top of HTTP/TCP, although the idea of ABS can be applied to other streaming protocols, e.g., UDP-based adaptive streaming. The prime benefit of HAS is the leverage of an ubiquitous and highly optimized network infrastructure. It consists of two aspects. First, using HTTP, HAS is flexible to be deployed on CDNs including data centers and their proxy servers (e.g., caches), which were originally developed for web services to optimize their performance e.g., with respect to load balancing and response time. Second, the usage of HTTP/TCP eases the development of services and applications due to no further requirements for congestion and error control, which are inherently built in TCP. Considering the advantages of HTTP-based streaming, three dominant companies — Microsoft Corporation, Apple Inc., and Adobe System Inc. — released their commercial HAS solutions — MSS [12, 13], HLS [11], HDS [14], respectively. Despite the wide adoption and commercial success, these solutions are incompatible to each other, even though they use the similar technology behind HAS [74]. In response to the scattered landscape, a standard to facilitate the interoperability became indispensable. The first specification of an HAS standard was initiated by the Third Generation Partnership Project (3GPP) and was published in TS 26.234 release 9 [75] in 2009. In collaboration with 3GPP, MPEG published an international standard ISO/IEC 23009-1 [17] in 2012, called DASH, also known as MPEG-DASH. The European Telecommunications Standards Institute (ETSI) released the technical specification of MPEG-DASH profile for DVB services [21] in 2015, which defines the delivery of TV content via HAS.

DASH specifies XML and binary formats that allow a normative client to retrieve video streams from any normative server, thereby enabling consistent playback and unification of servers and clients of various vendors. The DASH specification mainly defines two formats:

- The Media Presentation Description (MPD) provides sufficient information (e.g., the program timing, segment availability, bitrates, and resolutions) for a DASH client to establish a streaming service for the user. In particular, it defines formats to announce resource identifiers for segments and to provide the context for these identified resources using a hierarchical data model.

- The segment formats specify the formats of the entity body of the HTTP response to the DASH client's request. Segments contain video data and/or metadata to decode and render the included video streams. DASH is video codec agnostic and supports segment-container formats for both ISO Base Media File Format [76] and MPEG-2 Transport Stream [77].

Details on content provisioning, the delivery of the MPD and the segments, as well as normative client behavior for adaptation algorithms, fetching and playing content, are outside of the scope of the DASH standard. Therefore, DASH enables high flexibility for various use cases and delivery scenarios.

Today, DASH is gaining more and more deployment, accelerated by content providers and broadcasters such as Netflix, Youtube, and BBC[4]. Many DASH-enabled tools have been developed such as DASH VLC Plugin [78], dash.js[5], DASH-JS [79], dash.as[6], and libdash[7] [80]. The DASH Industry Forum[8] (DASH-IF) (a group including the leading streaming companies) drives the rapid adoption and tremendous research in and around MPEG-DASH. DASH-IF strives for interoperability of products and services, and compatibility with consortia standards, by providing publicly available implementation guidelines, test datasets, and software.

2.2.7 *Adaptation Controller*

Now that ABS needs to adapt its behavior to the variably available resource, the most important functions of ABS are how to detect or predict those resources accurately and then to react to the change adequately. An abstract model of common ABS adaptation is depicted in Figure 2.5. An adaptation controller takes some information about the resources (e.g., typically available network throughput and/or buffer occupancy) as inputs, and then based on a specific algorithm it performs the bitrate selection for the next segment(s) to be delivered. Generally, the information of the available throughput is unknown. Thus, an estimation technique is typically adopted to predict the throughput for the delivery of the next segment(s). The simple way is to use a smoothing method such as averaging based on measured throughput history. Since most information is the local information of the client and the network QoS involves client's network states, the adaptation controller are usually located at the client. Due to the delay between the client request and the server response, the client-side controller (e.g., [78, 29, 30]) is employed in many high-latency streaming services, while in the context of low-latency streaming, a server-side controller (e.g., [81, 82, 83]) is favorable.

A centralized controller is suggested to be deployed over a domain plane of CDNs, such that the best CDN and bitrate for a client can be chosen using data-driven prediction on a global view [84].[9] Due to the varying network conditions and the unawareness of network dynamics, the individual adaptation controller has difficulties to maintain a stable video bitrate and a fair share of available network resources. A network-assisted idea is proposed to further optimize the network resource allocation, and thus to improve efficiency and fairness. There are ongoing efforts (e.g., [85, 86, 87, 88, 89, 90]) to develop frameworks for such network-assisted adaptation controllers. An active-assisting example is to place a proxy server between client and server (in gateways or routers) in order to drive the bitrate

4 British Broadcasting Corporation. http://www.bbc.com/
5 https://github.com/Dash-Industry-Forum/dash.js
6 https://castlabs.com/open-source/dashas/
7 https://github.com/bitmovin/libdash
8 DASH Industry Forum. http://dashif.org/
9 It should be noted that the focus of such a centralized controller is on the optimization of CDNs, instead of an individual client. Therefore, it is out of the scope of this thesis.

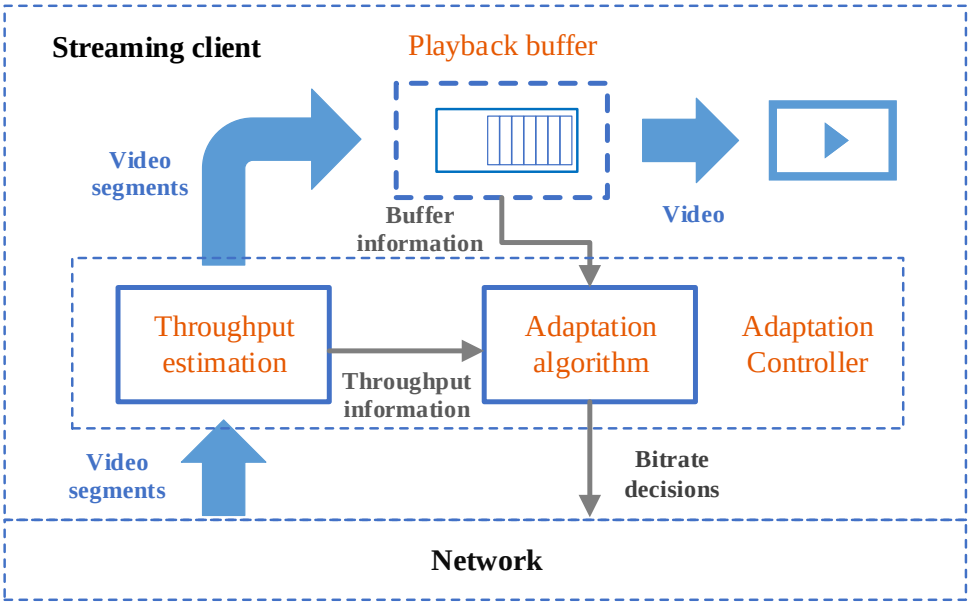


Figure 2.5: An abstract model of common ABS adaptation

adaptation by correcting bitrate requests [89]. Another example, in which a network controller is implemented to provide throughput reservation and bitrate guidance [86], gives clients more degrees of freedom for adaptation decisions. A recent extension to the DASH standard called Server and Network Assisted DASH (SAND [87]) specifies the communication between the assisting network elements and the clients as well as the communication between the assisting network elements.

2.3 QUALITY OF EXPERIENCE

The adaptation controller dynamically determines video qualities for a video stream, in order to offer the best streaming quality for users. In making this decision, there are many potential considerations of the quality — including network and video factors — a client must account for. Thus, it is crucial to define and identify those factors. On the other hand, when evaluating the performance of an ABS system, it is essential to define the goals it has to achieve. These goals are usually involved with various quality factors. Identifying these goals and expressing them in a way that facilitates objective measurement are also very important. In this section, we are going to describe those quality factors.

In the early days of Internet-based streaming, streaming services are assessed by using QoS metrics of the network such as the packet loss rate and delay variation. Later, it has been recognized that video streaming quality must unavoidably take into account user perception. The notion of the QoE was introduced in an effort to assess user's perceptual video quality. The International Telecommunication Union (ITU) defines QoE as "the degree of delight or annoyance of the user of an application or service. It results from the person's evaluation of the fulfillment of his or her expectations and needs with respect to the utility and/or enjoyment in the light of the person's context, personality and current state" [91, 92]. More succinctly, QoE is a measure of the perceptual QoS from the user's perspective. Be-

cause user perception is subjective and diversified in different environments, QoE is conventionally obtained from subjective test, where human viewers rate the quality of tested videos in a controlled laboratory environment. The QoE in those tests are commonly expressed using the Mean Opinion Score (MOS), which is an average over all individual scores on a predefined scale that a subject assigns to his opinion of the system quality [92]. Although the MOS can provide an overall measurement of user perception, subjective tests are often costly, offline, and not scalable. Therefore, many studies (e.g., [93, 94, 95]) have been focusing on objective quality models to predict QoE based on measurable influencing factors without human involvement. Subjective test results are often used as ground truth to validate the performance of the objective quality models. Most objective models even rely on subjective test results to determine model parameters. A wide adoption of such models is thus limited because of their complexity and indirectness. Since the dramatic development of Internet-based streaming makes large-scale data available for analyzing QoE, data-driven analysis models [96] have recently emerged and raised research interest. The massive data collection enables to replace sophisticated models with simpler models and map individual influencing factors to metrics such as the *user engagement* — a quantitative reflection of user involvement and interaction [97, 56, 98].

In the last part of this section, we first describe the QoE assessment in more details and then discuss the influencing factors of the QoE in ABS.

2.3.1 *Assessing Quality of Experience*

There is an immense number of factors influencing the QoE. Based on the classification of influence factors discussed in [91, 99], we focus on two types of factors — *network-related* and *media-related* —, which are used in the present work. We remark that taking into account other types of influencing factors (such as context and human influence) can potentially enhance the QoE. However, their adoption in existing technologies is restricted, due to their complexity, subjectivity, and the lack of models for integrating them into a quantified metric.

Network-related factors refer to data transmission over a network. The network characteristics mainly include throughput, delay, packet loss, and their variations over time. These factors are highly correlated with the network QoS, and thus are often termed network QoS, despite a broader scope of the QoS definition [92].

In consideration of economical and resource-limitation reasons, video content captured in high quality inevitably needs to be compressed in order to be transmitted over a network. To some degree, the compression can be performed without a visible distortion. Since there usually exists a substantial gap between the network throughput and the video bitrate, compressing video content with a transmittable bitrate often unavoidably results in a discernible quality degradation e.g., blocking artifacts, blurring, and motion jerkiness. Media-related factors refer to video configuration parameters of such as compression technique, coding format, resolution, frame rate, and sampling rate.

If a streaming service runs over a managed network (e.g., IPTV), many network-related factors are controllable. In this case, both network-related and media-related factors can be jointly optimized to offer a high QoE. In contrast, the Internet only

undertakes its "best effort" to deliver every packet in a quick manner but without any QoS guarantees. Therefore, Internet-based streaming services will and often do experience variations of both network-related and media-related factors. In that case, the variations of media-related factors are relative restrainable, compared to network-related factors, because network states of the Internet are typically unknown and varying. In order to deliver the best possible QoE, video characteristics (parameterized by media-related factors) are dynamically adapted to the varying network states. ABS is an example of this adaptation. Such an adaptation can be generalized as adjusting media-related and/or network-related factors that are controllable to those that are not.

The influence factors of the QoE we identified above are objective, quantifiable, and measurable. In view of this fact, they were used to assess the quality of video streaming services in the early days. Nevertheless, they cannot accurately reflect the QoE, although they could greatly impact the QoE. In the following, we discuss some methods of QoE assessment with respect to subjective tests, objective quality models, and data-driven analysis models.

2.3.1.1 *Subjective Tests*

With subjective tests, the QoE is measured by soliciting users' opinions in a laboratory environment. The opinions are frequently expressed using an MOS on a scale of 1 ("bad") to 5 ("excellent") [100]. ITU provides a reference for performing the subjective assessment of media quality [101] and the Video Quality Expert Group (VQEG) describes detailed plans for conducting subjective tests [102]. Though, being regarded as a direct and relative accurate way of evaluating QoE, subjective tests have three major limitations. First, they are costly in terms of time, money, and human resources. Second, they cannot evaluate the QoE on the fly, e.g., in the case of dynamic services. Third, they are carried out in a controlled environment, with limited test videos, test conditions, and interviewee demography.

2.3.1.2 *Objective Quality Models*

In order to address these issues, objective quality models were introduced. Most of them are based on the way the Human Visual System (HVS) perceives and processes video signals. Accordingly, they identify the objective and measurable factors that influence user perceptual quality, and map these factors to the QoE.

One commonly used method is to quantify the physical difference between the reference and target (distorted) video, and to weigh the errors according to spatial and temporal features of the video. Two basic examples are the Mean Squared Error (MSE) and the Peak Signal-to-Noise Ratio (PSNR). Due to the physical significance and the simplicity, such models are used in many studies (e.g., [103, 104, 105]) and usually serve as the benchmark. However, simply based on a pixel-to-pixel comparison of video data, they do not incorporate any HVS features in their computation, thus are poorly correlated to the perceived quality measurements [106, 107].

In contrast to a comparison without considering the content, one objective method — called Structural SIMilarity (SSIM) index [108] — takes into account the fact that the HVS is highly sensitive to structural information and distortion from a scene. Another method is the Video Quality Metric (VQM) [109], which was adopted by

American National Standards Institute (ANSI) and ITU as a standardized method of objective video quality measurement. VQM consists of objective parameters for measuring the perceptual effects of several video impairments such as blurring, jerky motion, global noise, block and color distortion, as well as error blocks. VQM is optimized to achieve a reasonably good correlation with subjective results.

Many efforts have been made to develop QoE prediction models that do not need to assess the reference video. These models (e.g., [110, 111, 112, 113]) leverage network statistics (e.g., the packet loss and network throughput) and application-specific factors (e.g., video bitrate and spatio-temporal features), in order to characterize the relationship between these measurements and the QoE estimation. The independence from the reference video allows QoE prediction models to meet the demand of online QoE monitoring.

Nevertheless, the aforementioned methods are proposed to measure the visual quality of a video impaired by packet losses and other artifacts, rather than by an adaptively streamed video (in ABS). Therefore, an direct application of these method is not suitable for ABS and new criteria should be considered in order to evaluate the QoE of ABS. Moreover, the majority of objective quality models employ subjective tests as ground truth to validate their performance. Even the optimization of their model parameters are based on subjective results.

2.3.1.3 *Data-Driven Analysis Models*

As Internet-based Streaming has become more and more popular, great amounts of data (with respect to e.g., content types, quality metrics, and user behavior) has been collected for the analysis of streaming services. Recently, data-driven QoE analysis has emerged as a promising way to circumvent the challenges in other methods. It drives a shift from user "experience" metrics (e.g., PSNR, VQM, and subjective MOS) to user "engagement" metrics [97, 56, 98] (e.g., the viewing time, the number of watched videos, abandonment rate, and the return rate). On the other hand, based on data-driven analysis, the engagement metrics can be derived from the quantifiable and measurable metrics concerning e.g., startup delay, rebuffering, and video bitrate. As a result, it enables to develop simple and reliable QoE prediction models by leveraging big data. It is worth noting, that user engagement is an important metric, which is of particular interest for content providers, because it can be directly translated into providers' business objectives e.g., the revenue of advertisements and subscriptions [97, 56].

2.3.2 *Quality of Experience for Adaptive Bitrate Streaming*

As ABS has been widely accepted as the technology basis of Internet-based streaming, the QoE for ABS has become a topic of prime importance in academia and industry [114, 115, 97, 116, 74, 117, 118, 119, 120]. The core factors influencing the QoE for ABS include: the playback stalls (in terms of e.g., the duration and the frequency), the bitrate trajectory (representing the selected bitrates for the individual video segments in sequence), the startup delay, and the latency (especially in the case of live streaming). In the following, we explain each of these influencing factors in more details.

2.3.2.1 *Playback Stalls*

When the playback buffer of the streaming client has been depleted and the new video segment does not arrive before its playback deadline, the playback of the video must temporarily stop and an event called playback stall occurs. This is often known as a buffer underflow. A buffer underflow is usually followed by a rebuffering period, in which the client accumulates enough video data in the buffer to resume playback. Thus, playback stall is also termed as rebuffering. The conditions required for resuming the playback depend on the rebuffering strategy of the client. The amount of rebuffered data needs to be traded off between the duration of a stall event and the risk of future shortly-recurring stall events. A too-short rebuffering duration may result in insufficient amounts of rebuffered data. Then, more potential stall events may recur too soon.

Furthermore, in live streaming, playback stalls increase the live delay since the subsequent segments are played back later than their scheduled playback deadlines. Either, users need tolerate such a progressive delay, or the client has to fulfill the subscribed latency constraint by e.g., skipping the playback of some delayed video frame(s), or even delayed segment(s).

Many studies find that rebuffering is a significant metric to measure the QoE [54, 121], and has the greatest impact on the user engagement [55, 56, 122]. The development of ABS is primarily motivated just by avoiding the negative impact of stall events on the QoE. The impact is mainly determined by both the duration and the frequency of stall events [54]. Users who experience stall events longer, will quit the view earlier [55, 56]. If stall events are inescapable, less-but-longer stalls are preferred to frequent-but-shorter stalls [95, 118].

2.3.2.2 *Bitrate Trajectory*

The image quality of a video is another important factor influencing the QoE. As mentioned in Section 2.2.4, the image quality is typically characterized by the video bitrate. The bitrate delivered in ABS is the key metric in determining the image quality. Consequently, the sequence of selected bitrates — referred to as *bitrate trajectory* here, also known as adaptation trajectory — embodies all the features that dramatically affect the overall QoE by its influence on the image quality of the individual video segments. Apparently, choosing a video bitrate as high as possible for segments will yield a maximum QoE with respect to the image quality. Academia and industry use the average bitrate as the standard metric to measure the video quality and define it as the average of the bitrates played during a streaming session [97, 123, 124]. In that way, the higher the average bitrate is, the better the video quality is. Ahmed *et al.* [122] find that not only the rate of rebuffering but also average bitrate have the most impact on user engagement in live streaming. We remark that other metrics such as PSNR, SSIM, and VQM can also be used to represent the video quality. For instance, Liu *et al.* [117] proposes to use VQM due to the good correlation with human perception and the heterogeneous setting of the video provided by streaming services.

Recent works (e.g., [125, 95, 126]) have found that, not just average bitrate, but also bitrate switching can influence the QoE. Two aspects must be accounted for: the amplitude and the frequency of the bitrate switching. Although highly frequent

bitrate switching may impair the QoE, gradual variations with multiple switches are generally preferable to abrupt variations with a few switches [125, 85, 95]. The authors of [117] observe that the impairment caused by an increasing bitrate switch is much lower than that caused by a decreasing switch, and thus suggest to omit the impact of increasing switches on their proposed impairment functions. This can also be interpreted as an observation that users would rather have constant bitrate than frequently varying bitrate, even though the average bitrate is lower [125, 114].

2.3.2.3 *Startup Delay*

The startup delay or initial delay is the time duration from a playback request being initiated till the first start of a video playback. It typically includes the time taken to download the manifest file (e.g., MPD file in MPEG-DASH), and the time required to complete the reception of the partial video. In this thesis, the startup delay excludes the time for receiving the manifest file. The reasons are two-fold. First, a client does not necessarily need the manifest file for a playback, e.g., in scenarios of a server-side adaptation architecture. Second, the size of the manifest file is often much smaller compared to the partial video required for the playback; especially, the manifest file for the initialization can be packed in a very small size so as to reduce the delay.

The startup delay is always present in streaming services, as the client has to wait until a specific amount of data is downloaded and stored into the playback buffer in order to begin the decoding and the playback. This phase is sometimes called *(initial) buffering* or *prebuffering*. We refer to the specific amount of buffered data in seconds of video as *buffering size*. The practical value of the buffering size depends on the network states (e.g., available throughput and network delay), the video parameters (e.g., available bitrates and segment duration), and QoE requirements (with respect to e.g., playback stalls and bitrate). Although a larger buffering size directly incurs a higher startup delay (and a higher latency), it also provides the client with a higher robustness to short-term throughput variations, and thus with a lower risk of buffer underflows. Hence, the consideration of the minimum achievable startup delay must account for the trade-off between the buffering size and the risk of rebuffering. Staelens *et al.* [127] find that IPTV and VoD users are willing to tolerate higher startup delays with a reward of less rebuffering. On the other hand, a moderate startup delay of 2 seconds might severely deteriorate the QoE and even make users quit the view completely, especially when users frequently start a new streaming session, e.g., by switching program channels or watching short videos [56]. In contrast to rebuffering, the client is typically unaware of the network conditions during the initial buffering. In order to quickly fill up the playback buffer and start the playback, a client will often choose the first (few) video segment(s) with the lowest bitrate.

It is worth noting, that the impact of the startup delay on the QoE can be characterized in different ways. The study [117] finds that the QoE will linearly decrease with the increase of the startup delay but will converge to a minimum. Rodríguez *et al.* [128] model the relationship between the QoE and the startup delay using an exponential decaying function.

2.3.2.4 *Latency*

An important factor to evaluate live streaming experiences is the latency, also referred to as *live latency*, which is the time difference between the instants when the live event occurs and when it is played back to users. This time difference is sometimes called end-to-end delay, and expresses the liveness of the video stream. While presently, Internet-based live streaming services might exhibit a latency on the order of tens of seconds, many services would greatly benefits from bringing down this value. In particular, low latency services — e.g., surveillance and passive participation in video conferences — require a smaller latency in the range of a few seconds; in the context of interactive services such as active participation in video conferences, augmented vision, and online gaming, the latency requirements can even reach below one second.

In the case of VoD streaming, since all video contents are already available for delivery before the beginning of a streaming session, VoD services do not have the same latency requirement as live services. Nevertheless, they still have requirements on the startup delay, which is directly affected by the buffering size. The maximum of the buffering size is equal to the client buffer size. Additionally, the buffer size of a VoD client is often bounded (by a threshold of typically several tens of seconds), as service providers only allow the client prefetch a certain number of video segments in view of resource waste and the video quality. This is due to the fact that when the client stores excessive video in the buffer, a user may start a new streaming session or quit the view, or a client may re-download some segments with a higher bitrate in order to offer a better image quality if the throughput is high enough. Therefore, we use the buffer size of a VoD client to represent its latency requirement of a VoD service, and refer to it as *VoD latency*, in order to distinguish between the live and VoD with respect to the latency.

In practice, the buffer size is an indicator of the latency not only in VoD streaming, but also in live streaming, because the latency in live streaming limits the maximum occupancy of the playback buffer. Since the buffer occupancy is upper-bounded by the user tolerable video playback lag, the buffer size is used to represent the streaming latency in the rest of this thesis. Ideally, both the startup delay and the buffer size should be as small as possible to reach the low-latency requirement, yet large enough to avoid buffer underflow events.

2.3.2.5 *Maximizing the QoE*

The ultimate goal of ABS is to deliver a superior QoE to users. To this end, the crucial challenge is to balance many potentially conflicting QoE considerations:

(a) Minimize rebuffering events where the client has to stall the playback due to an empty buffer.

(b) Maximize the image quality with as high a video bitrate as possible subjective to varying network conditions.

(c) Minimize the variations of the image quality by avoiding frequent and/or large bitrate switches.

(d) Minimize the startup delay so that users do not abandon the view while waiting for a playback start.

The conflicts are for instance as follows. Accomplishing goal (b) results in a trade-off with goal (a): the duration and the frequency of rebuffering events may be minimized by always choosing the lowest bitrate, which will yield a suboptimal image quality if the available network throughput can accommodate a video stream with a higher bitrate; while maximizing the image quality by always choosing the highest bitrate will too frequently incur unnecessary rebuffering events. Pursuing goal (c) is traded off with goal (b), because one may maximize the overall image quality by adapting the bitrates to any small changes of the network throughput, which obviously increases the number of bitrate switches, though. In practice, goal (a) is achieved at the cost of goal (c). Last, goal (d) and goal (b) contradict each other: by choosing the lowest bitrate for video segments during the startup, one succeeds in minimizing the startup delay, however, with a loss of the image quality.

Note that, because this thesis focuses on low-latency streaming, we do not subsume the latency in the performance metrics for our QoE evaluation, instead we interpret it as a constraint an ABS system must fulfill, in order to evaluate the ABS performance under a specific latency requirement for various application scenarios.

3

TOWARDS REDUCED LATENCY IN ADAPTIVE STREAMING

To cope with *low-latency* adaptive streaming and to push the latency to its limit, it is necessary to determine a reasonable lower bound on the latency for adaptive streaming. Few researchers address the achievable lower bound of the latency. The contributions in this chapter fill this gap by deriving the lower bound for the *buffering size*, which is the dominant factor of the streaming latency. Specifically, we first analyze the components affecting the latency and identify the key component of the latency. Then, we develop an analytical model for the process of adaptive streaming with respect to client buffer dynamics. Next, we present the required buffering size and its theoretical minimum given characteristics of a streaming scenario. Last, we introduce an approximation of the minimum buffering size and validate the approximation under a bitrate-adaptive streaming simulation. The originality of the contributions is acknowledged in an earlier conference publication [35].

3.1 LATENCY IN ADAPTIVE STREAMING

One fundamental characteristic of video streaming is the latency. It is also a key performance indicator for bitrate adaptation. With respect to content availability, we distinguish the latency with two categories: live streaming and VoD streaming.

3.1.1 *Live Latency*

In a live streaming system, the video content is continuously captured, encoded, and segmented while streaming. Once a video segment is available, the system can start to deliver it to the client. After the complete reception of a segment, the client decodes and renders it. In order to achieve a better responsiveness to the bitrate mismatch between network throughput and video quality, the client may buffer a specific amount of segments before the playback. In an adaptive streaming system, the video content is further encoded in several quality levels with respect to their video characteristics (e.g., resolutions, frame rates, and bitrates). Accordingly, the system dynamically adapts the characteristics of the video stream to varying network states, leading to a smoother viewing experience with less playback stalls and higher video bitrates.

In Section 2.3.2, the *latency* of live streaming is defined as the time difference between the time at which the content is recorded and the time at which it is displayed on a client. The time difference is often termed *live latency*. This latency is mainly affected by the following components: the video codec (for encoding and decoding), the delivery (traversal over the network layers at both ends) and the buf-

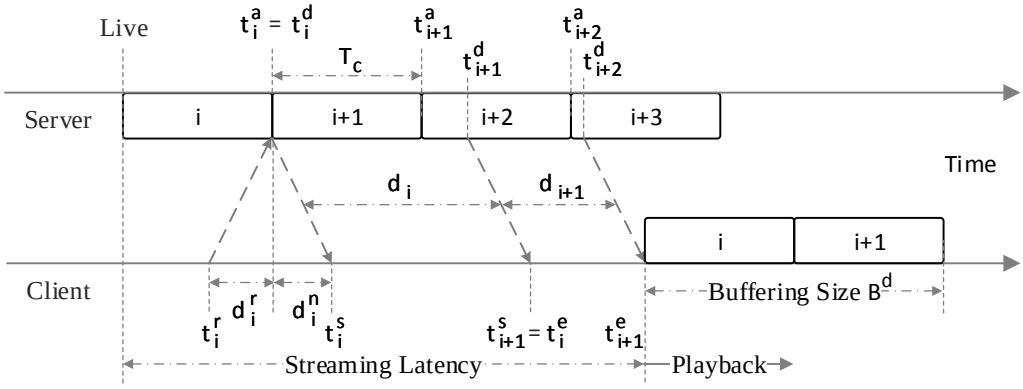


Figure 3.1: An example of Live latency with a buffering size of two segments.

fering. We assume that the video encoder will be parameterized appropriately for a given scenario (e.g., intra-frame or slice-based encoding for very low latencies) to allow a sufficient remainder of the overall delay budget for the delivery and the buffering. The delivery is considered throughout the thesis by including the network delays — typically characterized by the Round Trip Time (RTT) — visible to the application layer, and the equations are general so that they can be applied to different applications scenarios (e.g., VoD with acceptable latencies of several seconds, broadcast-like scenarios with latencies of several 100 *ms* to a second, and interactive scenarios with latencies below 100 *ms*). Therefore, in this thesis, we focus on the latency affected by the delivery and the buffering. The *delivery time* expresses the time required for the delivery of video data (e.g., a frame or a segment) over the network. It consists of the reception duration for a segment and the network delay along the downstream path. A client may need to store a specific amount of video data into its playback buffer before starting a video playback. We refer to the time of the client collecting these data as the *buffering time*, and the amount of these video data in seconds as the *buffering size*. The process of buffering these data is known as the *buffering phase*. Once the process of the buffering is finished, the playback of the video starts and continues till the end of the streaming session. We call this phase the *playback phase*.

Figure 3.1 illustrates an example of the latency in live streaming and the time notations used as follows. We assume that the delivery of segments are performed sequentially. Consider segment $i \in \mathbb{N}^+$ is available at time t_i^a. The client issues a request for the segment at time t_i^r and the delivery time for the request is d_i^r. Then the server starts to deliver the segment at time t_i^d. The network delay of segment i is denoted by d_i^n. We assume that d_i^n is constant during the delivery of segment i; i.e., the network delay for sending each bit of segment i is constant. At time t_i^s the client starts to receive the segment and finishes the reception at time t_i^e. Thus, the reception duration for segment i can be expressed as $d_i = t_i^e - t_i^s$. Under the assumption of the sequential delivery, we have

$$t_i^d = \max\left\{t_i^a,\, t_i^r + d_i^r,\, t_{i-1}^d + d_{i-1}\right\}\,, \tag{3.1}$$

where $i > 1$ and $t_1^d = \max\{t_1^a,\, t_1^r + d_1^r\}$. Note that $t_{i-1}^d + d_{i-1}$ in Equation 3.1 indicates the time at which the server finishes sending segment $i-1$. We denote the

buffering size by B^d and the segment duration by T_c. If a client buffer contains $m \geq 1$ segments (i.e., $B^d = m \cdot T_c$) when a playback starts, the latency can be calculated as:

$$L = t^e_{i+m-1} - t^a_i + T_c = t^s_{i+m-1} + d_{i+m-1} - t^a_i + T_c \quad , \tag{3.2}$$

where $i \geqslant 1$, $\forall j \geq 1 : t^s_j = t^d_j + d^n_j$, $\forall j > 1 : t^a_j = t^a_{j-1} + T_c = t_0 + j \cdot T_c$, $t^a_1 = t_0 + T_c$, and t_0 is the start time of the content.

According to Equation 3.2, we can derive the following three special cases, in order to provide an intuitive understanding of main contributors to the latency. The special cases are based on the following assumptions:

- A segment request that comes too early results in biased bitrate selections and therefore a suboptimal streaming performance, while a request that comes too late leads to an increase of streaming latency and may cause an unnecessary drop of the client buffer level. Consequently, we assume that the system employs an ideal pre-request scheme of segments such that the start of delivering a segment is performed once the delivery of the preceding segment is completed and the selected segment is available, i.e., $\forall j \geq 1 : t^r_j + d^r_j = \max\left\{t^d_{j-1} + d_{j-1}, t^a_j\right\}$. In practice, it is difficult to have an ideal pre-request scheme due to a lack of precise estimation of the delivery time (i.e., d^r_j, d^n_j, and d_j). To this effect, we introduce a server-side adaptation (Chapter 5) to fulfill this assumption.

- In order to eliminate the additional delay incurred prior to the delivery of segment i, we also assume that the server starts to deliver segment i as soon as it is available, i.e., $t^d_i = t^a_i$.

- Furthermore, we assume that the size of the client buffer is unlimited so that the reception will not delay due to a full buffer. This implies that $t^s_i = t^d_i + d^n_i$.

The example shown in Figure 3.1 satisfies the assumptions.

i.) If the network throughput is sufficiently high such that $\forall i \leq j \leq i + m - 1 : d_j < T_c$, the server starts the delivery for segment j at the time $t^d_j = t^a_j$. We can calculate the latency L as

$$\begin{aligned} L =& t^a_{i+m-1} + d^n_{i+m-1} + d_{i+m-1} - t^a_i + T_c \\ =& t^a_i + (m-1) \cdot T_c + d^n_{i+m-1} + d_{i+m-1} - t^a_i + T_c \\ =& m \cdot T_c + d^n_{i+m-1} + d_{i+m-1} \\ =& B^d + d^n_{i+m-1} + d_{i+m-1} \end{aligned} \quad . \tag{3.3}$$

This case yields the lower bound of the latency with a buffering size of B^d.

ii.) If the network throughput is so low that $\forall i \leq j \leq i + m - 1 : d_j > T_c$, the time at which the server finishes the delivery of segment $j - 1$ is after the available time of segment j (i.e., $t^d_j = t^d_{j-1} + d_{j-1} > t^a_j$). It implies that the start time of the

reception for segment j is $t_j^s = t_{j-1}^d + d_{j-1} + d_j^n$. Based on Equation 3.2, the latency can be computed as

$$\begin{aligned} L =& t_i^d + d_{i+m-1}^n + \sum_{j=0}^{m-1} d_{i+j} - t_i^a + T_c \\ >& t_i^d - t_i^a + d_{i+m-1}^n + (m+1) \cdot T_c \\ =& B^d + d_{i+m-1}^n + T_c \end{aligned} . \tag{3.4}$$

iii.) In adaptive streaming, adaptation algorithms strive to select segments whose video bitrates are close to the network throughput on average. Without loss of generality, we assume that $\forall i \leq j \leq i+m-1 : d_j = T_c$. So it holds $t_j^d = t_{j-1}^d + d_{j-1} = t_j^a$. Similarly to the previous case, we get

$$\begin{aligned} L =& t_i^d + d_{i+m-1}^n + \sum_{j=0}^{m-1} d_{i+j} - t_i^a + T_c \\ =& B^d + d_{i+m-1}^n + T_c \end{aligned} . \tag{3.5}$$

Based on the analysis of above cases, we see that the buffering size is the biggest contributor to the latency which can be optimized under a given network scenario. It implies that given network characteristics (e.g., throughput and network delay), we can optimize m, T_c, and video bitrates (for having small d_i), in order to minimize the streaming latency. Normally, T_c and a selected set of video bitrates are provided in a streaming system for preserving a high coding efficiency and video quality. Therefore, the minimization of m (the buffering size) is the challenge in low-latency adaptive streaming.

3.1.2 *Latency in VoD streaming*

In a VoD streaming system, all video contents or segments are prerecorded and stored at the server, and thus, there is no such a latency requirement as in live streaming. Theoretically, VoD streaming allows buffered video data to be as long as the duration of video content. Though, as discussed in Section 2.3.2, VoD streaming still has requirements on the startup delay, the video quality, and waste of network and server resources, especially in cases, when users watch long videos, frequently switch channels, or early abandon the view. Therefore, the amount of buffered data is typically bounded in a VoD streaming client.

As the complete video content in a VoD streaming session is available for the delivery right from the beginning of the session, in contrast to a live streaming system, we define the *latency* of VoD streaming as the maximum allowed amount of buffered data in seconds of video at the client, also known as *client buffer size* (in seconds). To distinguish from live latency, we also refer to it as *VoD latency*. The reasons for the definition of VoD lantency as client buffer size are two-fold: the client buffer size implies the maximum allowed delay between the arrival time and the display time of video data at the client; it is the upper bound of the buffering size, which affects the startup delay. Therefore, the buffering size is the dominant factor of VoD latency, as the lower bound of the latency. Furthermore, the buffering size dominates the *startup delay*, which is defined as the time difference between the

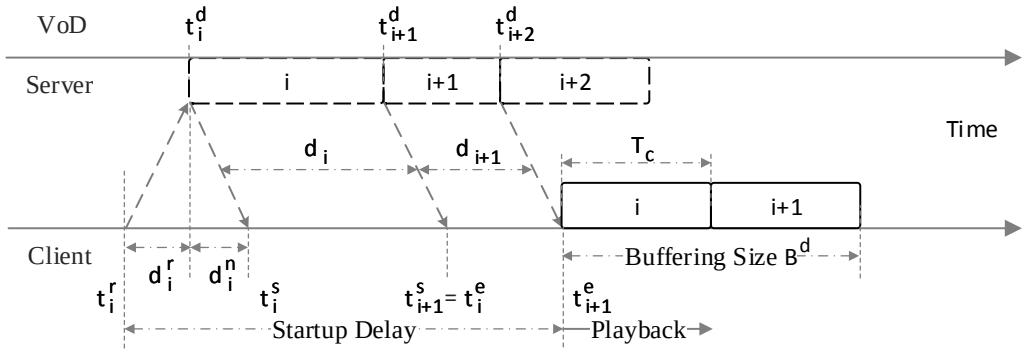


Figure 3.2: An example of VoD streaming with a buffering size of two segments.

time when the content is requested and the time when it is displayed on a client. Similar to live streaming, we derive and analyze the startup delay of VoD streaming as follows. Because all video segments are available for the delivery during the whole time throughout the streaming session, the available time of each segment is prior to its request time; i.e., $\forall i \geq 1 : t_i^a \leq t_i^r$. According to Equation 3.1, the server starts to deliver segment i at time

$$t_i^d = \max\left\{t_i^r + d_i^r,\, t_{i-1}^d + d_{i-1}\right\} \ , \tag{3.6}$$

where $i > 1$ and $t_1^d = t_1^r + d_1^r$. Analog to Equation 3.2, if the buffer of a VoD client contains $m \geq 1$ segments (i.e., $B^d = m \cdot T_c$) when a playback starts, the startup delay T_s by definition can be given as:

$$T_s = t_{i+m-1}^e - t_i^r = t_{i+m-1}^s + d_{i+m-1} - t_i^r \ , \tag{3.7}$$

where $i \geqslant 1$ and $\forall j \geq 1 : t_j^s = t_j^d + d_j^n$.

Similarly, we assume that an ideal pre-request scheme is employed such that $\forall j \geq 1 : t_j^r + d_j^r = t_{j-1}^d + d_{j-1}$. Then Equation 3.7 can be expressed as

$$\begin{aligned} T_s =& t_{i+m-1}^s + d_{i+m-1} - t_i^r \\ =& t_i^r + d_i^r + d_{i+m-1}^n + \sum_{j=0}^{m-1} d_{i+j} - t_i^r \\ =& d_i^r + d_{i+m-1}^n + \sum_{j=0}^{m-1} d_{i+j} \end{aligned} \ . \tag{3.8}$$

An example is illustrated in Figure 3.2. The reception duration d_i is proportional to the segment duration T_c dependent to the average network throughput during the reception of segment i; i.e., $\forall i \geq 1 : d_i \propto T_c$. Therefore, it holds $\sum_{j=0}^{m-1} d_{i+j} \propto m \cdot T_c = B^d$. In the case of adaptive streaming, a well-designed adaptation algorithm will select segments with video bitrates equivalent to the average throughput, such

that the video quality is optimized. This results in $\sum_{j=0}^{m-1} d_{i+j} \sim B^d$ and based on Equation 3.7 we have

$$\begin{aligned} T_s =& d_i^r + d_{i+m-1}^n + \sum_{j=0}^{m-1} d_{i+j} \\ \sim& d_i^r + d_{i+m-1}^n + B^d \end{aligned} . \tag{3.9}$$

Because the amount of buffered data is bounded in VoD streaming, B^d is bounded and its upper bound is the client buffer size β_{max} (in seconds) i.e., the VoD latency.

Based on Equation 3.9, we see that the startup delay of VoD streaming is dominated by the buffering size as in the case of live streaming. Moreover, the VoD latency (i.e., the client buffer size) is bounded below by the buffering size; namely, $L = \beta_{max} \geq B^d$. Therefore, if the buffering size is minimized, the client buffer size can also be minimized so that the resource usage and the video quality is optimized. To minimize the buffering size, we need to find out what the lower bound of the buffering size is. In the following we introduce a model of adaptive streaming for buffer size before the derivation of the minimum buffering size.

3.2 ADAPTIVE STREAMING MODEL FOR BUFFER SIZE

Consider a video as a set of N consecutive segments $\mathcal{V} = \{1, 2, 3, \cdots, N\}$, each of which represents T_c seconds of the video and is encoded at different nominal bitrates in set $\mathcal{R}$. A streaming client receives the video segments from index $i = 1$ and stores them into a playback buffer. Let $b(t)$ be the buffer level (in seconds) at time t. The client starts the video playback, once it completes the buffering i.e., the reception of the first $m \geq 1$ segments.

Let t_i^s and t_i^e be the time at which the client starts and finishes the reception of segment $i \geq 1$, respectively. If the size (in *kbit*) of segment i encoded at the nominal bitrate R is given and denoted by S_i^R, or if the average bitrate of segment i is given and denoted by r_i, we have

$$S_i^R = \int_{t_i^s}^{t_i^e} C(t)\, dt \quad \text{or} \quad r_i \cdot T_c = \int_{t_i^s}^{t_i^e} C(t)\, dt \,, \tag{3.10}$$

where $C(t)$ is the available network throughput at time t. As described in Section 3.1, the reception duration of segment i is defined as

$$d_i = t_i^e - t_i^s \,. \tag{3.11}$$

After the entire reception of segment i, the client may wait for Δt_{i+1} seconds and then starts to receive segment $i + 1$ at time t_{i+1}^s. The waiting time Δt_{i+1} is referred as the reception delay in this thesis and is determined by the time at which the data sent by the server arrive and the time at which the client allows to receive the data. The former one is dependent on the available time and the requesting time of segments as well as the network delay. The latter one considers the waiting time incurred by the client when the buffer is full. Note that at that moment, the client cannot receive any new segments and waits for the buffer to reduce a level which

allows a new segment to be received. Then, the start time of receiving segment $i+1$ can be given by

$$t^s_{i+1} = \max\left\{t^d_{i+1} + d^n_{i+1},\, t^e_i + \Delta t^w_{i+1}\right\}\,, \tag{3.12}$$

where Δt^w_{i+1} is the waiting time for receiving segment $i+1$ due to a full buffer after the reception of segment i. According to Equation 3.1, the delivery of segment $i+1$ starts at time

$$t^d_{i+1} = \max\left\{t^a_{i+1},\, t^r_{i+1} + d^r_{i+1},\, t^d_i + d_i\right\}\,. \tag{3.13}$$

Therefore the reception delay between segment i and segment $i+1$ (i.e., the time that elapses from the moment the client finishes receiving segment i until the client starts to receive segment $i+1$) Δt_{i+1} can be computed as

$$\Delta t_{i+1} = t^s_{i+1} - t^e_i\,. \tag{3.14}$$

As the segments are being filled into the buffer during the receiving process and being drained out for video playback, the buffer dynamics are expressed as follows:

$$b^s_{i+1} = \begin{cases} b^s_i + T_c & i \le m \\ \left(\left(b^s_i - d_i\right)_+ + T_c - \Delta t_{i+1}\right)_+ & else \end{cases}\,, \tag{3.15}$$

where $(x)_+ = \max\{x, 0\}$, $m \ge 1$ is the number of segments required for the buffering before the first playback (i.e., $B^d = m \cdot T_c$), $b^s_i = b(t^s_i)$ indicates the buffer level when the client starts to receive segment i, and the buffer level is initialized with $\forall t \le t^s_1 : b(t) = 0$, thereby $b^s_1 = 0$. We distinguish two cases in Equation 3.15: during the buffering phase, only the filling process of the buffer is being performed without the draining process, because the playback has not started yet; during the playback phase, both processes are being performed at the same time. Equation 3.15 assumes that each segment must be received in its entirety before it can be played back. Note that playback stall events occur if $b^s_i < d_i$ or $b^s_i + T_c < d_i + \Delta t_{i+1}$. Namely, an event of playback stalls occurs if the buffered data is not sufficient to be consumed before the entire reception of a segment, or if the buffered data succeeding the reception cannot hold to be consumed prior to the next reception. Analog to Equation 3.15, the waiting time Δt^w_{i+1} can be formulated as

$$\Delta t^w_{i+1} = \begin{cases} 0 & i \le m \\ \left(\left(b^s_i - d_i\right)_+ + T_c - \beta_{max}\right)_+ & else \end{cases}\,, \tag{3.16}$$

where β_{max} denotes the maximum occupancy level of the client buffer, i.e., the client buffer size. The client buffer size is assumed to be equal to or larger than the buffering size, i.e., $\beta_{max} \ge B^d = m \cdot T_c$, such that the buffering process can be completed. Here, for simplicity, we assume that B^d and β_{max} are a multiple of segment duration T_c.

Note that our seven equations (Equation 3.10–3.16) are indeed linear independent and the equation system can be solved, given network throughput and delay (i.e., $C(t)$, d^r_{i+1}, and d^n_{i+1}), available time and request time of segments (i.e., t^a_{i+1} and t^r_{i+1}), selected video bitrates r_i, segment duration T_c, maximum client buffer size β_{max},

buffering size m, as well as the status of the preceding segment (i.e., b_i^s, t_i^s, and t_i^d). An example of solving the system is shown in Excursion 3.1. The dynamics of the model are therefore deterministic.

Excursion 3.1 An example of solving the equation system of adaptive streaming model for buffer size

Consider $\beta_{max} = 2\,s$, $i > m$, $b_i^s = 2\,s$, $C(t) = 1000\,kbps$, $d_{i+1}^n = d_{i+1}^r = d_i^n = 1\,s$, $r_i = 500\,kbps$, $T_c = 2\,s$, $t_i^s = 10\,s$, $t_i^a = t_i^d = 9\,s$, $t_{i+1}^a = 11\,s$, $t_{i+1}^r = 11\,s$, and that we want to determine b_{i+1}^s. Based on Equation 3.10, we have $t_i^e = 11\,s$. Then we get $d_i = 1\,s$ with Equation 3.11. We further retrieve $\Delta t_{i+1}^w = 1\,s$ from Equation 3.16. By Equation 3.13, it yields $t_{i+1}^d = 12\,s$. Subsequently, we obtain $t_{i+1}^s = 13\,s$ with Equation 3.12. Equation 3.14 gives us $\Delta t_{i+1} = 2\,s$. According to Equation 3.15, we determine $b_{i+1}^s = 1\,s$; i.e., the buffer level becomes smaller.

If we employ an ideal pre-request scheme for segment $i+1$ with $t_{i+1}^r = 10\,s$, we can achieve $t_{i+1}^d = 11\,s$ and $b_{i+1}^s = 2\,s$, which introduces no additional delay for the delivery and is beneficial to the stability of buffer dynamics. This demonstrates the effect of the ideal pre-request scheme and confirms our motivation of presenting the server-side adaptation.

In summary, this example shows that our seven equations (Equation 3.10–3.16) can model the dynamics of streaming process.

In contrast to the model defined in [32], our model first introduces the available time and the request-response time for live video segments into the reception delay. The reasons are as follows. i.) With live streaming, video segments become available for receiving during the course of the streaming session. The client needs to determine precisely when a new segment is ready and make the necessary request. ii.) In the context of low-latency streaming, especially over wireless networks, the request-response time of receiving segments is not negligible for client side adaptation. This is due to a lack of precise estimation of the delivery time as indicated in Section 3.1. Second, our model takes into consideration the buffering size, which characterizes the streaming latency as a dominant factor. Consequently, our model is more generalized, can accommodate wide range of streaming scenarios, especially in the context of low latency.

3.3 MINIMUM BUFFERING SIZE

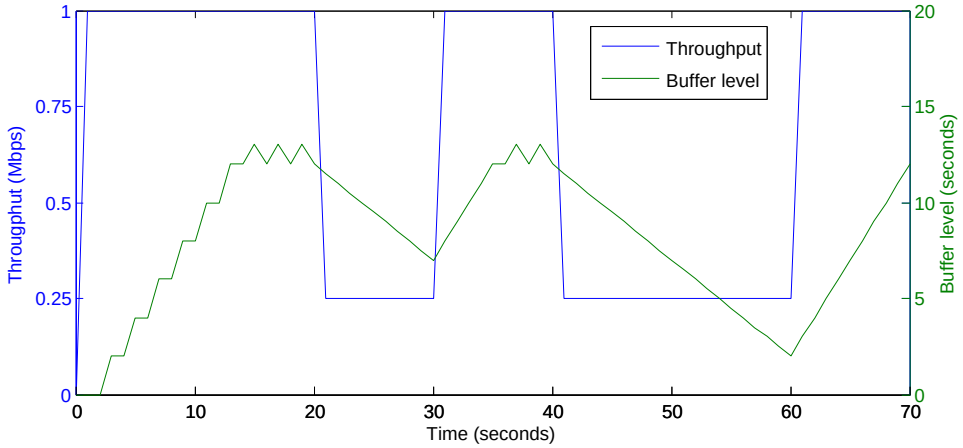
In the following we will address the question: what is a reasonable buffering size for adaptive streaming if the QoS of a network is given. Note that minimizing the buffering size implies minimizing the latency, as described in Section 3.1. To answer this question, we first introduce the concept of the network degradation. Most Internet Service Providers (ISPs) ensure the lowest available throughput (also known as *Guaranteed Throughput*) for their customers. However, they cannot ensure such network QoS anytime, and so the throughput may drop below the guaranteed throughput in a time period, e.g., due to the failure of hardware or quality degradation of network path (e.g., wireless path). Here, we define an event as *network*

degradation in which the average network throughput over each segment duration is lower than the guaranteed throughput.

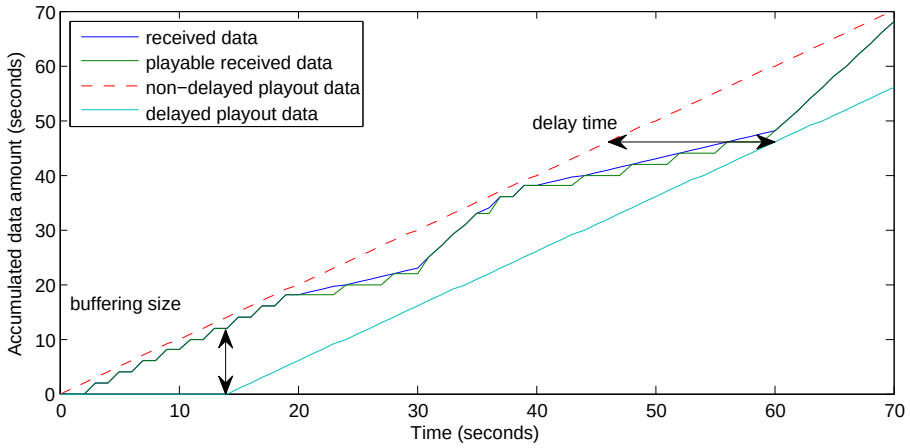
Given a set of video bitrates $\mathcal{R}$, the guaranteed throughput of the network is at least $\min\{\mathcal{R}\}$, such that it can support the minimum performance of video streaming service. Consider a streaming service that experiences a network degradation with a duration D^{deg}. During this time period, the service cannot offer a proper video bitrate to maintain the amount of buffered data over a specific level, because the throughput is smaller than $\min\{\mathcal{R}\}$. It means that $d_i > T_c$ holds for each received segment within D^{deg} and then the buffer level decreases constantly during the playback. If the buffer is empty, the client suffers from playback stalls. Therefore, a client needs to collect sufficient video data in the buffer before a network degradation occurs. Namely, there exists a minimum buffer level a client has when the network degradation starts, such that the client can avoid playback stalls during a network degradation with a duration of D^{deg}. Let B^{deg}_{min} denote this minimum buffer level (in seconds). Then, such a network with D^{deg} requires streaming services to buffer video data of at least B^{deg}_{min} prior to the network degradation. A streaming session may experience one or more events of network degradation at different time intervals. Thus, streaming services need to ensure sufficient video data in the buffer prior to the playback in order to offer continuous playback even when one or multiple degradation events occur. We denote the minimum buffer level prior to the playback of a streaming session by B_{min}. The buffer level required prior to the playback is equivalent to the buffering size per definition introduced in Section 3.1.

The minimum buffer level is the minimum requirement of streaming services with which no playback stalls occur in the course of streaming. Hence, we can derive its theoretical minimum for a streaming session by delaying the playback such that the accumulated playable data received in the buffer is always more than the accumulated playout data at any time points. Figure 3.3 illustrates the idea with an example of live streaming. Note that due to the assumption the data of a segment in the buffer is able to be played only after the segment is totally received by the client. We define *playable received data* as the data of all segments which are received in their entirety and stored in the buffer; while received data includes the playable received data and the data of a segment being received.

Consider the duration of video segments $T_c = 2\,s$ and the minimum of video bitrates $\min\{\mathcal{R}\} = 0.5\,Mbps$. To achieve the minimum buffer level, the selected video bitrate need to be the minimum bitrate, i.e., $r = \min\{\mathcal{R}\}$. The instant throughput during the streaming session is shown in Figure 3.3a and switches between $0.25\,Mbps$ and $1\,Mbps$. To simplify the illustration, we assume that the reception delay is zero and the client buffer size is infinite. Therefore, given the throughput and the selected video bitrates, we generate the curves of accumulated received data and accumulated playable received data in the client buffer as shown in Figure 3.3b. Note that no playback stalls are allowed during the streaming even if the client experiences the network degradation. As a result, the amount of accumulated playout data continuously and linearly increases with a slope of 1, because one second of video is played back in one second of real time. Moreover, the line of accumulated playout data needs to be below the curve of playable received data, because the client requires more received data than the data to be played back. We can determine the delay time for the playback by shifting the line to the right as



(a) Throughput and buffer level



(b) The accumulated amount of received data and playout data

Figure 3.3: An example of solving the minimum buffering size for a live streaming session. The idea is to delay the playback, such that the accumulated amount of playable received data is more than the one of playout data at any time points. With a delayed playback, the buffer level varies without reaching zero, even the throughput drops below the minimum supported bitrate.

shown in Figure 3.3b The buffering size is then the amount of received data at the time when the delayed playback starts. In this example, the delay time is $14\,s$ and the buffering size is $12\,s$.

Formally, we can calculate the accumulated amount of playable received data (in seconds of video) at time $t \geq 0$ as

$$\Phi'(t) = (i-1) \cdot T_c \mid t^e_{i-1} \leq t < t^e_i \quad , \tag{3.17}$$

where $i \in \mathbb{N}^+$ fulfills $t^e_{i-1} \leq t < t^e_i$, t^e_i denotes the time at which the reception of segment i is completed, and $t^e_0 = 0$. Equation 3.17 implies that the amount of playable received data increases by T_c seconds once a segment is completely received, and keeps constant before the reception of the next segment is finished. In contrast, the accumulated amount of received data (in seconds of video) in the buffer at time t can be calculated as

$$\Phi(t) = \begin{cases} (i-1)\cdot T_c & t < t_i^s \\ (i-1)\cdot T_c + \frac{1}{r_i}\int_{t_i^s}^{t} C(t)\,dt & else \end{cases} \Bigg| \; t_{i-1}^e \leq t < t_i^e \quad , \tag{3.18}$$

where t_i^s is the start time of the reception for segment i and fulfills $t_{i-1}^e \leq t_i^s < t_i^e$ because of the sequential reception. The additional term of the integral in Equation 3.18 represents the portion of segment i (in seconds) received by the client and stored in the buffer. Note that the data amount in the buffer actually increases continuously during the reception of segments in the case of $C(t) > 0$; while the amount of playable data increases in a step-wise fashion, due to the assumption that each segment can be played back only if it is received in its entirety.

t_i^s and t_i^e in Equation 3.17 and Equation 3.18 can be determined based on Equations 3.10–3.13 in Section 3.2. Here, we consider a live scenario and assume that the size of the playback buffer is infinite. The reasons are as follows. i.) To determine the minimum buffer level, we need to relax the size limit of the buffer. Otherwise, the minimum is not solvable in the case where the minimum is larger than the size limit. ii.) When the buffer is full, the client cannot receive any new segments and waits for the buffer to reduce to a level which allows a new segment to be received. The waiting time increases the streaming latency. In order to minimize the latency, we eliminate the size limit of the buffer. iii.) The memory and the storage are less important for modern clients compared to the live experience. The only exception is when playback stall events occur, the buffer level will cumulatively increase after the reception duration becomes $d_i < T_c$, because the client has to wait for sufficient data during playback stalls and this delays the playback and therefore increases the latency. Skipping video frames or segments can be used to limit the buffer level to a specific value, so as to fulfill the latency constraint.

In practice, the assumption of infinite buffer sizes in live streaming is feasible and does not overload the client with respect to the performance, because the restricted availability of video segments implicitly limits the number of segments stored in the buffer if the latency constraint is defined. In VoD scenarios, that all segments are available for the delivery, can lead to significant quality gains in times of very high available throughput. Moreover, a buffer which can physically hold specific long video with the highest bitrate in bytes, can accommodate much longer video with the lowest bitrate. Streaming systems may select video segments with the lowest bitrate to maintain the continuity of the playback in the case of critical network conditions (such as network degradation).

Hence, we can reformulate Equation 3.12 and have the start time of receiving segment $i+1$ with

$$t_{i+1}^s = t_{i+1}^d + d_{i+1}^n \; . \tag{3.19}$$

Together with Equation 3.10, Equation 3.11, and Equation 3.13, we can calculate t_i^e, the end time of the reception for segment i. To obtain the theoretical minimum buffer level, we need to select segments with the minimum video bitrate and minimize the reception delay Δt_{i+1}. Therefore, we assume that $r_i = \min\{\mathcal{R}\}$ and that an ideal pre-request scheme of segments as described in Section 3.1 is applied so that $t_{i+1}^r + d_{i+1}^r = \max\left\{t_i^d + d_i,\, t_{i+1}^a\right\}$. Then, Equation 3.13 can be reformulated as

$$t_{i+1}^d = \max\left\{t_{i+1}^a,\, t_i^d + d_i\right\} \; . \tag{3.20}$$

Note that, under the constraint $t^e_{i-1} \leq t < t^e_i$, i is unique and deterministic for each t, i.e., i is a function of t. The is due to the assumption that the reception and the delivery of segments are performed sequentially, and thus, t^e_i is monotonic increasing along i. Moreover, t^e_i is determined based on the aforementioned equations. So $\Phi'(t)$ of Equation 3.17 and $\Phi(t)$ of Equation 3.18 both are deterministic.

Let $\Omega(t)$ denote the accumulated amount of playout data at time t. Because the playout runs without interruptions and t seconds of video are played back in t seconds of real time, we have

$$\Omega(t) = t \,. \tag{3.21}$$

Because $\Omega(t)$ has a slope of 1, the magnitude of right-shifting (i.e., the delay time) is equal to the one of down-shifting (i.e., the maximum difference between $\Omega(t)$ and $\Phi'(t)$). Therefore, the delay time of the playback needs to be at least

$$T_d = \left(\max_{0 \leq t \leq T} \{\Omega(t) - \Phi'(t)\} \right)_+ , \tag{3.22}$$

so that the buffer contains more received data than the data to be played out at any time points during a streaming session with a duration of T. Accordingly, this delay ensures that no playback stalls occur in the course of the streaming. Note that $\Phi'(t)$ is used in Equation 3.22 instead of $\Phi(t)$, because the data of a segment is playable only after the entirety of the segment is available in the buffer.

As mentioned before, the amount of received data increases constantly, in contrast to a step-wise increase of the amount of playable received data. At time $t = T_d$, the client receives $\Phi(T_d)$ video data. It implies that the client buffer level is $\Phi(T_d)$ at time $t = T_d$, if the playback delays by T_d. Hence, the minimum buffer level required prior to the playback is

$$B_{min} = \Phi(T_d) \,, \tag{3.23}$$

so that streaming services avoid playback stalls even if the network degradation occurs. We refer to this minimum buffer level as the *minimum buffering size* of a streaming session. Going back to the question at the beginning of this section, Equation 3.23 gives an answer to the lower bound of the buffering size (B_{min}) with which streaming services offer continuous playback during the entire streaming session even when network degradation events occur.

3.4 AN APPROXIMATION OF MINIMUM BUFFERING SIZE

The determination of the minimum buffering size presented in the previous section requires perfect throughput and delay knowledge of the network during the streaming session. However, this is unrealistic in practice, because the exact future network states are unknown. In this section, we introduce an approximation to solve the minimum buffering size. Our approximation is straightforward. It may therefore serve as the foundation for potential requirements and improvements of streaming systems.

Consider a live streaming scenario. The service experiences an event *deg* of network degradation with a duration of D^{deg}. Let B^{deg}_{min} denote the minimum buffer level required prior to the degradation event, such that the playback can run without stalls within D^{deg}. This minimum buffer level B^{deg}_{min} can be calculated as:

$$B_{min}^{deg} = K \cdot \left(\Delta t + \frac{r}{C^{deg}} T_c\right) - (K-1)_+ \cdot T_c + \left(\max\left\{D_{res}^{deg}, \Delta t\right\} + y - T_c|_{K>0}\right)_+ , \tag{3.24}$$

where K is the number of completely received segments during the time period D^{deg} and is defined as

$$K = \left\lfloor \frac{D^{deg}}{\Delta t + \frac{r}{C^{deg}} T_c} \right\rfloor , \tag{3.25}$$

r is the selected video bitrate, Δt is the reception delay of two consecutive segments, T_c is the segment duration, C^{deg} is the average throughput within D^{deg},

$$D_{res}^{deg} = D^{deg} - K \cdot \left(\Delta t + \frac{r}{C^{deg}} T_c\right) \tag{3.26}$$

denotes the remaining time of D^{deg} after the entire reception of all K segments within D^{deg},

$$y = \frac{r \cdot T_c - C^{deg} \cdot \left(D_{res}^{deg} - \Delta t\right)_+}{C_a^{deg}} \tag{3.27}$$

is the time required to receive the portion of a segment (or the entire segment) after the network degradation, C_a^{deg} is the average throughput succeeding the network degradation within the time period y, and

$$T_c|_{K>0} = \begin{cases} T_c & K > 0 \\ 0 & else \end{cases} . \tag{3.28}$$

Note that the equation still holds if we define $\frac{0}{0} = 0$ in the case of $C^{deg} = 0$. To achieve the extreme, we can select segments with the minimum bitrate (i.e., $r = \min\{\mathcal{R}\}$) and set the reception delay to zero (i.e., $\Delta t = 0$).

It should be noted that Equation 3.24 is derived based on the following assumptions: i.) The reception of a segment is completed at the time when a network degradation starts; ii.) The reception delay of two consecutive segments is constant and equal to Δt; iii.) The average throughput for receiving each segment during the network degradation is constant and equal to C^{deg}; iv.) C_a^{deg} is constant and fulfills $\Delta t + \frac{r \cdot T_c}{C_a^{deg}} \leq T_c$ such that no playback stalls occur succeeding the network degradation. As a result, our equation provides an approximation of streaming requirements with respect to the buffer level, video bitrates, the segment duration, the duration of the network degradation, the average throughput, and the reception delay. The derivation of the equation is in Appendix A.1.

Equation 3.24 gives an approximation of the minimum buffer level required prior to the network degradation, with which streaming services have a chance to offer constant playback during that degradation. Streaming services may experience one or multiple events of network degradation during the entire streaming session. In the case of multiple events, the interval between two sequential events may be insufficiently long and the network throughput within the interval may be insufficiently high, so that the client buffer after one degradation can not be refilled to a

buffer level equal to or above the previous level prior to the degradation before the next degradation occurs. Consequently, the buffer levels prior to those degradation events progressively decrease. We refer to such an interval between the degradation events as *insufficient degradation interval*. To approximate the minimum buffer level in this scenario, we take into consideration the following two cases: i.) the maximum of minimum buffer levels required to compensate the buffer consumption during each single degradation event and ii.) the minimum buffer level required for the progressive buffer reduction within a time period having multiple degradation events with insufficient intervals. We choose the maximum of the required buffer levels in both cases as the minimum buffer level prior to the playback B_{min} (i.e., the minimum buffering size of a streaming session) required for a continuous playback in the course of the entire session.

Formally, we denote the approximate value for the minimum buffering size by $\widetilde{B_{min}}$ and express it as

$$\widetilde{B_{min}} = \max\left\{B_{max}^{sd}, B_{min}^{md}\right\} , \tag{3.29}$$

where B_{max}^{sd} and B_{min}^{md} represent the required buffer level for maintaining constant playback during each single degradation event and during the time period having multiple degradation events with insufficient intervals, respectively. By separately calculating all minimum buffer levels required for all degradation events during the entire session using Equation 3.24, we have

$$B_{max}^{sd} = \max_{deg \in \mathcal{D}}\left\{B_{min}^{deg}\right\} , \tag{3.30}$$

where $\mathcal{D}$ denotes the set of all degradation events and B_{min}^{deg} is the minimum buffer level required prior to a degradation event deg. To calculate B_{min}^{md}, we consider a streaming scenario having sequential degradation events with N^{idi} insufficient intervals. Let D_n^{idi} be the duration of the n-th insufficient interval with $1 \leq n \leq N^{idi}$ and C_n^{idi} the average throughput over the n-th interval. The minimum buffer level required prior to those sequential degradation events can be determined by

$$B_{min}^{md} = \sum_{deg \in \mathcal{D}^{idi}} B_{min}^{deg} - \sum_{n=1}^{N^{idi}} \left(\frac{C_n^{idi} \cdot D_n^{idi}}{r} - D_n^{idi} \right) , \tag{3.31}$$

where $\mathcal{D}^{idi}$ indicates the set of sequential degradation events with N^{idi} insufficient intervals. Note that the number of events in $\mathcal{D}^{idi}$, denoted by $\left|\mathcal{D}^{idi}\right|$, is equal to $N^{idi}+1$. The second sum term of Equation 3.31 represents the amount of the refilled buffer during the insufficient degradation intervals.

In practice, it is inconvenient to retrieve the characteristics of all intervals between network degradation events, in order to determine C_n^{idi} and D_n^{idi} of Equation 3.31. We consider the worst case of D_n^{idi} in our approximation. In detail, we assume that the insufficient degradation interval is the shortest interval between two degradation events within the entire streaming session, and that the sequential degradation events with insufficient intervals are all the events which are connected with the shortest intervals. Furthermore, we assume C_n^{idi} to be the average network

throughput over the entire streaming session. Accordingly, Equation 3.31 can be reformulated as

$$\begin{aligned} B_{min}^{md} &= \sum_{deg \in \mathcal{D}^{sdi}} B_{min}^{deg} - \left(\left|\mathcal{D}^{sdi}\right| - 1\right) \cdot \left(\frac{C \cdot D_{min}^{di}}{r} - D_{min}^{di}\right) \\ &= \left|\mathcal{D}^{sdi}\right| \cdot \overline{B_{min}^{sdi}} - \left(\left|\mathcal{D}^{sdi}\right| - 1\right) \cdot \left(\frac{C \cdot D_{min}^{di}}{r} - D_{min}^{di}\right) \end{aligned}, \tag{3.32}$$

where $\mathcal{D}^{sdi}$ is the set of the degradation events which are connected with the shortest interval between two events in the entire streaming session, $\left|\mathcal{D}^{sdi}\right|$ indicates the number of the events in $\mathcal{D}^{sdi}$, D_{min}^{di} represents the duration of the shortest interval, C is the average throughput over the entire session, and $\overline{B_{min}^{sdi}}$ denotes the average of the minimum buffer levels required for each network degradation event in $\mathcal{D}^{sdi}$.

Finally, substituting B_{max}^{sd} from Equation 3.30 and B_{min}^{md} from Equation 3.32 into Equation 3.29, we can approximate the minimum buffering size for the streaming session with

$$\begin{aligned} \widetilde{B_{min}} = \max \Bigg\{ & \max_{deg \in \mathcal{D}} \left\{ B_{min}^{deg} \right\}, \\ & \left|\mathcal{D}^{sdi}\right| \cdot \overline{B_{min}^{sdi}} - \left(\left|\mathcal{D}^{sdi}\right| - 1\right) \cdot \left(\frac{C \cdot D_{min}^{di}}{r} - D_{min}^{di}\right) \Bigg\} \end{aligned}. \tag{3.33}$$

It should be noted that our approximation assumes no degradation occurred during the buffering phase.

Our approximation (Equation 3.33) implies that given the average throughput of the network, the worst case of one single network degradation, as well as the average case of multiple degradation events with the minimum degradation interval and the number of those degradation events, we can estimate the minimum buffering size required for a streaming scenario. Also, this can provide ISPs with guidelines to ensure their network QoS for low-latency adaptive streaming with respect to the worst case of one network degradation event and the average case of degradation events with the shortest interval.

3.5 VALIDITY OF APPROXIMATIONS

In the following, we validate our approximation of the minimum buffering size under a trace-driven simulation of a mobile network in the context of video streaming.

3.5.1 *Methodology*

We implement a simulation testbed based on the model of adaptive streaming (formulated in Section 3.2) using Matlab[1]. Given throughput and delay traces of the network, the testbed can simulate the receiving and playback process of the video as well as the buffer dynamics at the client for a specific bitrate algorithm and video profiles. To validate the approximation of the minimum buffering size, we first

1 Math. Graphics. Programming. - A tool for analyzing data, developing algorithms, or creating models. http://www.mathworks.com/products/matlab/

solve the theoretical minimum for a streaming session using the method presented in Section 3.3. Then, we calculate the minimum buffering size of a streaming session based on the approximation introduced in Section 3.4. Last, we compare the two values using the error ratio. The error ratio is computed as

$$ER = \frac{\widetilde{B_{min}} - B^*_{min}}{B^*_{min}} \times 100\% \,, \tag{3.34}$$

where $\widetilde{B_{min}}$ and B^*_{min} are the approximate and theoretical minimum of the buffering size for a streaming session without playback stalls, respectively.

Video profiles. We use the *Big Buck Bunny*[2] test sequence with a segment duration of 2 s [68]. The video is encoded with the following nominal bitrates using constant bitrate encoding: $\mathcal{R} = \{500\,kbps, 700\,kbps, 1200\,kbps, 3000\,kbps, 5000\,kbps\}$. This is consistent with the encoding setting for YouTube video and the bitrate range for 240p, 360p, 480p, 720p, and 1080p, respectively[3]. We also use different segment durations of 1 s, 4 s, 6 s, 8 s, and 10 s, to evaluate the validity of the approximation.

Network profiles. Our network profiles for the validation are distinguished into two categories: one single event and multiple events of network degradation. The former one is used to validate the approximate value of the minimum buffer level required prior to one single network degradation (i.e., B^{deg}_{min} of Equation 3.24). The latter one is used to validate the approximation of the minimum buffering size required for a streaming session (i.e., Equation 3.33).

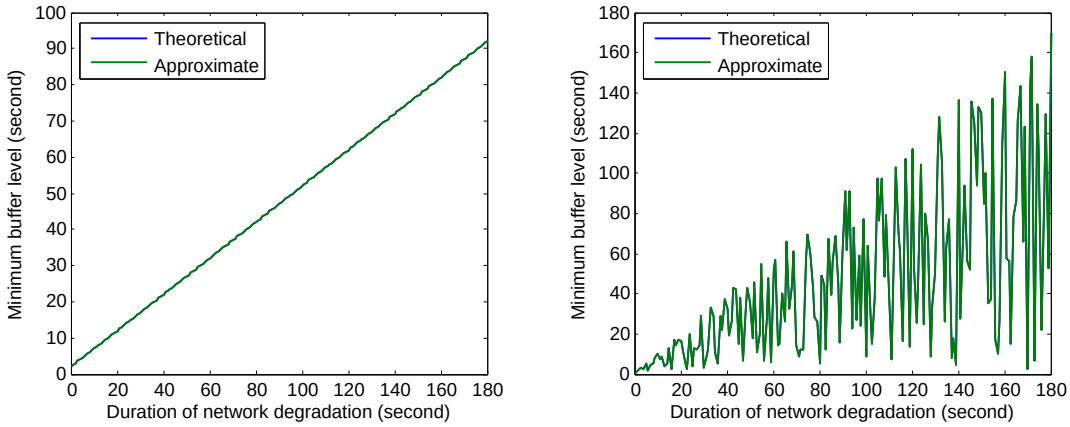
- Single network degradation: Our approximation of B^{deg}_{min} assumes that the average throughputs for receiving a video segment during the degradation and succeeding the degradation (i.e., C^{deg} and C^{deg}_a in Equation 3.24) are constant. So, we first conduct an experiment with constant throughput ($C^{deg} \in (0, 500)$ and $C^{deg}_a \in [500, 5000]$) and validate the approximates with increasing duration of a network degradation event from 0 s to 180 s by 1 s. Then, we conduct another experiment with non-constant throughput, in which the throughputs of each second during the degradation and succeeding the degradation are uniformly randomly chosen from the intervals $(0, 500)$ and $[500, 5000]$, respectively. Similarly, we validate the approximation with increasing duration of network degradation as the first experiment.

- Multiple network degradation: Riiser *et al.* [129] provide a set of video streaming throughput samples measured every one second in a mobile network. The dataset covers different vehicular mobility scenarios (metro, tram, train, bus, car, and ferry) and various measured durations (up to 40 minutes). Since the throughput samples were collected in different types of public transportation, each throughput trace reports significant fluctuations in network conditions when streaming video over mobile network, and then contains multiple network degradation events during a streaming session. We randomly pick 64 throughput traces from the 84 traces, and ensure that the selected traces cover all mobility scenarios and each scenario contains at least five traces[4].

2 https://peach.blender.org/

3 Live encoder settings, bitrates, and resolutions: https://support.google.com/youtube/answer/2853702?hl=en

4 One of six scenarios in the dataset consists of only five traces.



(a) Fixed constant throughput

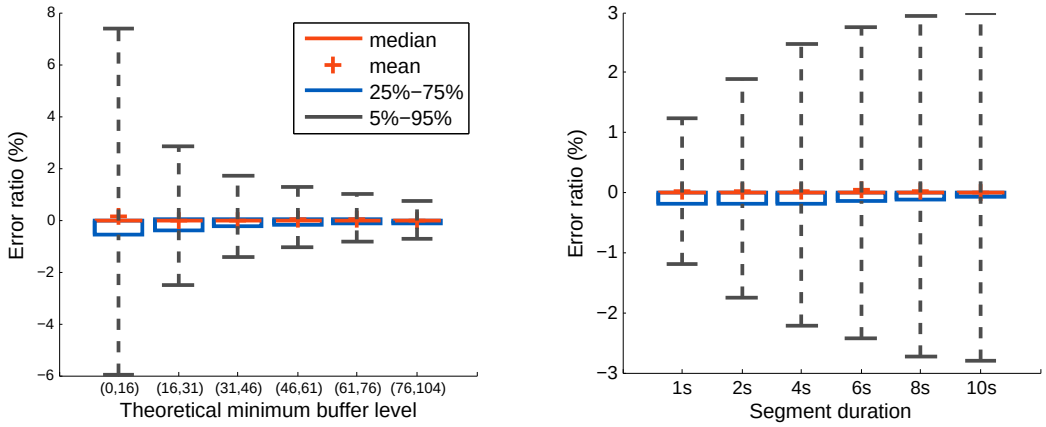
(b) Random constant throughput

Figure 3.4: The theoretical and approximate minimum buffer levels required prior to one single network degradation event with the constant network throughput, as the degradation duration increases. The segment duration is 2 s. a.) The throughputs of each second during the degradation and succeeding the degradation are fixed to 250 $kbps$ and 500 $kbps$, respectively. b.) For each degradation duration, the throughputs of each second during the degradation and succeeding the degradation are fixed to a uniform random value from the intervals $(0, 500)$ and $[500, 5000]$, respectively.

3.5.2 *Experimental Results*

One single network degradation. We first validate the approximation of the minimum buffer level required prior to one single network degradation event (i.e., the approximate values for B_{min}^{deg}). In the first experiment, the throughputs during the degradation and succeeding the degradation are constant and the duration of the degradation increases from 0 s to 180 s by 1 s. Figure 3.4 shows the theoretical and approximate minimum buffer levels as a function of the degradation duration, in which the segment duration is 2 s. In the case of $C^{deg} = 250\,kbps$ and $C_a^{deg} = 500\,kbps$ (Figure 3.4a), the minimum buffer level monotonically increases with increasing duration of the degradation event, and the curve of the theoretical values is in line with that of the approximate values. Figure 3.4b presents the results of scenarios, in which we choose $C^{deg} \in (0, 500)$ and $C_a^{deg} \in [500, 5000]$ in $kbps$ uniformly at random for each degradation duration, respectively. Also, the approximate values coincide with the theoretical ones. Namely, the error ratios are zero in both cases. We omit the results for other segment durations, because the error ratios are zero.

Figure 3.5 shows the error ratios of the approximation when the throughput is not constant. In this experiment, since the throughputs of each second during the degradation are uniformly randomly chosen, we set C^{deg} to the average over all throughputs per second during the degradation. Similar to the previous experiment, we compute the theoretical and approximate minimum buffer levels for different degradation durations from 0 s to 180 s with an increase of 1 s. We repeat the test of each segment duration 100 times and plot the distributions and averages. Figure 3.5a shows the results for the case of 2 s segment duration. The results are



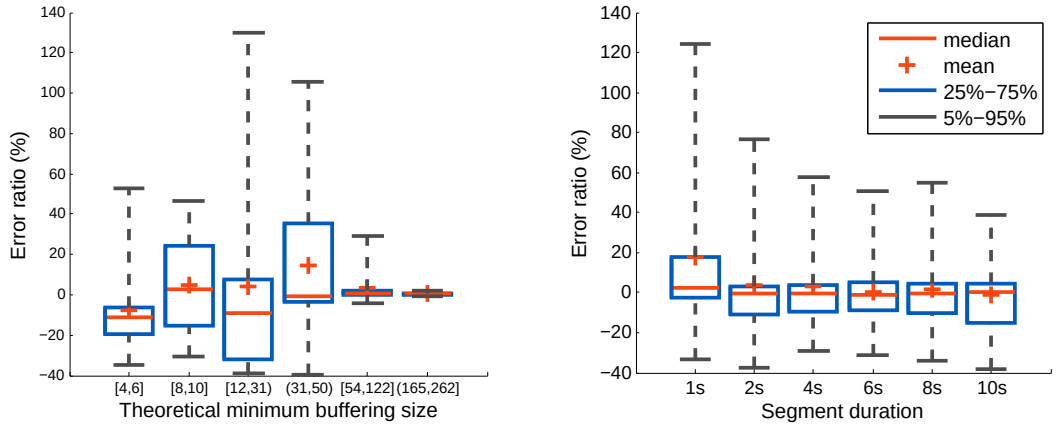
(a) Segment duration of 2 *s*

(b) Various segment durations

Figure 3.5: Error ratio of the approximate and theoretical minimum buffering size in the case of one single network degradation event with the non-constant network throughput, indicated with different percentiles and the mean. a.) Error ratio for scenarios with segment duration 2 *s*. Results are grouped into six sets with equal sizes according to the theoretical minimum. The range for each set is presented via open intervals under the axis. b.) Error ratio for scenarios with segment durations of 1 *s*, 2 *s*, 4 *s*, 6 *s*, 8 *s*, and 10 *s*.

grouped into six sets with equal sizes and an ascending order of theoretical minimum buffer levels. The ranges of minimum buffer levels in each set are indicated with open intervals: $(0, 16)$, $(16, 31)$, $(31, 46)$, $(46, 61)$, $(61, 76)$, and $(76, 104)$. According to the results, our approximation has an average error ratio of up to 0.09% over each set and the approximate values converge to the theoretical minimum along the increase of the minimum buffer level. Moreover, the error ratio of our approximation is 0.02% on average in the case of 2 *s* segment duration. As shown in Figure 3.5b, the average error ratios of each segment duration 1–10 *s* vary from 0.01% to 0.05% .

Multiple network degradation events. The error ratios of the approximates of the minimum buffering size against the theoretical ones are shown via different percentiles and the mean in Figure 3.6. We first look at the results for scenarios with 2 *s* segment duration. To check the impact of theoretical minimums on the error of the approximations, we group the results into six sets with roughly equal sizes. The ranges of minimum buffering sizes in each set are presented with mathematical intervals of sets: $[4, 6]$, $[8, 10]$, $[12, 31)$, $(31, 50)$, $[54, 122]$, and $(165, 262]$. The numbers of results in each set are 11, 12, 10, 10, 10, and 11, respectively. Figure 3.6a shows that the approximation of the minimum buffering size has average error ratios from -8% to 14% over each set of the theoretical minimums. In the case of the large buffering sizes (≥ 54 *s*), the error ratios converge and are significantly lower, with an average of 2%. Moreover, the means of error ratios are much higher than the medians in the case of the minimum buffering sizes from 12 *s* to 50 *s*. This implies that our approximation highly overestimates the minimum buffering size in few cases. For instance, 5% of error ratios are more than 106% in the range of the minimum buffering sizes between 12 *s* and 50 *s*. The reason is that the approx-



(a) Segment duration of 2 *s*

(b) Various segment durations

Figure 3.6: Error ratio of the approximate and theoretical minimum buffering size in the case of multiple network degradation events, indicated with different percentiles and the mean. a.) Error ratio for scenarios with segment duration 2 *s*. Results are grouped into six sets with roughly equal sizes according to the theoretical minimum. The range for each set is indicated via intervals under the axis. b.) Error ratio for scenarios with segment durations of 1 *s*, 2 *s*, 4 *s*, 6 *s*, 8 *s*, and 10 *s*.

imation is biased to the worse cases of network degradation events. Nevertheless, the error ratio of the approximation for scenarios of 2 *s* segment duration is 3% on average as shown in Figure 3.6b.

Figure 3.6b shows the error ratios of the approximation under scenarios of various segment durations: 1 *s*, 2 *s*, 4 *s*, 6 *s*, 8 *s*, and 10 *s*. The average error ratios lie between −1% and 17%. In general, the error ratios are up to 3% on average and reduce along the increase of the segment duration. The segment duration of 1 *s* is the extreme case, in which the approximation has the largest average error ratio. The reason is as follows. The smaller the segment duration is, the more frequently the degradation events occur. Also, the consecutive degradation events with the intervals which allow clients to recover the buffer occupancy to the minimum level prior to the following degradation events occur more frequently, and those intervals are smaller. This results that the second term of our approximation (i.e., B_{min}^{md} from Equation 3.32) yields a larger value. Therefore, the approximate value is more higher than the theoretical one.

In summary, the approximation of the minimum buffering size is close to the theoretical minimum on average, with a near-zero average error ratio in the case of one single network degradation and an average error ratio of up to 17% in the case of multiple degradation events.

3.6 SUMMARY

Our work in this chapter fills the gap of few research on the achievable lower bound of the latency. Specifically, we first analyze the components affecting the streaming latency and identify the buffering size as the main contributor to the latency. This implies that to achieve low-latency streaming, we need to minimize the buffering

size. However, streaming services need to ensure sufficient buffering, in order to offer constant video playback. To study the dynamics of low-latency adaptive streaming, we introduce an analytical model of adaptive streaming for buffer size. With the model, we then derive the minimum buffering size based on the characteristics of the streaming session. The theoretical minimum buffering size can support adaptive video streaming without playback stalls. In addition, we present an approximation of the minimum buffering size which can be computed without the exact future network states.

The work of this chapter provides us with the foundation to evaluate existing adaptation solutions and to design more efficient streaming systems in the context of low latency. It motivates the work of the following chapters. In Chapter 4, we develop a bitrate adaptation algorithm which relies on the idea of the analytical model for buffer size and stabilizes the buffer dynamics by selecting an optimal video bitrate. The performance evaluation is performed using the bitrate-adaptive streaming simulation based on the model and shows that the algorithm is suitable to adaptive streaming with a small buffer and the minimum buffering size. In Chapter 5, we further improve the performance of our algorithm by employing a novel streaming architecture, which achieves near-zero reception delay. Our algorithm benefits from the architecture, because the reception delay directly influences the buffer dynamics in the analytical model.

4

ADAPTATION ALGORITHM: BITRATE SELECTION FOR BUFFER STABILIZATION

From the previous chapter, we know that the buffering size is the main contributor to the latency of a streaming session. Minimizing the latency implies minimizing the buffering size. Moreover, the previous chapter introduces the analytical model for buffer size and then derives the lower bound of the buffering size in view of the network characteristics, such that we can determine a reasonable lower bound on the latency for adaptive streaming. In this chapter, we assume an upper bound of the latency given from the perspective of user's requirement and present a suitably targeted algorithm of bitrate adaptation for low-latency video streaming. We first identify the main challenge of existing algorithms in adaptive streaming with small buffer sizes. Then, we propose an novel adaption algorithm to tackle the open issues and analyze its properties. Our proposed algorithm models buffer dynamics in a way motivated by the model for buffer size. Based on this model, it effectively stabilizes client buffer dynamics using quality-optimized bitrate selection. We call the algorithm *Buffer Dynamics Stabilizer* (BDS).

4.1 CHALLENGES IN LOW-LATENCY ADAPTIVE VIDEO STREAMING

As will be detailed in Section 6.1 and Section 6.3, state-of-the-art bitrate adaptation algorithms achieve acceptable performance with a large buffer size on the order of tens of seconds. This leads to high latency in video delivery, which is undesirable especially in the context of live features. The allowed latency limits the size of the client buffer to seconds of video. Even in VoD scenarios, the buffer size is also bounded due to the requirements on the startup-delay, the video quality, and the resource waste, as mentioned in Section 2.2.2 and Section 2.3.2. Therefore, a reasonable small buffer is required in both live and VoD streaming. However, a small buffer lacks resilience to buffer fluctuation imposed by packet jitters and the mismatch between the network throughput (bitrate) and the bitrate of the selected video segment. Based on our own observations, the following inefficiencies are the main causes for the buffer fluctuation:

First, the underlying TCP transport layer of streaming applications introduces severe packet jitter and unstable throughput due to its congestion and error control [7]. As a result, the network throughput acquired by streaming applications is very unstable, especially in the case of highly variable throughput; e.g., on wireless and mobile Internet paths as well as highly congested paths. In Chapter 5, we will employ an optimized transport protocol instead of TCP to evaluate the effects and to further improve the streaming performance.

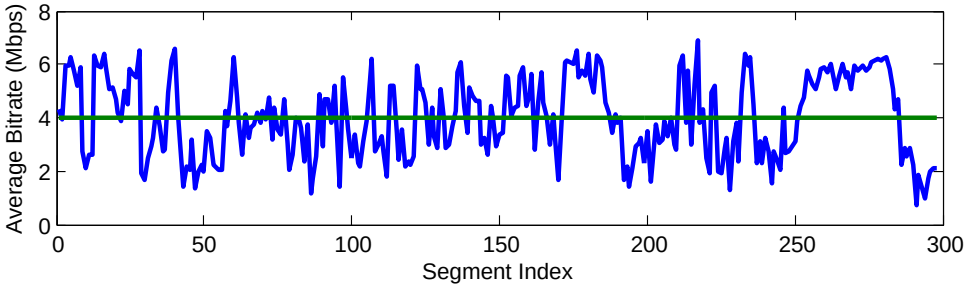


Figure 4.1: Average bitrate of each segment in variable bitrate encoding. The green line indicates the nominal video bitrate (4 *Mbps*) of the *Big Buck Bunny* video sequence [68] with a 2 *s* segment duration.

Second, as the transport-layer throughput is not explicitly known, the application needs to estimate it based on segment reception history. Specifically, the throughput upper bound during the reception of a segment is calculated as the ratio of the size of the segment to the time it took to be received. In practice, the estimation is often inaccurate [57, 58]. In particular, an ON-OFF streaming pattern incurred by a full buffer leads to inaccurate throughput estimates, as described in Section 2.2.3. Worse still, bitrate selection based on such biased estimates may trigger a downward spiral effect, resulting in a unnecessary variable and low video quality [58]. Furthermore, the ON-OFF streaming pattern results in quality variations and unfairness with multiple video streaming sessions [57].

Third, as stated in Section 2.2.7, although most adaptation schemes are client-based due to the scalability and the flexibility, the client feedback introduces a significant delay into the throughput estimation and adaptation in the context of low latency. Thus, it leads to suboptimal bitrate selections. In Chapter 5, we will detail an solution to address this issue.

Last, even in case the transport-layer throughput can be accurately perceived at the application layer, it often significantly differs from the average bitrate of the selected video segment due to a two-fold *quantization error*. Namely, the client's throughput estimate is quantized by a nominal video bitrate that is chosen from a discrete set of quality levels. In addition, the actual average bitrate of any segment of the selected quality level oscillates around the nominal bitrate of the respective level due to VBR encoding, as mentioned in Section 2.2.4. Figure 4.1 gives us a further example with respect to the instantaneous bitrate of a segment along the time, which significantly differs from the nominal bitrate. This leads to larger buffer level fluctuations and increases the likelihood of buffer underflow or suboptimal bitrate selection.

Altogether, these results show that buffer dynamics are unstable at the client side and therefore the stabilization of client buffer dynamics is challenging. A smaller buffer is more challenging for the buffer stabilization. As a result, the performance of streaming applications is very unstable in scenarios with a small buffer, because the client suffers from frequent video playback stalls (rebuffering) and/or low video quality — an empty buffer results in playback stalls, which have the largest impact of the QoE on user engagement; a full buffer leads to sub-optimal video quality. In contrast to most state-of-the-art bitrate algorithms, the focus of our adaptation

algorithm is on the stabilization of buffer dynamics with a small buffer while adapting video quality, rather than on a minimization of the bitrate mismatch.

4.2 BITRATE SELECTION FOR BUFFER STABILIZATION

The first step in our adaptation algorithm is the derivation of a model for client buffer dynamics. We derive this model as a difference in seconds of video between received video and played back video in the buffer and update it at each segment-selection interval. Next, our algorithm selects the VQLs, such that the buffer level is maintained at a desired level. Last, to reduce the rate of VQL switching, we incorporate a simple-but-effective scheme for stable VQL. For the sake of consistency and brevity, we will continue to use the same terminologies and notations as in Chapter 3 throughout the thesis.

4.2.1 *Modeling buffer dynamics*

Let b_i^e be the buffer level in seconds when the client completes the reception of segment i. Similar to the equation of the buffer dynamics (Equation 3.15) in the analytical model of ABS for buffer size (Section 3.2), we estimate b_i^e as follows when the client receives the entire segment $i-1$:

$$b_i^e = b_{i-1}^e + T_c - (d_i + \Delta t_i)\ , \tag{4.1}$$

where b_{i-1}^e is the buffer level when the reception of segment $i-1$ is completed and is initialized with $b_0^e = 0$, T_c is the segment duration, d_i represents the reception duration for segment i, and Δt_i denotes the reception delay between segment i and segment $i-1$. Note that d_i and Δt_i are the estimates here, because their measured values are not available for the estimation of b_i^e. Moreover, in Equation 4.1 we assume that the client receives segments sequentially. Hence, the reception of segment i does not overlap with the reception of segment $i-1$; i.e., it holds $\Delta t_i \geq 0$. This equation implies that when each reception of a segment is completed, the client buffer level increases by T_c (each segment containing a fixed duration of video) and decreases by the corresponding reception duration for that segment (since d_i seconds of video are played back in d_i seconds of real time), if the delay of receiving that segment (i.e., Δt_i) is negligible. In practice, we can achieve near-zero values for Δt_i by employing server-side adaptation. Chapter 5 will detail an architecture of server-side adaptation to minimize the reception delay. We note that Equation 4.1 models the buffer dynamics during the playback phase and the subtractive term $(d_i + \Delta t_i)$ of the equation is omitted in the case of the buffering phase.

Note that the buffer level b_i^e in Equation 4.1 may be negative. A negative b_i^e implies that there exist playback stalls with a duration of $\mid b_i^e \mid$ during the reception of segment i. It should also be noted that b_{i-1}^e is a non-negative value in the equation due to the measurement of the buffer level. This is due to the fact that when the amount of video data in the buffer is not sufficient to decode a single video frame, the client stops video playback until it has enough data again.

As discussed in Chapter 3, the reception delay Δt_i directly influences the buffer dynamics at the client. In particular, it leads to additional decrease of client buffer occupancy and then potentially a buffer underflow according to Equation 3.15.

Therefore, we introduce Δt_i into Equation 4.1 for estimating the future buffer occupancy, so as to improve the accuracy of the estimation and bitrate selections.

The reception duration d_i can be estimated as:

$$d_i = \frac{S_i^R}{a_i} \,, \tag{4.2}$$

where a_i is an estimate of the average throughput (in *kpbs*) achieved while receiving segment i and S_i^R denotes the size (in *kbit*) of segment i encoded at the VQL (the nominal bitrate) R. In the case of CBR encoding, the segment size S_i^R is equal to $T_c \cdot R$; while in the case of VBR encoding, the relationship of $S_i^R \sim R$ can vary over segments. In the sequel, our focus is on the estimation of the throughput.

4.2.2 *Estimating the network throughput*

An accurate throughput estimation can improve streaming performance [130]. In this thesis, we focus on bitrate adaptation algorithms only and assume that techniques for the estimation are available to us. Many techniques of throughput estimation can be found in the literature (e.g., [24, 131, 132, 133]).

We estimate the throughput for video segment $i \geq 2$ as the mean of $A \geq 1$ previous throughput samples:

$$a_i = \begin{cases} \frac{1}{A} \sum\limits_{i-A \leq j < i} \tilde{a}_j & i > A \\ \\ \frac{1}{i-1} \sum\limits_{1 \leq j < i} \tilde{a}_j & else \end{cases} \,, \tag{4.3}$$

where $\tilde{a}_j$ is the average measured throughput (in *kpbs*) experienced during the reception of segment j. Since adaptation algorithms typically initialize the selected video bitrate for the first segment with the lowest bitrate of the adaptation set (i.e., $R_1 = \min\{\mathcal{R}\}$) in order to achieve a low startup delay, the estimated throughput for the first segment a_1 is omitted here. The coefficient A represents the window size of a moving average filter. Which size is selected depends on the type of movement of interest, such as short, intermediate, or long-term.

Although averaging is a simple technique for throughput estimation, our evaluation shows that it gives accurate estimates under the assumption that throughput samples obtained from the transport layer accurately reflect the available network throughput. Namely, the transport protocol can accurately perceive available throughput only if it can increase its transmission rate to the point where it saturates the link. This assumption holds when the bitrate control at the application layer selects segments whose bitrates are slightly larger than, or equal to, the available throughput. However, the bitrate control generally selects segments conservatively with a lower bitrate to avoid playback stalls; the buffer level then starts to increase. When the buffer level reaches its maximum, the client application will pause reading the data, thus triggering the ON-OFF streaming pattern and then affecting the transport protocol's (TCP) congestion and flow control. As a consequence, the TCP sender no longer increases the transmission rate, which prevents accurate estimation of the available throughput for a number of future segments. In other words,

an accurate throughput estimation at the transport layer requires support from the application layer in that the bitrate control performs an accurate selection of the segment bitrate. We refer to such throughput estimation as *system*ic, since the requirement for accurate estimation is a correct operation of several system components, in this case the transport-layer congestion and flow control as well as the application-layer bitrate control.

Furthermore, averaging is a simple technique for throughput estimation, therefore it is also a "stress test" of the effectiveness of the buffer control in our algorithm. Note that smoothed throughput estimates cause delayed responses to large throughput drops, which in turn must be compensated with a large buffer size [25]. Hence, it is challenging to use such a simple estimation technique in the bitrate control of low-latency adaptive streaming. We believe that our algorithm benefits from the accuracy of the throughput estimation, because the more accurately we can estimate the throughput, the more accurate the estimation of buffer dynamics can be.

4.2.3 *Stabilizing buffer dynamics (basic bitrate selection)*

In order to stabilize the buffer dynamics, our segment selection minimizes the deviation of the current buffer level from the desired level. For each segment, we select the nominal bitrate (the VQL) as follows:

$$R_i = \underset{R \in \mathcal{R}}{\arg\min} \left| b_i^e - \beta_{ref} \right| , \tag{4.4}$$

where $\mathcal{R}$ is the set of nominal bitrates of the video, $\beta_{ref} \in (0, \beta_{max}]$ is the desired buffer level, and β_{max} is the maximum buffer level corresponding to the client buffer size. By plugging Equation 4.1 and Equation 4.2 into Equation 4.4, we obtain

$$R_i = \underset{R \in \mathcal{R}}{\arg\min} \left| b_{i-1}^e + T_c - \left(\frac{S_i^R}{a_i} + \Delta t_i \right) - \beta_{ref} \right| . \tag{4.5}$$

The selection of the desired buffer level affects the streaming performance of the algorithm. Because we focus on a small buffer size (i.e., β_{max} is small) and playback stalls have the largest impact on user engagement, β_{ref} is normally set to a value above $0.5 \cdot \beta_{max}$, so that the buffer level has more room to stay away from zero level. However, as mentioned in Section 4.1, a full buffer (i.e., the buffer level is at its maximum level in seconds) disturbs the transport-layer throughput and then may lead to sub-optimal video quality and fairness issues with other competing flows. It implies that β_{ref} cannot be too close to β_{max}. Section 4.4.3 will give a further discussion on the impact of the desired buffer level on streaming performance. Based on our observations, we suggest that $\beta_{ref} = 0.8 \cdot \beta_{max}$.

As the goal of our bitrate adaptation is to reduce the buffer fluctuation while the bitrate selection is being performed, we refer to our algorithm as *Buffer Dynamic Stabilizer* (BDS).

Note that our bitrate selection of BDS (Equation 4.5) minimizes the quantization error (as mentioned in Section 4.1), which is due to two factors: i.) deviation of the nominal bitrate (R) from the average bitrate of the segment ($\frac{S^R}{T_c}$) due to VBR encoding and ii.) deviation of the desired bitrate (r) from the nominal bitrate. The reason

is that we directly apply the sizes of candidate segments (i.e., S_i^R) into the minimization of the buffer deviation. As a result, R_i from Equation 4.5 is the nominal bitrate that minimizes the deviation of the average bitrate from the desired bitrate; namely,

$$\forall R \in \mathcal{R} \text{ and } R \neq R_i : \left| r_i - \frac{S_i^{R_i}}{T_c} \right| < \left| r_i - \frac{S_i^R}{T_c} \right| , \tag{4.6}$$

where r_i is the desired bitrate for segment i that minimizes the buffer deviation. A proof of Inequality 4.6 is in Appendix A.2.

4.2.4 *Stabilizing video quality levels (improved bitrate selection)*

Based on Equation 4.4, our algorithm chooses the VQLs such that the buffer levels converge to a specific value. However, this choice does not take the stability of VQLs into account and thus may lead to frequent VQL switching, especially in the case of highly variable network throughput and/or video complexity. Therefore, we present a modified version of BDS to reduce the switching rate. The modification introduces a specific region of the buffer level for non-switching VQLs as shown in Algorithm 4.1. In particular, let β_l and β_u denote the lower threshold and the upper threshold of a buffer region, respectively. We select the nominal bitrate $R_i = R_{i-1}$, if the estimated buffer level $b_i^e \in [\beta_l, \beta_u]$. We will see in Section 4.4.2 that by stabilizing VQLs, BDS benefits from the modification in terms of the QoE. We will also explore the impact of β_l and β_u on the performance in Section 4.4.3. For brevity, we call the basic version of BDS, directly from Equation 4.5, BDS-0, and call the modified version, as described in Algorithm 4.1, BDS-1. To allow a fast startup playback, BDS-0 and BDS-1 both select the video segments with the lowest bitrate of the adaptation set during the buffering phase; i.e., $\forall 1 \leq i \leq m : R_i = \min\{\mathcal{R}\}$, where m is the number of received segments for the buffering before the first video playback. The input of the algorithm b_{i-1}^e is a direct measurement of the client buffer level. The inputs a_i and Δt_i are the estimates based on their history measurements; they are more accurate if the application layer can retrieve those information from the transport layer. Other inputs are available as predefined values and dynamic values directly queried from streaming applications.

Since BDS aims to stabilize the buffer dynamics away from the zero and maximum buffer level, it can significantly reduce the OFF period of the ON-OFF streaming pattern as mentioned in Section 2.2.3. As a result, BDS mitigates the disturbance of the transport-layer throughput by the application-layer adaptation decisions, so as to solve the issues of quality variations and unfairness imposed by the disturbance between the two layers.

4.3 OPTIMAL BITRATE SELECTION

The ultimate goal of adaptation algorithms is to deliver the highest possible user's QoE by selecting the quality level for future video segments. Thus, it allows streaming services to achieve higher long-term user engagement [55]. Depending on the definition of the QoE, the optimum of QoE can be characterized by e.g., maximizing the video bitrate while minimizing the likelihood of playback stalls and avoiding too many bitrate switches.

Algorithm 4.1 BDS Algorithm (an improved version, BDS-1)

Input: a_i, b^e_{i-1}, Δt_i, R_{i-1}, β_l, β_u, β_{ref}, m, T_c, $\mathcal{R}$, $\forall R \in \mathcal{R} : S^R_i$
Output: R_i

1: **if** $1 \leq i \leq m$ **then** ▷ buffering phase
2: $R_i = \min\{\mathcal{R}\}$
3: $b^e_i = b^e_{i-1} + T_c$
4: **else if** $i > m$ **then** ▷ playback phase
5: $b^e_i = b^e_{i-1} + T_c - \left(\frac{S^{R_{i-1}}_i}{a_i} + \Delta t_i \right)$
6: **if** $b^e_i \in [\beta_l, \beta_u]$ **then**
7: $R_i = R_{i-1}$
8: **else** ▷ basic bitrate selection, BDS-0
9: $R_i = \underset{R \in \mathcal{R}}{\arg\min} \left| b^e_{i-1} + T_c - \left(\frac{S^R_i}{a_i} + \Delta t_i \right) - \beta_{ref} \right|$
10: **end if**
11: **end if**

4.3.1 *QoE function*

While the definition of the QoE may differ across users, the four key components of the QoE are listed as follows.

- *Average video quality*. Consider that there are N video segments received in a streaming session. The average per-segment quality (nominal video bitrate) over all segments can be expressed as

$$\frac{1}{N} \sum_{i=1}^{N} R_i \ . \tag{4.7}$$

- *Average quality variations*. This characterizes the magnitude of the bitrate change in the quality between every two consecutive segments:

$$\frac{1}{N-1} \sum_{i=1}^{N-1} |R_{i+1} - R_i| \ . \tag{4.8}$$

- *Startup delay* T_S. This delay indicates the reaction time for the playback start. In practice, its value is a direct measure of the time interval between the initiation of a request and the playback start at the client. The startup delay of an algorithm depends on the buffering size and the bitrate selection strategy during the buffering phase applied by the algorithm. Thus, it also reflects the streaming latency. To allow a fair comparison, we apply the same buffering size to all algorithms evaluated in this thesis, unless noted otherwise.

- *Total rebuffering duration* T_{ST}. A metric represents the total time of all stalling events during the playback of the entire streaming session due to a buffer underflow. In practice, the value is the sum of all measured durations of rebuffering events that occurred in the course of the playback.

According to user preferences, the QoE function $QoE : \mathcal{R}^N \to \mathbb{R}$ can be defined as

$$QoE = \sum_{i=1}^{N} R_i - \lambda \sum_{i=1}^{N-1} |R_{i+1} - R_i| - \mu \cdot T_S - \nu \cdot T_{ST} \,, \tag{4.9}$$

where λ, μ, and ν are the non-negative weighting factors for bitrate variations, startup delay, and rebuffering duration, respectively. A larger λ indicates that the user prefers less quality variations. In the case of low-latency streaming, we can employ a large μ to the QoE function. A larger ν enables user preferences for a smoother video playback with less stalls.

Defining suitable QoE functions for video quality is an open issue [134]. The QoE can be mapped by various function models such as based on linear, logarithmic or exponential functions. Even QoE functions can model user-perceived video quality based on subjective test results (e.g., [135]). Here, we adopt a simple linear model, which is the weighted sum of different factors. Nevertheless, our QoE function is quite general such that it can model various user preferences by applying different weighting factors.

4.3.2 *QoE Optimization Problem*

The object of bitrate adaptation is to maximize the QoE of the users. Based on the adaptive streaming model introduced in Section 3.2, the QoE optimization problem for bitrate adaptation can be formulated as:

$$\begin{aligned} \underset{R_1, \ldots, R_N}{\text{maximize}} \quad & QoE \\ \text{subject to} \quad & \text{Equations 3.10–3.16}, \\ & b_i^s \in [0, \beta_{max}], \\ & B^d \in (0, \beta_{max}], \\ & R_i \in \mathcal{R}, \ \forall i = 1, \ldots, N. \end{aligned} \tag{4.10}$$

β_{max} and B^d are assumed to be a multiple of segment duration T_c, as in our adaptive streaming model for buffer size. In Problem 4.10, B^d is a known variable of the QoE optimization. This formulates the optimization problem of a streaming system in which the buffering size is fixed and given. However, it should be noted that B^d can be considered as a decision variable. Namely, B^d can also be an output of the optimization of Problem 4.10, apart from selected bitrates R_1, ..., R_N. In this case, solving the optimization problem yields the optimal bitrate selection and the optimal buffering size, with which the maximum QoE can be achieved. Note that if the objective function QoE imposes sufficiently large penalties on the startup delay and the playback stalls, this optimization problem can solve the minimum buffering size as presented in Section 3.3.

Our formulation of the QoE optimization problem is sufficiently generalized to be applicable to any QoE function. Here, we apply the QoE function defined as Equation 4.9 to Problem 4.10. Because the selected bitrate R_i for each segment is an available variable in our adaptive streaming model, we can freely determine the values for the first two components of the QoE (i.e., video quality and quality

variations) based on Equation 4.9. According to our streaming model, we then define the other two components as follows.

- *Startup delay* T_S. By definition, it can be determined by

$$T_S = t^e_m - t^r_1 \,, \tag{4.11}$$

  where the buffering size contains $m \geq 1$ video segments (i.e., $B^d = m \cdot T_c$), t^e_m represents the end time of the reception for segment m, and t^r_1 is the request time of the first segment at the client. Note that T_S can also be a decision variable of the optimization problem, like the buffering size B^d. On the other hand, T_S can be a fixed and given variable for the optimization. For instance, it can be integrated into the optimization problem as a constraint, in cases where the user prefers to achieve a specific T_S, especially a specific small value.

- *Total rebuffering duration* T_{ST}. After the buffering phase, for each reception of a segment, rebuffering occurs in two cases: i.) it occurs during the reception if the reception duration is higher than the playout buffer level; ii.) it occurs between the reception of two consecutive segments if the reception delay is higher than the buffer level succeeding the entire reception of one segment. Cf. Equation 3.15, the total rebuffering duration is thus calculated as

$$T_{ST} = \sum_{i=m+1}^{N} \left((d_i - b^s_i)_+ + \left(\Delta t_{i+1} - T_c - (b^s_i - d_i)_+\right)_+ \right) \,, \tag{4.12}$$

  where d_i denotes the reception duration of segment i (i.e., the time required to receive segment i in its entirety).

Given network throughput and delay traces as input, the optimization of Problem 4.10 outputs a sequence of selected bitrates R_1, ..., R_N, which represents an optimal solution of bitrate selection. Let OPT denote the optimal solution of bitrate selection. Given an OPT, Equation 4.9 yields the maximum QoE, denoted by $QoE(OPT)$. This implies that the maximum QoE can be achieved with perfect knowledge of future network throughput and delay. In Section 4.4.5, we demonstrate an example of optimal bitrate selections for typical network scenarios.

4.4 PERFORMANCE EVALUATION

We evaluate the algorithm performance using modeling and simulation of adaptive video streaming. Our evaluation methodology consists of a prototype implementation, a benchmark comparison with state-of-the-art algorithms, as well as a simulation of a streaming model and the dynamic network conditions.

4.4.1 *Methodology*

Our evaluation methodology combines a bitrate-adaptive streaming simulation with a trace-driven simulation of a mobile network. Our choice to use the simulation is motivated by the need to find out how far away existing algorithms are from the optimal solutions.

Performance metric. Our evaluation employs a normalized QoE metric based on the streaming model introduced in Section 3.2. As described in the previous section, given an optimal solution of bitrate selection OPT, the QoE function QoE returns the maximum QoE $QoE(OPT)$. The optimum OPT can be computed by solving the optimization problem with the corresponding QoE function as well as perfect knowledge of future network throughput and delay. Because the assumption of perfect knowledge about the future is not true in reality, for any bitrate adaptation algorithm, $QoE(OPT)$ is an upper bound of the achievable QoE.

Let $QoE(A)$ be the QoE achieved by a bitrate adaptation algorithm A. *Normalized QoE* of A is defined as the evaluation metric for algorithm A:

$$n\text{-}QoE(A) = \frac{QoE(A)}{QoE(OPT)} \,. \tag{4.13}$$

We specify a default QoE function (formulated by Equation 4.9) by setting $\lambda = 1$ and $\mu = \nu = 6000$. That is, $1\,s$ rebuffering or startup delay has the same penalty as the bitrate reduction of a segment by $6000\,kbps$, corresponding to the maximum bitrate in the adaptation set. Based on the specific QoE function, we compute the QoE achieved by any algorithm and the maximum QoE by solving Problem 4.10.

Streaming simulation. We continue to apply the simulation testbed used in Section 3.5; i.e., a simulation is implemented based on our model of adaptive streaming using Matlab. We also integrate the prototype implementation of the evaluated algorithms into the simulation. Given throughput and delay traces, the testbed simulates the streaming process, the bitrate selection of the algorithm, and the buffer dynamics at the client. Furthermore, we solve the QoE optimization problem (formulated by Problem 4.10) under the Matlab simulation, using an optimization solver (CPLEX[1]). This gives us the maximum QoE required in the normalized QoE metric. A validation of this simulation testbed will be discussed in Section 4.4.5.

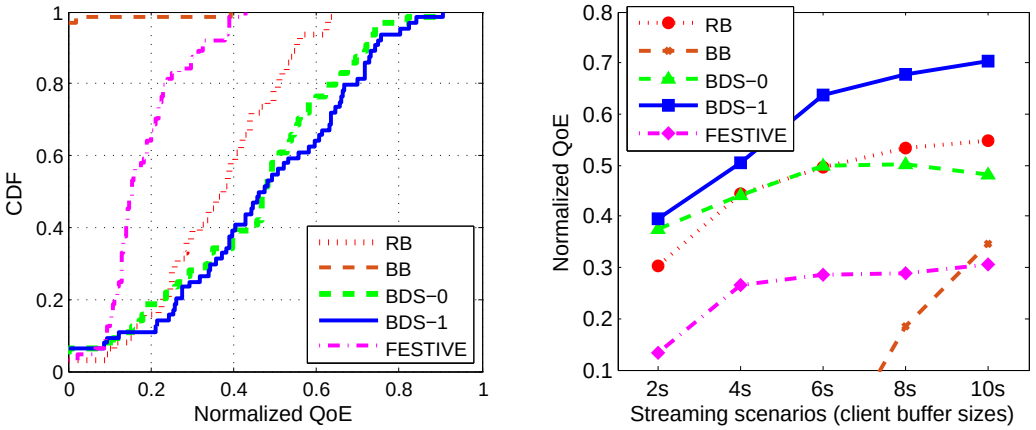
Algorithms. We first implemented a basic version of BDS (BDS-0) formulated by Equation 4.4. For simplicity, the desired buffer level β_{ref} of BDS-0 is chosen to be $0.5 \cdot \beta_{max}$. We also implemented an improved version of BDS (BDS-1) as shown in Algorithm 4.1, in order to examine the effect of the modification with direct comparison to BDS-0. We set the parameters of BDS-1 $\beta_{ref} = 0.5 \cdot \beta_{max}$, $\beta_l = 0.1 \cdot \beta_{max}$, and $\beta_u = 0.9 \cdot \beta_{max}$. Using a large buffer region for stabilizing VQLs implies that BDS can achieve a low rate of bitrate switching. The impact of the different parameters on the QoE will be discussed in Section 4.4.3.

Our benchmark adaptation algorithms are RB [78], BB [30], and FESTIVE [29].

- RB is a rate-based algorithm and follows the algorithm applied in DASH VLC plugin [78], which simply chooses the highest bitrate smaller than the throughput estimate without considering the effect of previous bitrates. Most commercial players appear to employ this *stateless* adaptation algorithm [29].

- We use the function introduced by Huang *et al.* [30] as a representative of buffer-based algorithms (BB). The function with the reservoir $re = 0.1 \cdot \beta_{max}$ and the cushion $cu = 0.8 \cdot \beta_{max}$ maintains a mapping between buffer occupancy and video bitrates.

1 http://www.ibm.com/software/products/en/ibmilogcpleoptistud



(a) CDF of the normalized QoE

(b) Average normalized QoE

Figure 4.2: QoE performance of the five adaptation algorithms. a.) CDF of normalized QoE for scenarios with the client buffer size of 2 s. b.) Average normalized QoE over all throughput trace cases for scenarios with client buffer sizes of 2 s, 4 s, 6 s, 8 s, and 10 s.

- FESTIVE balances both efficiency and stability with respect to video quality. Since we could not find a free implementation of FESTIVE, we developed our own implementation for FESTIVE and applied all parameters following the description in the original paper [29].

Therefore, our performance comparison contains five sets of performance results: RB, BB, BDS-0, BDS-1, and FESTIVE. Due to our focus on low-latency streaming, we specifically use small client buffer sizes of 2 s, 4 s, 6 s, 8 s, and 10 s. Correspondingly, we set the buffering sizes same as the client buffer sizes (i.e., $B^d = m \cdot T_c$ with $m = 1, 2, 3, , 4,$ and 5).

Video profiles. We use the *Red Bull Play Streets* test sequence (genre: sports) with a segment duration of 2 s [68]. The adaptation set of video segments includes 9 bitrates from 100 $kbps$ to 6000 $kbps$ using VBR encoding [68]: $\mathcal{R} = \{100\,kbps, 200\,kbps, 300\,kbps, 500\,kbps, 900\,kbps, 1500\,kbps, 2500\,kbps, 4000\,kbps, 6000\,kbps\}$.

Network profiles. We employ the same network profiles as described in Section 3.5.1. Namely, we simulate a mobile network based on the throughput dataset [129], which consists of video streaming throughput measurements of a moving device in a mobile network. Since the dataset does not include network delay measurements, we assume a one-way delay of 50 ms on the network (i.e., $d_i^n = d_i^r = 50\,ms$ of the adaptive streaming model introduced in Section 3.2), according to the median RTT of 100 ms observed empirically in prior work [136, 137].

4.4.2 *Benchmark Results*

According to the above experimental setup, we perform a benchmark comparison of five adaptation algorithms (RB, BB, BDS-0, BDS-1, and FESTIVE) over all throughput trace cases. We plot both the empirical Cumulative Distribution Function (CDF) and the averages of the normalized QoE as shown in Figure 4.2.

Figure 4.2a shows the CDF of the normalized QoE in scenarios with the client buffer size of 2 *s*. Our proposed algorithms (BDS-0 and BDS-1) outperform other algorithms with an increase of the median normalized QoE by at least 11%. The improvement of the median reaches even the 32% of the normalized QoE, compared to FESTIVE. BB offers the worst performance on the normalized QoE and its achievable QoE is up to 39% of the optimum. Since the results of BB is significant worse than other algorithms in small buffer sizes, we omit to show the results with the negative values of normalized QoE in the figures, in order to better distinguish between the different curves using zoom-in. Moreover, we omit the results of the CDF for other client buffer sizes and report the averages of the normalized QoE instead, in order to provide an illustrative summary of the results.

Figure 4.2b shows the average normalized QoE in scenarios with different client buffer sizes (2 *s*, 4 *s*, 6 *s*, 8 *s*, and 10 *s*). First, BDS-1 achieves the best performance in all scenarios with different buffer sizes. Specifically, BDS-1 has 6–15% and 24–40% higher average normalized QoE, compared to RB and FESTIVE, respectively. Similar to the case of 2 *s* buffer size, BB gets the worst performance over different buffer sizes, except that it obtains 4% higher normalized QoE than FESTIVE on average in the scenario with 10 *s* buffer size. Second, as expected BDS-1 improves the normalized QoE with the scheme for stabilizing VQLs. We see that BDS-1 achieves a 2–22% higher average normalized QoE compared to BDS-0. This reflects the effectiveness of our simple modification with respect to quality stabilization. Third, we observe that all algorithms benefit from large client buffer sizes. For each 2 *s*-increase in the buffer size, BB has an increase of average normalized QoE by at least 16%, while the average normalized QoE of RB, BDS, and FESTIVE improves by up to 14%. The one exception is that BDS-0 suffers from a QoE degradation by 2% when the buffer size increases from 8 *s* to 10 *s*. This is due to the fact that BDS-0 relatively frequently switches VQLs in the case of a large buffer size; in our experiments we observe that average quality variations of BDS-0 increase when the buffer size increases from 6 *s*. Finally, as existing algorithms achieve only less than 71% of the optimal QoE on average, it suggests that there is still room for the improvement.

The reasons for the superior performance of BDS against other algorithms are summarized below.

- The bitrate selection of BDS aims to maintain stable dynamics of the client buffer with a small size. Accordingly, BDS is able to control the buffer at near-zero levels and to reduce the possibility of rebuffering. On the other hand, the stabilization of buffer dynamics with a nearly full buffer supports systemic throughput estimation, thereby increasing the accuracy of throughput estimation at the application layer and lessening the interactive disturbance between the transport-layer throughput and the application-layer adaptation decisions.

- In contrast, RB performs its bitrate selection based on the measurement of the long-term average throughput. However, it lacks of a mechanism for buffer stabilization. This often leads to significant variations of the buffer level due to incorrect throughput estimates. Thus, it suffers from a worse QoE performance. This inferior performance can be compensated with a larger buffer.

- BB maps the ranges of buffer levels to video bitrates and performs the selection based on the mapping. By allocating a specific range from the limited

buffer size for each bitrate, it enables quasi-stabilization of buffer dynamics. However, the stabilization is very limited, as only a small portion of the whole buffer is used for a specific bitrate, compared to BDS. Therefore, although BB achieves the worst performance in scenarios of small buffer sizes, its performance significantly improves with the increase of the buffer size.

- FESTIVE favors small quality variations over higher video quality. As a result, it achieves a very low average video quality and its achievable QoE is significantly worse than that of other algorithms. Although FESTIVE allows to balance video quality and quality variations by tuning the parameters, we find that it is complex to achieve both high average video quality and low rebuffering, without taking rebuffering metrics into account during the design of the algorithm.

In summary, BDS outperforms the state-of-the-art adaptation algorithms with at least 6% higher average normalized QoE, despite a client buffer size as small as a single segment duration (2 s).

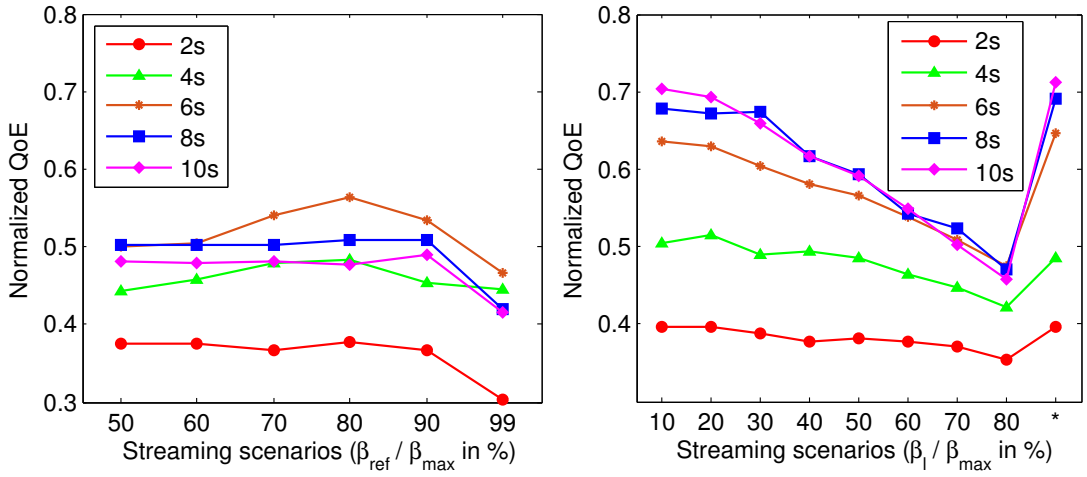
4.4.3 *Impact of parameters*

Our algorithm BDS has three main free parameters (β_{ref}, β_l, and β_u), which affect the performance of the algorithm. In the following, we analyze the impact of these parameters on the QoE performance.

Desired buffer level. There is a tradeoff in choosing a suitable value for β_{ref}. Setting it to too high or too low causes low video quality or playback stalls as mentioned in Section 4.1. Figure 4.3a shows the impact of β_{ref} in BDS-0 on the average normalized QoE over all throughput trace cases. We focus on the evaluation of the desired buffer level β_{ref} in the range of 50–100% of the maximum buffer level β_{max}. The ratio β_{ref}/β_{max} increases in steps of 10%. Empirically, we observe the maximum of the average normalized QoE when the ratio β_{ref}/β_{max} is set to a value between 70–90%, in particular, when $\beta_{ref}/\beta_{max} = 80\%$. We also observe that the average normalized QoE generally increases along the increase of the desired buffer level, specifically by up to 4% for each 10%-increase of β_{ref}/β_{max} from 50% to 90%.

Threshold buffer levels. Recall that BDS-1 does not switch the VQL when the buffer level is between the lower threshold level β_l and the upper threshold level β_u, i.e., when $b(t) \in [\beta_l, \beta_u]$. The larger the interval $[\beta_l, \beta_u]$ is, the more room is allowed to reduce the switching rate. When the buffer level is frequently close to its maximum it incurs a poor performance, as also evidenced by the results for $\beta_{ref}/\beta_{max} = 99\%$ in Figure 4.3a. Therefore, we fix $\beta_u/\beta_{max} = 90\%$ and explore the impact of $\beta_l/\beta_{max} \in [10\%, 80\%]$ on the QoE performance. Based on the above observations, we set $\beta_{ref}/\beta_{max} = 80\%$. Figure 4.3b shows that the average normalized QoE generally decreases along the increase of β_l/β_{max} in scenarios with different client buffer sizes. We observe that the maximum is achieved if $\beta_l/\beta_{max} = 20\%$ in the case of client buffer sizes of 2 s and 4 s. The maximum is up to 2% larger than the second maximum, which is achieved when $\beta_l/\beta_{max} = 10\%$.

As our basic bitrate selection (BDS-0) can effectively control a buffer level as small as a segment duration, we "aggressively" suggest a fixed value for β_l of as small as a segment duration if the buffer size is large than the segment duration; e.g., in our



(a) β_{ref}/β_{max} of BDS-0 (b) β_l/β_{max} of BDS-1

Figure 4.3: Impact of BDS's parameters on normalized QoE for scenarios with client buffer sizes of 2 s, 4 s, 6 s, 8 s, and 10 s, on average of 64 mobile throughput traces. In b.), the scenario marked with the asterisk on the x-axis indicates the "best" case where $\beta_l = T_c$, if $\beta_{max} > T_c$, otherwise $\beta_l = 0.1 \cdot \beta_{max}$.

experiments we use $\beta_l = T_c$, if $\beta_{max} > T_c$, otherwise $\beta_l = 0.1 \cdot \beta_{max}$. The results for this suggestion are indicated by the asterisk as the "best" case in Figure 4.3b. The results show that it achieves the best performance in scenarios with client buffer sizes of 6 s, 8 s, and 10 s, with an increase of the average normalized QoE by up to 2% compared to the second best. The one exception is the case of $\beta_{max} = 4\,s$ where the achievable average normalized QoE is 3% less than the maximal.

In summary, based on the above observations, we suggest the following parameters used in BDS: $\beta_{ref} = 0.8 \cdot \beta_{max}$; $\beta_u = 0.9 \cdot \beta_{max}$; $\beta_l = T_c$, if $\beta_{max} > 2 \cdot T_c$, otherwise $\beta_l = 0.2 \cdot \beta_{max}$.

4.4.4 *Performance with Minimum Buffering Size*

In the previous chapter, we introduce the minimum buffering size, with which streaming services can ensure a constant video playback without rebuffering even in case of the network throughput lower than the guaranteed throughput (i.e., in case of the occurrence of the network degradation). In this subsection, we evaluate the performance of the four adaptation algorithms — i.e., RB, BB, BDS-1, and FESTIVE presented in Section 4.4.1 — under the constraint of the minimum buffering size.

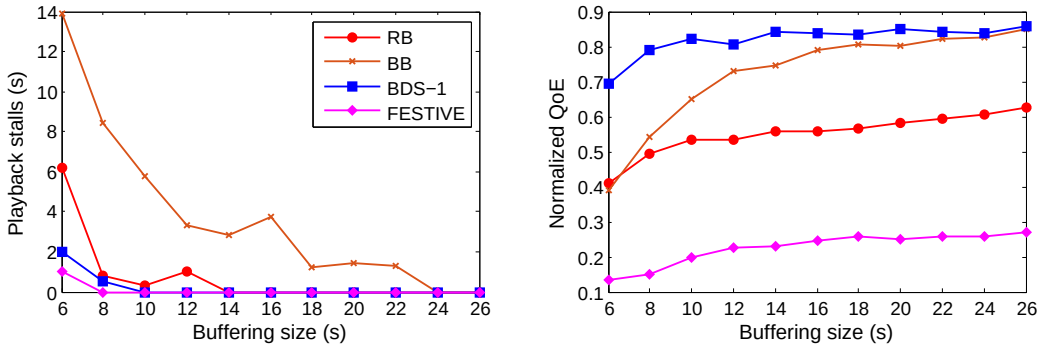
Section 3.5 solves the theoretical minimum buffering sizes for each streaming session with the given video and network profiles. Figure 3.6a groups the results of the minimum from 4 s to 262 s into six sets with roughly equal sizes. Due to our focus on low-latency streaming, we use the set of minimum buffering sizes $[4, 6]\,s$ with 2 s segment duration. We randomly pick one from the ten throughput traces in the set, which consists of 631 throughput samples and requires a theoretical minimum buffering size of 6 s. We evaluate the four algorithms with this trace with the theoretical minimum buffering size. The parameters of BDS is set following the

suggestion in Section 4.4.3. The maximum client buffer size is equal to the buffering size for all algorithms.

We perform the evaluation over scenarios with increasing buffering sizes from 6 s to 44 s in steps of 2 s (i.e., totally 20 scenarios), because algorithms may not achieve the same result as the ideal algorithm does — i.e., experiences no playback stalls. Figure 4.4 shows the performance of algorithms with respect to the total time of playback stalls and the normalized QoE from 6 s to 26 s. We omit the results of scenarios with buffering sizes of $> 26\,s$ due to the convergence of the results. The performance analysis here is performed with respect to the total time of the playback stalls experienced by an algorithm with the minimum buffering size during the whole streaming session, because the metric of playback stalls has the largest impact of the QoE on user engagement and the minimum buffering size is obtained under the constraint of a continuous playback.

Results. Figure 4.4a shows that FESTIVE experiences the least playback stalls (of 1 s with the theoretical minimum buffering size) compared to the other three algorithms and can provide a continuous playback with a buffering size of $\geq 8\,s$. However, it achieves the lowest QoE as shown in Figure 4.4b. The reason is that FESTIVE tends to maintain a stable video quality against higher video quality and therefore obtains the lowest average bitrate in a highly variable throughput scenario. BDS achieves the second best performance with respect to playback stalls and the best QoE performance, respectively. Specifically, playback stalls drops from 2 s to 0.5 s for an increase of buffering size from 6 s to 8 s and down to 0 s for a buffering size of $\geq 10\,s$; the normalized QoE reaches 0.82 in the case of 10 s buffering size and converges to around 0.85 for a buffering size of $\geq 14\,s$. In particularly, the average video bitrate achieved by BDS converges to 1.66 $Mbps$ for a buffering size of $\geq 16\,s$ while the available network throughput is 1.78 $Mbps$ on average. Our observations conform to the results presented in Section 4.4.2. BDS effectively stabilizes the buffer dynamics by selecting appropriate video bitrates and keeps the bitrates high and stable if the client buffer level lies in the specific region. Although BB achieves the second best QoE performance of all four algorithms, it suffers from the worst performance with respect to playback stalls. BB only benefits from a large buffering size, because its bitrate selection is based on a mapping between buffer levels and video bitrates. A larger buffering size allows more space to separate different bitrates such that the algorithm can select an adequate bitrate. Our observations on BB also conform to the results presented in Section 4.4.2. RB obtains a relative high and stable average video bitrate due to the bitrate selection based on the measurement of the long-term average throughput. However, incorrect throughput estimates lead to significant variations of the buffer level. Therefore, it suffers from more playback stalls, compared to BDS. Although the normalized QoE of RB increases along the increase of the buffering size, the improvement is not significant.

In summary, BDS is most suitable for low-latency streaming, compared to the other state-of-the-art algorithms. It achieves the best QoE performance in all scenarios and no playback stalls with a buffering size of two more segment durations than the theoretical minimum buffering size.



(a) Total time of playback stalls

(b) Normalized QoE

Figure 4.4: Performance evaluation of bitrate adaptation algorithms with respect to the total time of playback stalls and the normalized QoE during the streaming session under the constraint of various buffering sizes. The theoretical minimum buffering size is 6 s and the segment duration is 2 s.

4.4.5 *Discussion*

We note that our results are based on the simulation of a video streaming model, and therefore may not be representative of realistic results. To validate our modeling and simulation as well as to provide relatively realistic results, we conduct a further experiment using a combination of the prototype implementation and the emulation network. It should be noted that this is not a rigorous validation of our simulation for video streaming process in practice. Nevertheless, in the following we will explore simulation results compared to emulation ones. We will see that our simulation results can at least match the improvement of the algorithms, although the exact numerical improvements are scenario-dependent. This implies that our simulation may serve as a testbed for the foundation of potential improvements.

Implementation and emulation setup. Our streaming prototype implementation is Linux-based and is deployed in the VLC media player[2]. The client buffer size is as small as a segment duration; i.e., $\beta_{max} = 2\,s$. Based on given network profiles, we control the available throughput of the end-to-end bottleneck via a network emulation tool, Dummynet[3] [138]. Here, we consider three typical network throughput scenarios as shown in Figure 4.5:

(a) Long-term throughput variations: an abrupt change between 1 $Mbps$ and 3 $Mbps$ occurs every 30 s;

(b) Short-term throughput variations: the same change as in long-term variations, however, occurs every 2 s;

(c) Mobile throughput dataset (HSDPA): a random one of aforementioned 64 throughput traces in Section 4.4.1.

All other setup variables are same as in Section 4.4.1.

2 http://www.videolan.org/vlc/index.html
3 http://info.iet.unipi.it/~luigi/dummynet/

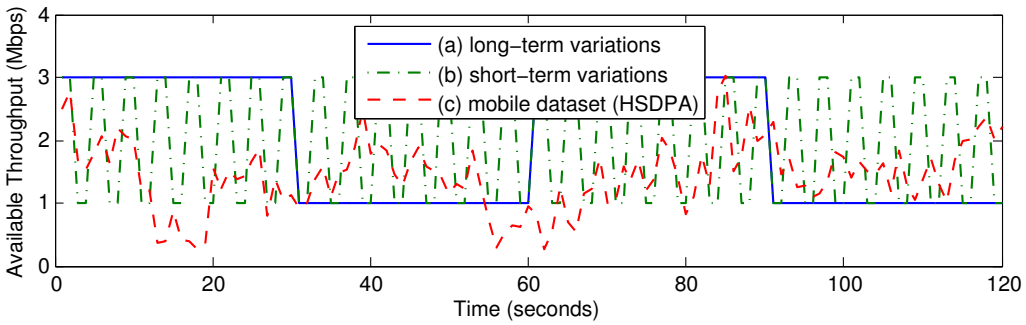


Figure 4.5: Network throughput scenarios

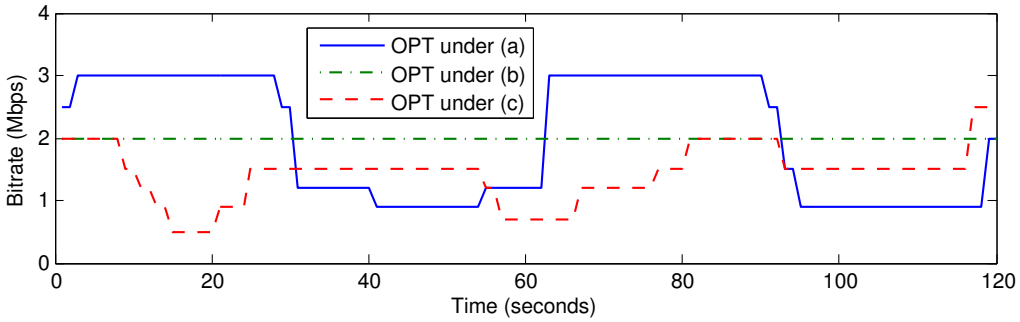


Figure 4.6: The optimum *OPT* of bitrate selection under each of three scenarios: (a) long-term throughput variations; (b) short-term throughput variations; (c) mobile throughput traces.

Results. We begin with Figure 4.6, which shows an optimal solution of bitrate selection under each of three network throughput scenarios. First, the selected bitrates of the optimal solutions accurately track the dynamics of the network throughput without frequent and significant bitrate switches. Second, the optimum incurs no rebuffering events. These results confirm the objective of the simulation and the optimization problem.

Figure 4.7 shows the performance of three algorithms (RB, BDS-0, and FESTIVE) in terms of the normalized QoE over three different throughput scenarios in both simulation and emulation. We omit the results for BDS-1 for brevity, since BDS-1 performs slightly better than BDS-0 in this experiment. We see that BDS-0 achieves the best performance in all cases and RB the second best. Specifically, BDS-0 increases the QoE performance at least by 61% in the simulation and RB at least by 43%, compared to FESTIVE. Similarly, the normalized QoEs of BDS-0 and RB in the emulation are increased at least by 45% and 41%, respectively. Furthermore, the performance of BDS-0, RB, and FESTIVE in the simulation are generally up to 22%, 6%, and 11% higher than the performance in the emulation, respectively. We conjecture that the better performance in the simulation is due to the fact that our modeling and simulation represent an ideal and theoretical behavior of bitrate adaptation. The one exception, which sounds abnormal, is that the simulation performance of RB is slightly lower than the emulation one in the case of long-term throughput variations, specifically 0.9% lower. However, the results in both cases (the simulation and the emulation) are quite comparable and do not contradict in terms of the performance ranking between the three algorithms.

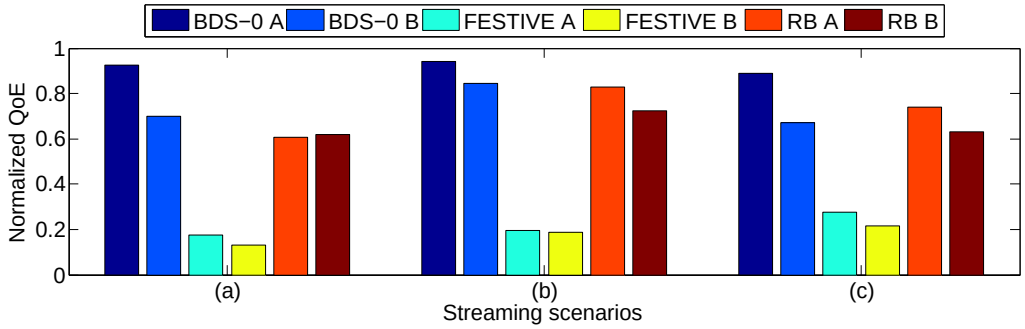


Figure 4.7: Normalized QoE achieved by three algorithms (BDS-0, FESTIVE, RB) over three scenarios: (a) long-term throughput variations; (b) short-term throughput variations; (c) mobile throughput traces. The letters A and B following each algorithm name indicate the performance of that algorithm in the simulation and the emulation, respectively.

4.5 SUMMARY

In this chapter, from the perspective of the upper bound of streaming latency, we propose a novel bitrate adaptation algorithm for low-latency dynamic video streaming that stabilizes a small client buffer. Our algorithm models buffer dynamics in a way motivated by the analytical model for buffer size (Section 3.2) and performs a quality-optimized bitrate selection based on the model of buffer dynamics. In particular, we pose the buffer stabilization as an optimization problem where the quality level of each segment is determined by i.) jointly considering a switch decision of the quality level according to the specific buffer region and ii.) selecting the quality level that minimizes the deviation of the buffer level from the desired buffer level.

We demonstrate the performance of our algorithm using a simulation methodology, which combines a bitrate-adaptive streaming simulation with a trace-driven simulation of mobile network in the context of video streaming. The results of simulation experiments and prototype trials show that our algorithm effectively stabilizes the buffer at a level as small as a single segment duration (2 s) and achieves at least 6% higher QoE, compared to the state-of-the-art algorithms. We also explore the impact of free parameters in our algorithm on the user's QoE and provide a suggestion for further improvements. Finally, we evaluate existing algorithms under the constraint of the minimum buffering size. It shows that low-latency streaming benefits from the stabilization of buffer dynamics, and that our algorithm achieves the best QoE performance in all cases and no playback stalls with a buffering size of two more segment durations than the theoretical minimum.

By a direct comparison of existing algorithms to the optimal bitrate selection, we observe that existing algorithms achieve only less than 71% of the optimal QoE on average. This suggests that there is still room for the improvement, and therefore motivates us to present a novel streaming architecture of bitrate adaptation in the next chapter.

5

ADVANCED IMPROVEMENTS IN LOW-LATENCY ADAPTIVE VIDEO STREAMING

The previous chapter shows that state-of-the-art bitrate adaptation is not suitable for low-latency adaptive video streaming due to a lack of explicit stabilization of client buffer dynamics, and therefore introduces an adaptation algorithm (Buffer Dynamics Stabilizer, BDS) that minimizes buffer deviations by quality-optimized bitrate selection. The performance results indicate that there is still room for improvements. To this end, we present further improvements for low-latency adaptive streaming in practice. The main contribution of this chapter is the design of a server-side architecture for adaptive streaming, which provides a low-delay feedback for the video bitrate selection and allows a hybrid adaptation algorithm based on throughput and buffer information even at the server side. The design of the server-side architecture is motivated to minimize the reception delay, which has a direct impact on client buffer dynamics as stated in Chapter 3 and Chapter 4. The proposed architecture is flexible to operate with different transport-layer protocols, such that a streaming-optimized transport protocol other than TCP can be applied to promote the improvements. The improvements are evaluated with respect to user-perceived video quality.

5.1 INTRODUCTION

Most adaptation controllers are deployed at the client side. The main reason is that a client-side controller is close to users and therefore flexible to retrieve a wide set of user local information for the adaptation: e.g., regarding to client buffer occupancy, available network throughput, CPU load, etc. Although the decision at the client may be convenient for a bitrate selection at the segment request time, the delay between client and server due to the request-response mechanism may make the selection not viable anymore at the response time of the requested segment. Such a client-side feedback introduces a significant delay into the bitrate adaptation and leads to a suboptimal bitrate selection, especially in quickly changing scenarios like wireless and mobile networks. Even worse, the feedback delay incurs unnecessary reception delay of segments, as explained in in Chapter 3 and Chapter 4. According to Equation 3.15 and Equation 4.1, the reception delay results in additional decrease of client buffer occupancy and then potential playback stalls due to a buffer underflow. This reduction of buffer occupancy is especially critical in low-latency streaming. Even though a pre-request scheme can be employed, which intends to alleviate the impact of the feedback delay, its efficiency strongly relies on the accuracy of the estimation of the pre-request time. However, the estimation is often inaccu-

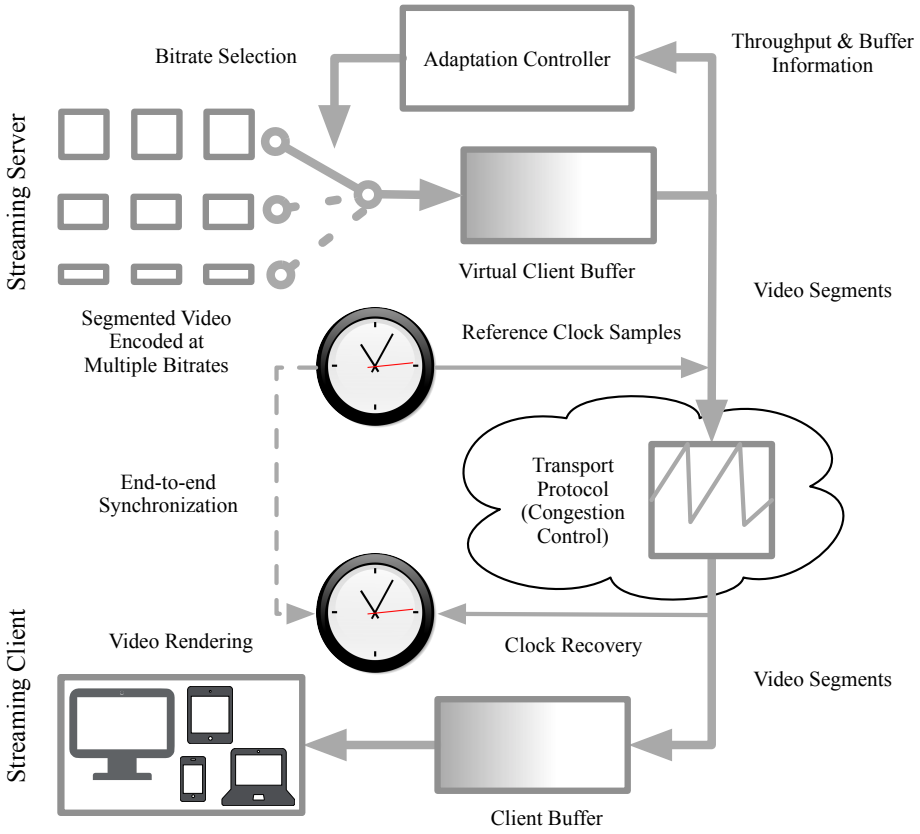


Figure 5.1: Open-loop rate control architecture

rate, because it is hard to accurately predict the future network states. Accordingly, the effectiveness of pre-requesting segments for a client-side adaptation is limited in terms of the delay minimization. As a result of these inefficiencies, client-side adaptation architecture becomes one of the main limiting factors for the deployment of a high-quality video streaming, especially for a low-latency live content.

In contrast to existing adaptation architectures, we propose a novel server-side architecture to adaptive streaming that effectively supports low-latency streaming applications with a buffering size as small as the segment duration of the adaptation set. Since our adaptation controller takes place at the server so that the control loop is not affected by delays and does not require explicit feedback from the client, we name the architecture as *Open Loop rAte Control* (OLAC).

5.2 OPEN LOOP DYNAMIC RATE CONTROL

The stabilization of client buffer dynamics is essential to low-latency adaptive video streaming, as discussed in Chapter 4. Hence, the main design goal of OLAC is to maintain a stable buffer level for the client playback buffer as small as the segment duration. To achieve this goal, OLAC implements the adaption controller at the server side (Figure 5.1), in contrast to most streaming solutions that adopt a client-side adaptation controller. With this architecture modification, we eliminate the feedback delay from clients in the control loop of bitrate adaptation and minimize the reception delay of video segments, which allows OLAC to increase the

accuracy of throughput estimation and bitrate adaptation. In addition, the adaptation algorithms in OLAC are specifically designed to maintain a stable buffer in low delay streaming. To this end, the OLAC adaptation controller is based on both the throughput and the buffer information.

Our proposed OLAC approach has four components as shown in Figure 5.1. First, the virtual client buffer at the server avoids delayed feedback from clients. Second, an adaptation controller based on both the throughput and the buffer information is implemented in order to maximize user's QoE. Third, an open-loop synchronization scheme eliminates the drift between the free-running server and client clocks. Last, the underlying transport mechanism of the OLAC architecture is designed to be replaceable, thereof enabling greater performance improvements by using different emerging transport protocols.

5.2.1 *Virtual Client Buffer*

To avoid the delayed feedback from the client, our *virtual buffer* simulates the client buffer level (in seconds) at the streaming server as follows:

$$b(t) = \sum_{v \in \mathcal{V}_t} \frac{S(v)}{r_{I(v)}^{R_{I(v)}}} - (t - t_m^e)_+ \ , \tag{5.1}$$

where t is the time elapsed since the first video packet was sent (excluding the duration of playback stalls), t_m^e is the time at which the server finishes sending m video segments, $\mathcal{V}_t$ is the set of all video packets sent within the duration of t, $S(v)$ is the size of a video packet v (in bytes), $I(v)$ is the video segment to which the video packet v corresponds, $R_{I(v)}$ is the nominal bitrate of segment $I(v)$, and $r_{I(v)}^{R_{I(v)}}$ represents the average bitrate of video segment $I(v)$ encoded at a nominal bitrate $R_{I(v)}$. For example, by putting $i = I(v)$, we define that a video packet v corresponds to segment i, has a nominal video bitrate R_i, and $r_i^{R_i}$ is the actual average bitrate of segment i when the segment is encoded at a nominal bitrate R_i. Equation 5.1 assumes that the client starts the playback once it buffers m segments; i.e., the buffering size is $B^d = m \cdot T_c$. Note that t excludes the duration of playback stalls, which ensures that $b(t)$ is non-negative. Therefore, given the average bitrate of a segment, we translate the transmitted data volume into its corresponding video duration and calculate the difference between the video duration of sent video data and the actual elapsed time of sending those video data excluding the buffering.

5.2.2 *Adaptation Controller*

An adaptation controller performs bitrate selections for the adaptation based on its algorithm logic. Our design goal focuses on the low-latency video streaming with a buffer size as small as the segment duration. In order to make the optimal choice when a video bitrate has to be selected, the controller needs to be able to predict the trend of the buffer occupancy as accurately as possible, to avoid buffer underflows and overflows. Therefore, it is important for the adaptation algorithm inside the controller to take both buffer and throughput information into consideration when making decisions for the bitrate selection. Thanks to the server-side simulation

of the client buffer, the adaptation controller in OLAC is flexible to employ different algorithms based on the buffer and throughput information even at the server. Moreover, such a hybrid adaptation controller needs to stabilize the adaptive response to dynamics of transport and application layers. To this end, we apply the adaptation algorithm (BDS) presented in Chapter 4 into our OLAC controller. BDS performs the bitrate selection for the next segment once the delivery of a segment is completed at the server side, and effectively maintains a stable buffer at a level as small as the segment duration. It benefits from the OLAC architecture, which provides a low delay of bitrate adaptation, and an improvement of its effectiveness is evidenced in Section 5.4.4.

5.2.3 *End-to-end Synchronization*

Live streaming services require each single client to consume video frames at exactly the same rate they are produced at the source; i. e., capturing and rendering of the video content must be *generator locked* between source and sink [77]. Otherwise, the clock drift between the server and the client in a long-term streaming session causes a constant increase or decrease of buffered data at the client. In addition, the clock drift leads to a divergence of the virtual client buffer from the actual client buffer in our OLAC system.

Conventional HTTP-based streaming compensates for the clock drift between the server and the client by lowering the sending rate via TCP flow-control mechanism and by receiving more video data from the server in advance. A constant increase of buffer occupancy will finally trigger TCP flow-control mechanism, such that the server throttles the arrival rate of video data at the client. In the case of a constant buffer decrease, the client may request to receive more data in advance, so as to keep the buffer away from the empty. However, neither are applicable in live services: lowering the sending rate delays the arrival of video data and thus increases the streaming latency; subject to the availability of video segments, receiving more video data in advance is infeasible. Instead, we apply the concept of the Program Clock Reference (PCR) in MPEG (Moving Picture Experts Group) systems [77], where the reference clock samples are injected into the packet stream on the server and enable the client to perform the recovery with an adjustable virtual clock via a standard Phase Locked Loop (PLL). Since our method of the clock synchronization is independent to TCP flow-control mechanism, the OLAC architecture is flexible to employ other transport-layer protocols than TCP. In the sequel, we focus on a novel streaming protocol which is designed to support real-time video delivery.

5.3 REAL-TIME TRANSPORT

HTTP is commonly considered a poor fit to low-latency Internet communications, since error and congestion controls of the underlying TCP interfere with the timeliness requirements of such applications. Thanks to the flexibility of the protocol configuration, OLAC allows an alternative transport paradigm beneath the streaming architecture, in order to improve the streaming performance by replacing HTTP/TCP. In the following, we present dynamic adaptive streaming architecture over an emerging transport protocol.

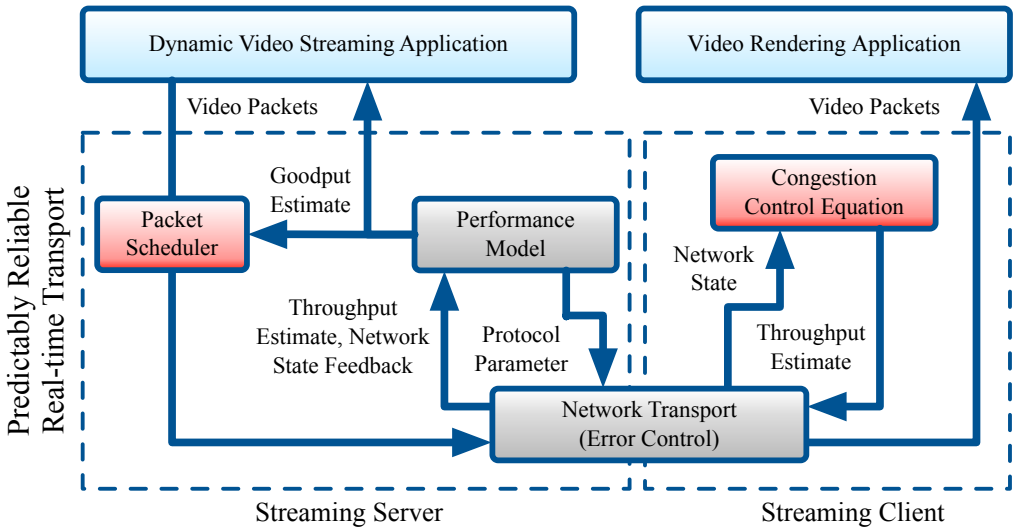


Figure 5.2: Dynamic streaming architecture based on the PRRT protocol

5.3.1 *Predictable Reliability*

Real-time video streaming is subject to specific delay constraints in order to limit the buffering at the receiver and preserve interactivity. Gorius [38] addresses this requirement and introduces a streaming-specific transport protocol, named Predictably Reliable Real-time Transport (PRRT). PRRT specifies a delivery time budget for each packet and must finalize the error control operations for it within this time budget. If this is impossible, the application experiences residual packet loss. The delay-constrained error control is based on a packet-level, *adaptive Hybrid Error Coding* (HEC) method that achieves *capacity-approaching error control* on bidirectional packet-erasure channels [139]. The scheme can be understood as a variant of Type-II Hybrid-ARQ coding [140], which has been found to be optimal in case the capacity of a packet erasure channel is unknown or dynamic.

PRRT's transport paradigm is *predictable reliability*. The protocol includes a stochastic performance model (Figure 5.2) that allows for instant optimization of the protocol configuration under consideration of the transport-layer's periodic network state feedback. Specifically, this optimization process adjusts the error control so as to fulfill the reliability requirement of the application under the given delay constraint with a high probability. Besides adapting the protocol configuration, the performance model calculates the maximum goodput under the throughput estimate obtained from the congestion control equation at the client. The goodput calculation considers the protocol overhead in terms of packet headers and repair packets for the error control.

In PRRT's paradigm, as shown in Figure 5.2, the server runs a packet scheduler that feeds video packets at a rate of less than or equal to the received goodput estimate into the network. Under selection of the suitable bitrate of the dynamic video source, this ensures continuous real-time transmission of the video stream. The throughput estimate is obtained at the client based on the network state of the network path (e.g., RTT and packet loss information). Periodic feedback updates the throughput estimate at the server. The server determines the throughput overhead

that is required for the error control and obtains the resulting goodput via the protocol's performance model.

5.3.2 *Delay-based congestion control*

Loss-based congestion control interprets all packet losses as congestion events, thus is prone to errors in present of packet-loss channels, e.g., wireless and mobile network. As a result, most TCP flavors significantly underutilize the available network throughput in wireless and mobile scenarios. Since delay-based congestion control evaluates the queuing delay of the network to obtain a congestion signal, it is not affected by additional packet losses other than congestion losses. Moreover, delay-based control schemes maintain a stable queuing on the network over a long time such that they increase the throughput and reduce delay jitter for the long term. Both are desired properties of streaming-friendly congestion control. Therefore, we employ delay-based congestion control into the PRRT framework.

In order to obtain smooth and tunable congestion control, we modify the delay-based control equation from Fast TCP [39], as follows:

$$w = \min\left\{2 \cdot w',\ (1-\gamma)\cdot w' + \gamma \cdot \left(\eta \cdot \frac{baseRTT}{avgRTT} \cdot w' + \alpha \cdot (1-\eta)\right)\right\}, \tag{5.2}$$

where $\alpha > 0$, $\gamma \in (0, 1]$, and $\eta \in (0, 1]$. w and w' are the most recent and the previous estimate of the congestion window, respectively. $baseRTT$ represents the minimum RTT observed so far, and $avgRTT$ is the average RTT using Exponentially Weighted Moving Average (EWMA). The control equation itself performs EWMA with parameter γ over the current and the previous estimate of the congestion window. α reflects the number of packets buffered in the network, which should not exceed the aggregate queue size of the network. Both parameters α and γ originate from the Fast TCP equation. We introduce η to provide control over the protocol's aggressiveness in the throughput acquisition, as a smaller value for η attenuates the congestion signal [141].

Under the modified delay-based congestion control, PRRT improves the throughput utilization of continuous packet streams by providing a stable estimate of the available throughput and realizes *opportunistic TCP-friendliness* [142].

Our OLAC architecture benefits from PRRT in that PRRT provides the adaptation controller with predictable packet delay variations, smooth, explicit goodput estimates, and a high throughput utilization.

5.4 PERFORMANCE EVALUATION

For the quality evaluation of OLAC, we perform experiments in scenarios of wireless home networks with an emulated broadband access. We evaluate the performance of the architecture with respect to impairment functions that model user-perceived video quality, and compare it against well-known streaming architectures.

5.4.1 *Methodology*

Our experiments on the performance evaluation of video streaming are more realistic and challenging in this chapter. We run the experiments in a wireless network with an emulated broadband access and evaluate the performance in terms of user-experience metrics. We repeat each test ten times and plot both the averages and the distributions.

Performance metrics. In this thesis, we focus on the industry-standard video quality metrics explored in [124], which are also acknowledged in the research community [55, 97]:

- *Startup delay (join time)*, measured in seconds, represents the duration it lasts from a request of video playback being initiated till the start of the playback.
- *Rebuffering ratio* captures the rebuffering duration during a streaming session, and is defined as the ratio of the total time all rebuffering events take to the duration of the session.
- *Rate of rebuffering* reflects the frequency of rebuffering events occurred during a streaming session, and is computed as the ratio of the number of rebuffering events to the duration of the session.
- *Average bitrate (video quality)* is the mean of nominal video bitrates selected during an adaptive streaming session, since the selected bitrate during the session is changing between different bitrate streams. It is measured in bits per second (e.g., *kbps* or *Mbps*).
- *Rate of bitrate switching (quality variants)* indicates the frequency of bitrate switching during a streaming session, and is calculated as the ratio of the number of bitrate switching events to the duration of the session.

There are three important factors that impact user-perceived video quality in adaptive video streaming: *startup delay*, *stalls*, and *quality variation*. Liu *et al.* [117] derive a set of impairment functions to quantitatively measure user experience. These impairment functions are defined as follows.

- *Impairment of startup delay* is defined as

$$I_{SD} = \min\{3.2 \cdot T_S, 100\} \,, \tag{5.3}$$

  where T_S is the duration of startup delay in seconds and the coefficients are computed by linear regression between the startup delay and the average subjective impairment.

- *Impairment of stalls (rebuffering)* is defined as

$$I_{ST} = 3.35 \cdot T_{ST} + 3.98 \cdot N_{ST} - 2.5 \cdot \sqrt{T_{ST} \cdot N_{ST}} \,, \tag{5.4}$$

  where T_{ST} indicates the total duration of playback stalls and N_{ST} stands for the number of stalls. This impairment function I_{ST} models the user experience affected by the duration and the number of stalls. The third term in I_{ST} is used to compensate the simultaneous effects of stalls duration and stalls number;

i.e., the impairment value does not increase monotonically with stalls number, if stalls duration is fixed [117]. Note that, compared to the original function defined in [117], Equation 5.4 does not take into account the additive term which affects I_{ST} due to the motion information of the video. The omission does not affect the performance comparison, because only one video sequence is used here for the comparison and the term is determined by the average magnitude of motion vectors over the whole video.

- *Impairment of quality variation (level variation)* is given as

$$I_{LV} = 73.6 \cdot P_1 + 1608 \cdot P_2 \,, \tag{5.5}$$

where

$$P_1 = \frac{1}{N} \sum_{i=1}^{N} M_i \cdot e^{0.02 \cdot T_c \cdot N_i} \tag{5.6}$$

models the impairment caused by selecting a particular video bitrate with the video quality metric value M_i, N is the number of transmitted segments during the streaming session, M_i is the VQM value [109] of segment i, T_c is the segment duration, and N_i is the number of consecutive segments preceding segment i that have the same bitrate as segment i. In turn, P_2 models the impairment caused by bitrate fluctuations:

$$P_2 = \frac{1}{N} \sum_{i=1}^{N} |M_i - M_{i+1}|^2 \cdot sign(M_{i+1} - M_i) \,, \tag{5.7}$$

$$sign(x) = \begin{cases} 1 & x > 0 \\ 0 & otherwise \end{cases} . \tag{5.8}$$

Namely, the impairment of quality variation I_{LV} spans all three dimensions of video quality: i.) average video quality covered by P_1; ii.) number of quality switches covered by P_2; iii.) average magnitude of switches covered by P_2.

All the coefficients in the above functions are applied according to their definition in [117]. We use VQM software that is freely available[1]. The value is between 0 and 1, and a lower VQM value implies better quality and less annoyance.

Network profiles. We experimentally evaluate the performance of the proposed solution in scenarios with competing greedy TCP traffic and multiple streaming sessions in an IEEE 802.11n wireless network. In order to emulate a wired broadband access network, between the streaming server and the Access Point (AP) we introduce a bottleneck of 16 *Mbps* with 40 *ms* RTT and a queue size of 80 *KBytes* via the Dummynet[2] network emulator. The wireless receiver is positioned at a distance of 5 *m* from the AP with a wall in the direct signal path. Figure 5.3 illustrates the experimental setup. To demonstrate the fairness and the response to competing traffic, we take into account the following two scenarios.

- Competing greedy TCP traffic: each streaming session runs for 180 *s*. After 60 *s*, we inject one bulk TCP flow with a duration of 60 *s*, using Iperf[3].

1 http://www.its.bldrdoc.gov/resources/video-quality-research/software.aspx
2 http://info.iet.unipi.it/~luigi/dummynet/
3 http://iperf.fr

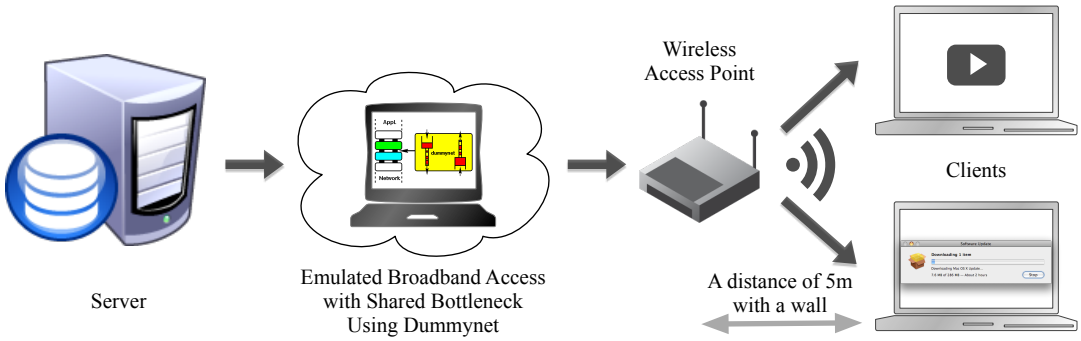


Figure 5.3: An overview of the experimental setup

- Competing multiple streaming sessions: three concurrent streaming sessions simultaneously run for 120 *s*, each corresponding to one server and one client.

Video profiles. We use a high definition video sequence *Big Buck Bunny* with a resolution of 1920 × 1080 pixels and various segment durations of 2 *s*, 4 *s*, 6 *s*, and 8 *s*. The adaptation set includes bit rates between 1 *Mbps* and 16 *Mbps* under VBR encoding. Between 1 *Mbps* and 6 *Mbps*, the bitrate increases in steps of 1 *Mbps*, and above 6 *Mbps* the step size is 2 *Mbps*. We apply MPEG Transport Stream (MPEG-TS) as the video stream format, in order to compensate for the clock drift between the server and the client in case of a long streaming session. By inserting PCRs in the packet stream, we enable the client to synchronize its clock to the server clock by means of these clock samples and a PLL, and thus avoid the drift.

Algorithms. For the quality evaluation of OLAC, our benchmark bitrate adaptation are DASH VLC plugin [78] and Quality Adaptation Controller (QAC) for adaptive video streaming [81].

- DASH VLC plugin, as a reference implementation of DASH system architecture [78], is freely available[4]. The reference implementation uses the rate-based adaptation algorithm (RB) in its adaptation-logic component, as described in Section 4.4.1; i.e., DASH VLC plugin chooses the highest bitrate smaller than the long-term average of all throughput measured samples. Since DASH VLC plugin is a reference implementation of DASH standard, we refer to it as DASH in our evaluation for brevity.

- QAC selects the video bitrate to match the available throughput based on feedback control theory. The adaptation controller of QAC is also deployed at the server so that the control loop is not affected by feedback delays. To the best of our knowledge, a reference implementation is not available. To this end, we have developed our own implementation of QAC following the description in the research paper [81].

To demonstrate the effectiveness of OLAC in practice, we implement its architecture and algorithms in a Linux-based streaming prototype. We employ the basic version of BDS (i.e., BDS-0) as the adaptation algorithm of the OLAC controller, in order to highlight the performance improvements of OLAC. According to the results in Section 4.4, we believe that using BDS-1 in OLAC further enhances the performance.

4 http://www-itec.uni-klu.ac.at/dash/?page_id=10

The OLAC streaming is flexible in that it can be configured to operate with different transport-layer protocols. In this thesis, we present two such configurations:

- First, OLAC is configured to operate over standard TCP-Cubic[5] available in Linux systems. An OLAC implementation over TCP is immediately deployable and allows a direct comparison to state-of-the-art bitrate selection methods, which commonly use standard TCP as their transport layer. For brevity, we refer to this OLAC configuration as Dynamic Adaptive Streaming over TCP (DAST).
- Second, we configure OLAC to use PRRT as its transport layer [139]. PRRT implements an efficient, predictably reliable error control for optimized utilization of the available throughput on wireless Internet paths. By incorporating the delay-based congestion control, PRRT achieves opportunistic TCP-friendliness. We refer to the OLAC-over-PRRT configuration as Dynamic Adaptive Streaming over PRRT (DASP).

Therefore, our performance comparison contains four sets of performance results: DASH (VLC plugin), QAC, DAST (OLAC over TCP), and DASP (OLAC over PRRT).

We specifically focus on low-latency streaming scenarios with small client buffer sizes β_{max} of $2\,s$, $4\,s$, $6\,s$, and $8\,s$. To demonstrate the performance of OLAC with a latency as small as the segment duration, we use the segment durations T_c of $2\,s$, $4\,s$, $6\,s$, and $8\,s$ (corresponding to client buffer sizes $\beta_{max} = T_c$) and the buffering size $B^d = m \cdot T_c$ with $m = 1$.

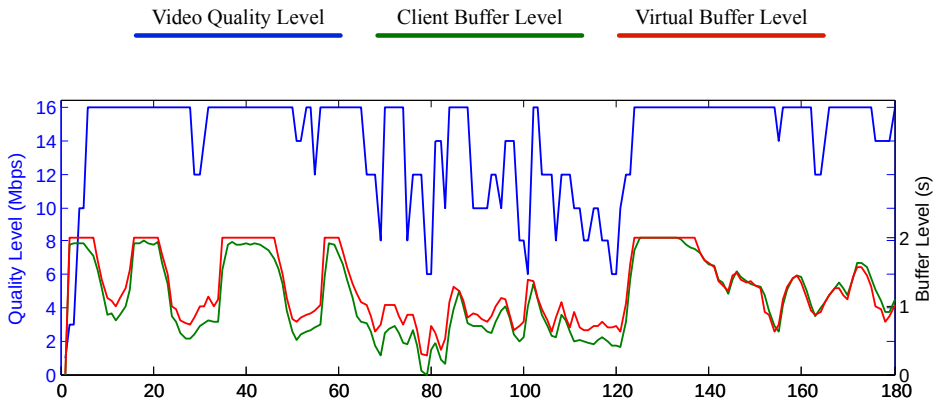
5.4.2 *Mirror the Client Buffer Dynamics*

The key component of the OLAC architecture is the virtual client buffer, which simulates client buffer dynamics at the server. As a result, OLAC enables server-side bitrate adaptation based on the buffer and throughput information, and thus the performance increase for bitrate adaptation. The effectiveness of OLAC relies on the simulation accuracy of the virtual client buffer — i.e., whether the virtual client buffer is able to accurately mirror the dynamics of the client buffer. Hence, we first inspect the accuracy of the buffer simulation before we present the results of performance evaluation for OLAC.

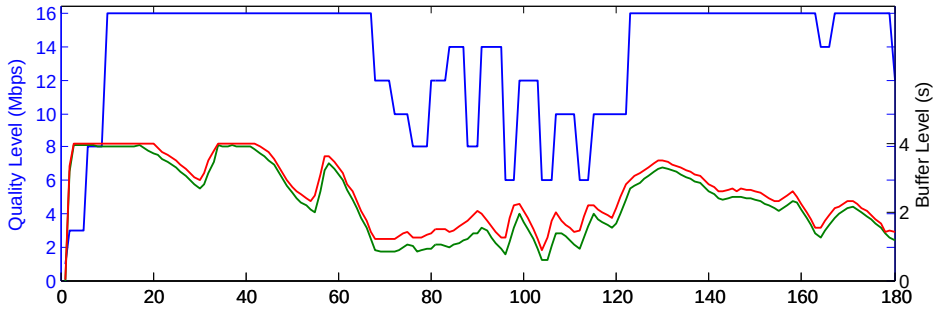
Figure 5.4 and Figure 5.5 show the performance of DAST and DASP when competing with a TCP flow in terms of video quality level and buffer level, respectively.

In general, the virtual buffer level in the DAST configuration tracks the dynamics of the client buffer. However, it is observed that there are obvious skews and deviations between the virtual and actual client buffer levels, when the competing TCP flow is added into the bottleneck network. In consequence, due to misleading buffer information, DAST cannot always pick suitable video bitrates to adapt the dynamics of the network, and then occasionally suffers from critical low buffer occupancies. As shown in Figure 5.4a, the buffer level reaches $0\,s$ at second 79 for the client buffer size of $2\,s$ and at second 85 for the client buffer size of $6\,s$ in Figure 5.4c. The reason is that the delivery of video packets over TCP has no delay constraints

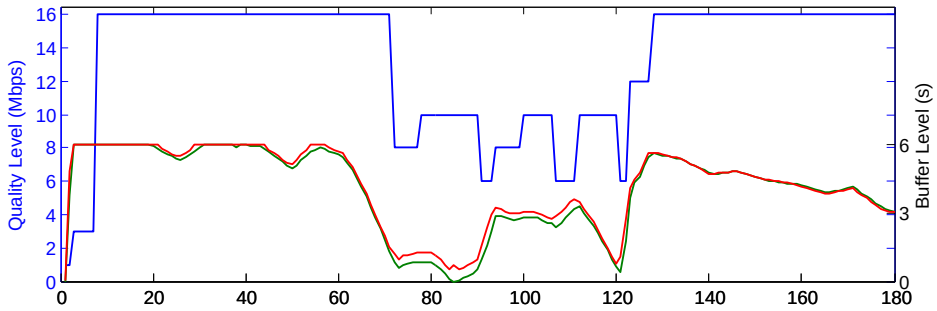
5 All implementations of DASH, QAC, and DAST used in this thesis are based on TCP-Cubic[143], the default TCP flavor in current Linux and Android systems.



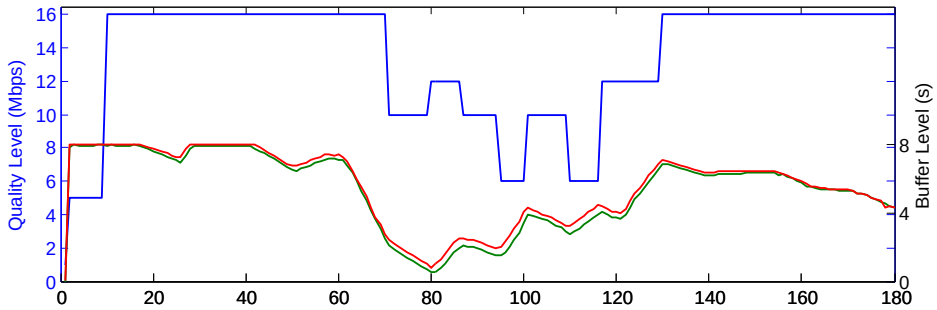
(a) DAST with client buffer size 2 *s*



(b) DAST with client buffer size 4 *s*



(c) DAST with client buffer size 6 *s*



(d) DAST with client buffer size 8 *s*

Figure 5.4: Video quality level and buffer level of DAST when competing with a single TCP flow. The segment durations include 2 *s*, 4 *s*, 6 *s* and 8 *s*.

and experiences packet jitter (packet delay variation). Namely, TCP packet jitter introduces a mismatch between the set of packets used for buffer simulation at the sender ($\mathcal{V}_t$ in Equation 5.1) and the set of packets that actually arrive at the client. In contrast, PRRT ensures that per-packet arrival deadlines are met, therefore the virtual buffer level of DASP accurately tracks client buffer dynamics. As shown in Figure 5.5, the virtual buffer level is almost in line with the actual client buffer level; in other words, the virtual buffer is like a mirror of the client buffer dynamics in our DASP configuration.

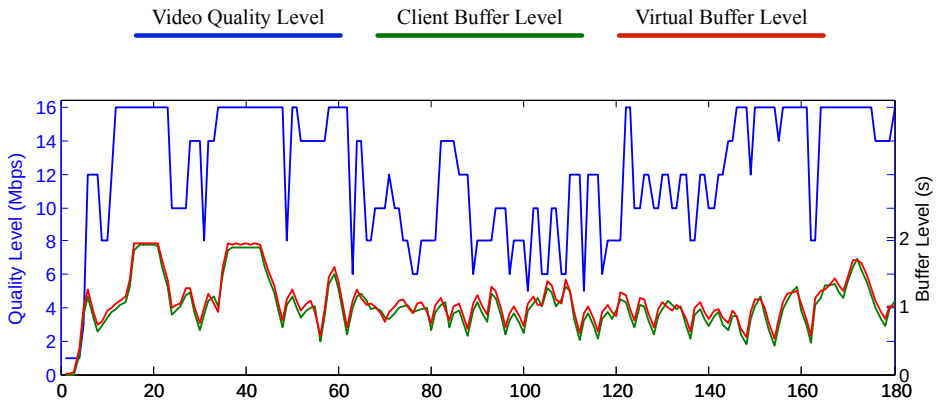
As a result, bitrate adaptation benefits from the OLAC architecture. From Figure 5.4 and Figure 5.5, we see that the video stream with both configurations is continuously delivered at a high video bitrate and our bitrate adaptation reacts smoothly to the injection of competing traffic by converging to an equal share of the bottleneck throughput. After removal of the competing flow, the video stream completely re-acquires the available throughput. The buffer level converges towards the desired buffer level. These results confirm the efficiency of our OLAC control scheme: the converging buffer level indicates that the application layer absorbs the dynamics of transport layer with precise bitrate selections.

In summary, DASP accurately mirrors the dynamics of the client buffer thanks to the timely arrival of packets provided by PRRT. In contrast, due to packet jitter, DAST cannot track the client buffer dynamics as well as DASP.

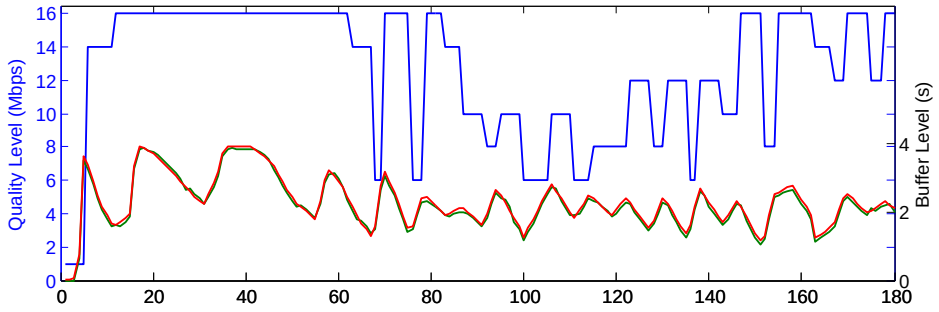
5.4.3 *Performance Comparison*

Single Competing TCP Session. Figure 5.6 shows the performance in terms of the impairment functions when each of four approaches (DASH, QAC, DAST, DASP) competes with a single TCP flow. With respect to the startup delay, DAST achieves the lowest values on average, as shown in Figure 5.6a. Specifically, the startup delays in DAST are 0.3 s, 0.6 s, 0.8 s, and 1.0 s on average in the case of segment durations 2 s, 4 s, 6 s, and 8 s, respectively. Correspondingly, the average impairments of startup delay for DAST are 0.5, 1.3, 1.7, and 2.0, respectively. QAC has a delay approximately equal to the segment duration, because QAC server feeds video data into the sender buffer at a rate equal to the video playback rate. Therefore, it has the largest impairment of startup delay. In the case of DASP, the startup delay is larger than that of DASH or DAST, since PRRT has a slower sending rate than TCP in the beginning of a session and a smoother throughput variation. Specifically, the impairment for DASP is $3.1 - 6.0$ higher than that for DAST on average.

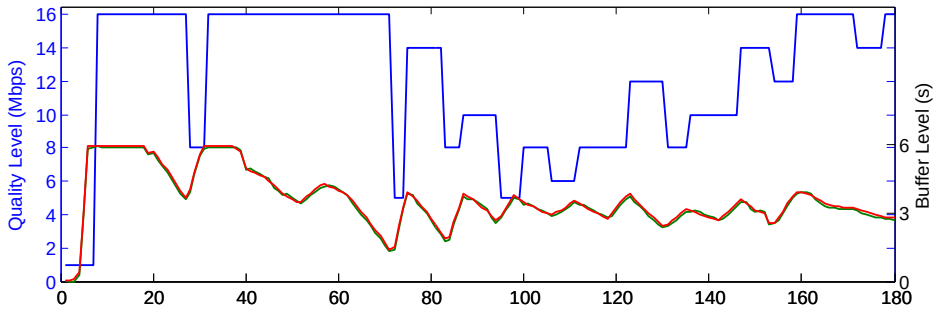
Figure 5.6b and Figure 5.6c show the performance in terms of the impairment of stalls and quality variation, respectively. Overall, DAST and DASP obtain the best performance. First, compared to DASH and QAC, OLAC reduces the impairment of stalls by at least 80%. In fact, our DASP sessions had no stalls. The reason is that OLAC effectively controls the buffer level and PRRT provides an accurate throughput information. Second, compared to DASH and QAC, DAST and DASP reduce the impairment of quality variation by at least 26% and 20%, respectively. Third, the variance of impairment values in OLAC is smaller than for QAC and DASH in most cases. This shows that OLAC has a more stable performance than the other approaches.



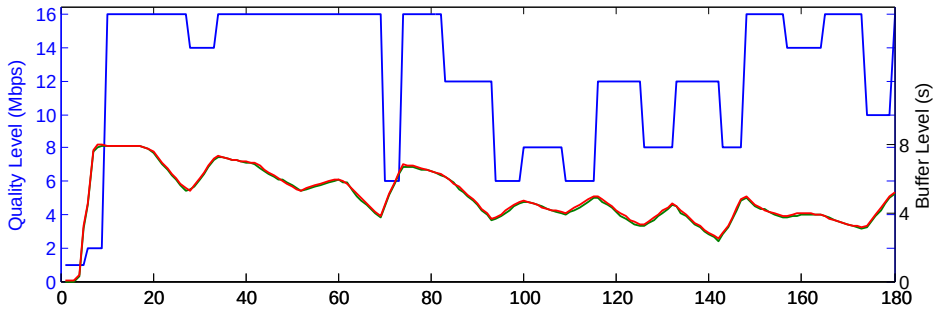
(a) DASP with client buffer size 2 *s*



(b) DASP with client buffer size 4 *s*

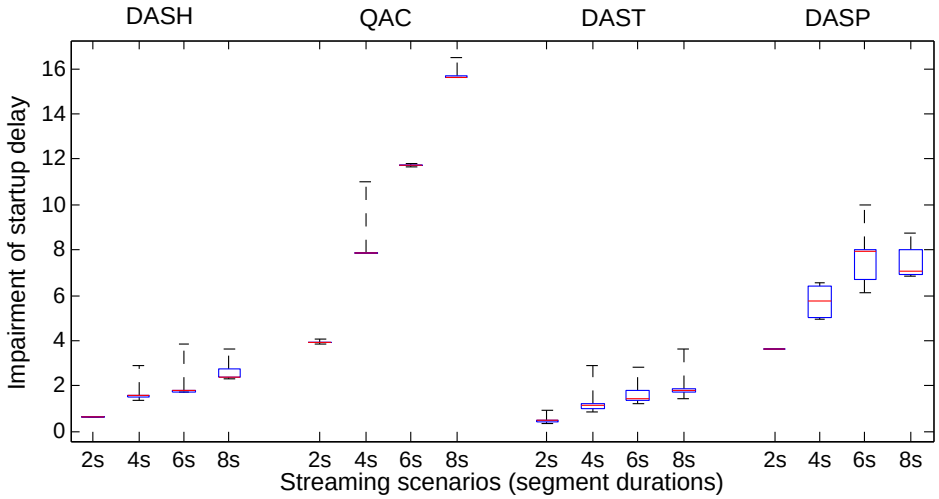


(c) DASP with client buffer size 6 *s*

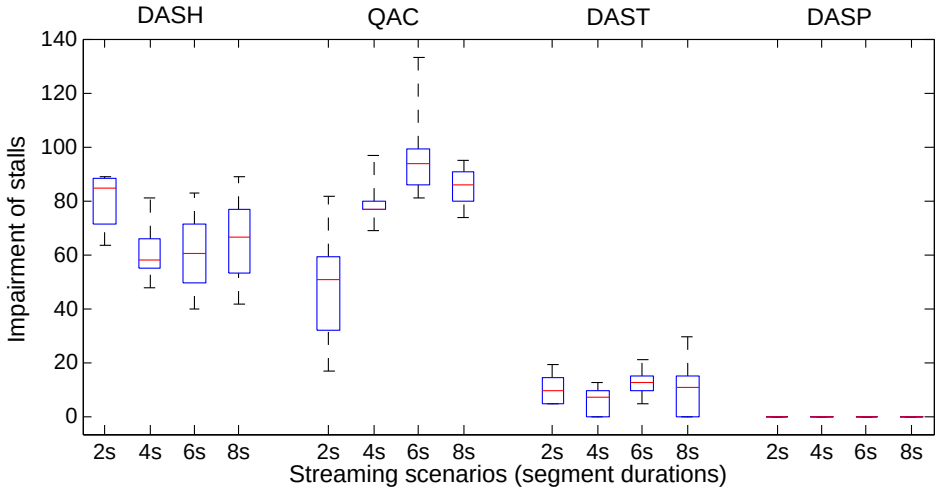


(d) DASP with client buffer size 8 *s*

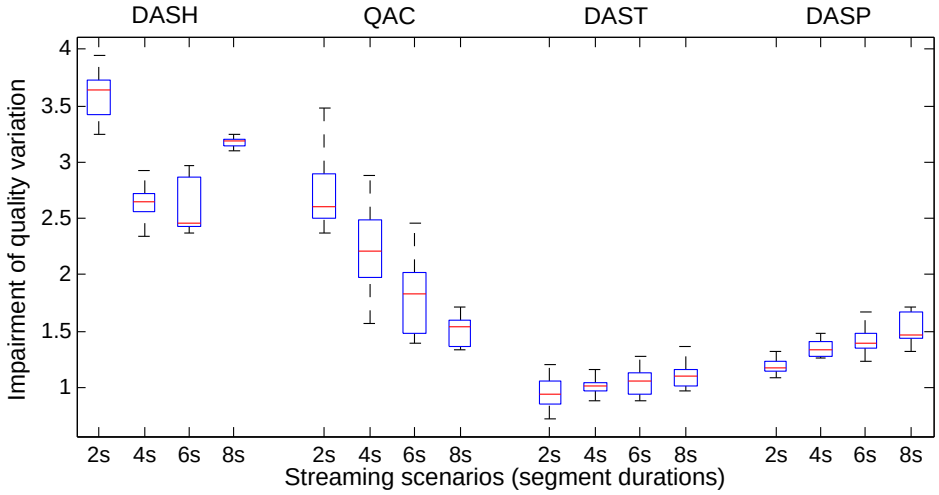
Figure 5.5: Video quality level and buffer level of DASP when competing with a single TCP flow. The segment durations include 2 *s*, 4 *s*, 6 *s* and 8 *s*.



(a) Impairments of startup delay



(b) Impairment of stalls



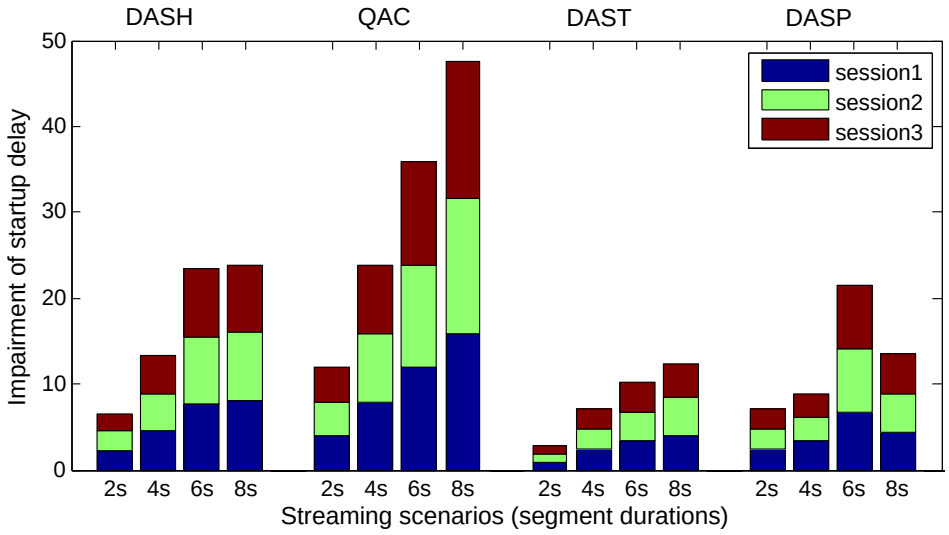
(c) Impairment of quality variation

Figure 5.6: User experience modeled by impairment functions for single competing TCP traffic in ten runs of each streaming scenario. The segment durations include $2\,s$, $4\,s$, $6\,s$ and $8\,s$. The 5-th and 95-th percentiles are marked with black bars. The blue box contains the impairment values between 25-th and 75-th percentile. The red bar in the center indicates the mean of impairment values.

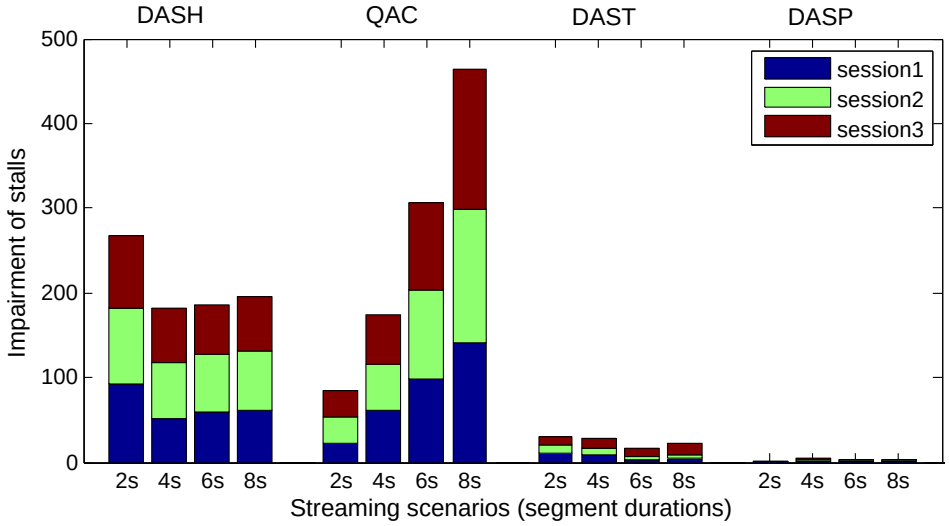
Multiple Streaming Sessions. Figure 5.7 shows that all four tested approaches ensure fairness in the sense of enabling multiple competing video sessions to achieve an equal user-perceived quality. OLAC achieves the best performance. First, the total impairment of stalls of three DAST streams is reduced by at least 85% and 64%, compared to DASH and QAC, respectively. In the case of DASP, only very few stalls were detected, thus it reaches a reduction of at least 98%. Second, the total impairment of quality variation of three DAST streams is reduced by at least 26% and 37% compared to DASH and QAC, respectively. In the case of DASP, a reduction of at least 26% and 41% is achieved. For the impairment of initial delay, we obtain similar results as in scenarios of competing a single TCP session. The only difference is that the average startup delays in DASH and DAST increase by 0.7–2.9 s and 0.1–1.0 s, because the available throughput for each stream is reduced due to the competition among streams.

Results analysis. The reason for superior performance of OLAC in comparison with DASH and QAC is as follows. OLAC improves the accuracy of bitrate selection by minimizing feedback delays of the adaptation compared to other approaches. The effectiveness of OLAC with respect to the feedback delay will be detailed in Section 5.4.4. OLAC takes into account the buffer and throughput information while performing bitrate selections. In contrast, the bitrate adaptation algorithms of DASH and QAC consider either buffer information or throughput information, but do not consider them jointly. In addition, the bitrate selection algorithms of DASH and QAC do not stabilize the client buffer dynamics, while BDS selects video bitrates to reduce buffer fluctuations. As a result, the buffer level oscillates significantly and streaming suffers from stalls. Finally, the bitrates in DASH and QAC are often selected such that they do not match the available throughput, which may lead to a low quality and a high quality variation. Specifically, DASH performs its bitrate selection based on the measurement of the long-term average throughput at the client. However, the average throughput does not reflect the instantaneous available throughput. This often leads to significant variations of the buffer level, such that video bitrates frequently switch to the minimum or the maximum available bitrate. Although QAC achieves the smoothest change of bitrates, it has more stalls in the case of large segment durations and a higher impairment of quality variation in the case of segment duration of 2 s. The reason is that QAC reacts slowly in scenarios with a small client buffer. In particular, QAC applies a feedback control loop where the bitrate selection is based on the buffer deviation from the target buffer in the Proportional-Integral (PI) control law. The target buffer level is limited by the maximum client buffer level. In the case of a small target buffer, errors accumulate and cause a slow control behavior. Although QAC allows to change the behavior of the adaptation by tuning the parameters of the PI control law, it can only eliminate stalls if it enforces a low throughput utilization at all times during streaming.

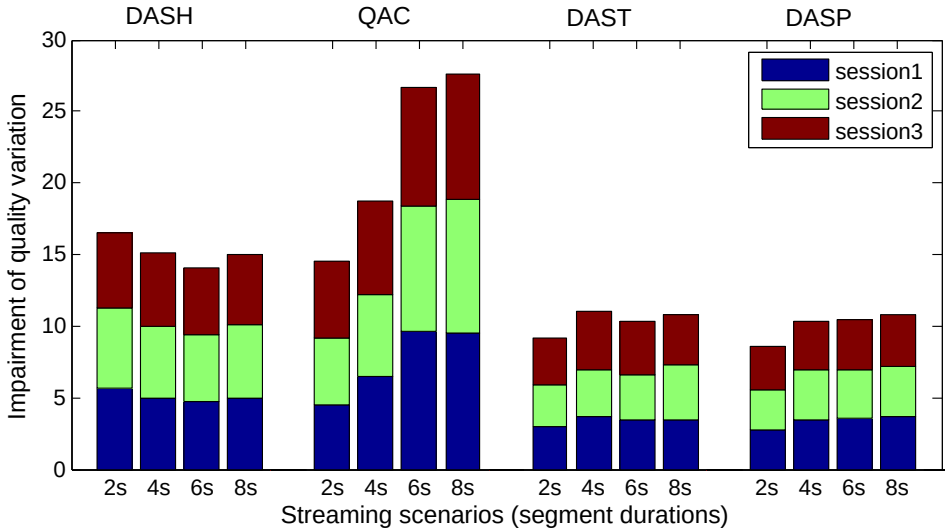
In summary, thanks to the low-delay feedback of OLAC and the buffer stabilization of BDS, our approaches (DAST and DASP) significantly outperform QAC and DASH in terms of the impairment functions. The results show that our approach achieves a very low startup delay, at least 64% lower impairment of stalls and 20% lower impairment of quality variation. Since the inter-percentile range of the results



(a) Impairment of startup delay



(b) Impairment of stalls



(c) Impairment of quality variation

Figure 5.7: Aggregate performance for multiple streaming sessions on average of ten runs of each streaming scenarios. The segment durations include 2 *s*, 4 *s*, 6 *s* and 8 *s*..

(i.e., the difference between 25-th and 75-th percentile) is smaller than for QAC and DASH, we can also conclude that the performance of our approaches is more stable.

5.4.4 *Impact of OLAC*

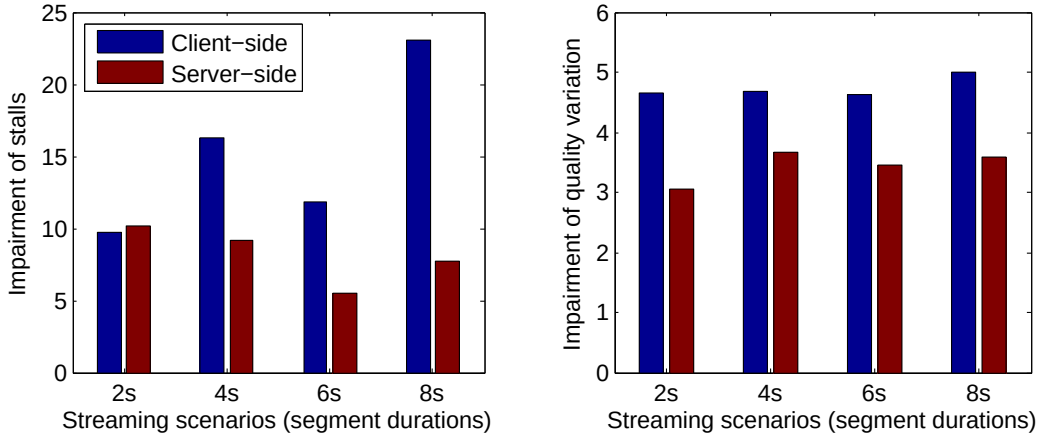
The superior performance of DASP configuration shown in the previous section is contributed by three main components: the adaptation algorithm for buffer stabilization (BDS), the open-loop adaptation architecture with low-delay feedback (OLAC), and the streaming-optimized transport protocol (PRRT). The performance of BDS is presented in Chapter 4. The effectiveness of PRRT in OLAC is also demonstrated in Section 5.4.2 and Section 5.4.3. To evaluate the benefit of OLAC on the proposed bitrate adaptation algorithm, we implement the same algorithm (BDS-0) on the client side and compare this client-side adaptation controller with the server-side adaptation controller (BDS-0 within OLAC). We employ the TCP protocol configuration for these two architectures. In the case of a client-side adaptation, we require an estimation of the request time as discussed in Section 4.2, in order to minimize the reception delay between two consecutive segments (i.e., Δt_i in Equation 4.1). To this end, we estimate the request time based on the measurement of the average request-response delay at the client. In detail, we first measure the request-response delay for each request, i.e., the time elapsed from the moment the client requests a segment until the client starts the reception of the segment. We then estimate the request-response delay d_i^d for segment i as the average of all previous delay samples. Last, the client requests segment i at the moment when d_i^d seconds remain until the end of the reception for segment $i-1$. The end time of the reception for segment $i-1$ is estimated based on the same throughput estimate for segment $i-1$ as the algorithm does.

Figure 5.8 shows that our bitrate adaptation algorithm within OLAC outperforms the client-side adaptation in most of our experimental scenarios. OLAC reduces the impairment of stalls and quality variation by up to 67% and 34%, respectively. The reason is that OLAC can achieve near-zero values for Δt_i by deploying the adaptation controller on the server side. In the case of a segment duration of $2\,s$, OLAC has a higher impairment of stalls than the client-side adaptation. This is due to the fact that virtual client buffer over TCP cannot track the dynamics of the client buffer as shown in Section 5.4.2 because the delivery of video packets over TCP suffers from severe delay variation. We omit the results with respect to the startup delay, since they are very similar here in both scenarios.

According to the results in Figure 5.7 and Figure 5.8, we can also conclude that our bitrate adaptation algorithm within a client-side architecture outperforms DASH and QAC. Specifically, the impairment of stalls reduces by at least 65%, while the impairment of quality variation is reduced by 2–48%.

5.4.5 *Impact of Client Buffer Size*

Finally, we explore the impact of the client buffer size on the OLAC performance in low-latency streaming. In this experiment, we use the segment duration of $2\,s$ as an example, and increase the buffer size in steps of $2\,s$, resulting in buffer sizes of $2\,s$,



(a) Impairment of stalls

(b) Impairment of quality variation

Figure 5.8: Comparison of the client-side and server-side bitrate adaptation with the buffer stabilization over TCP configuration. User experience modeled by impairment functions for multiple streaming sessions on average of ten runs of each streaming scenario. The segment durations include 2 *s*, 4 *s*, 6 *s* and 8 *s*.

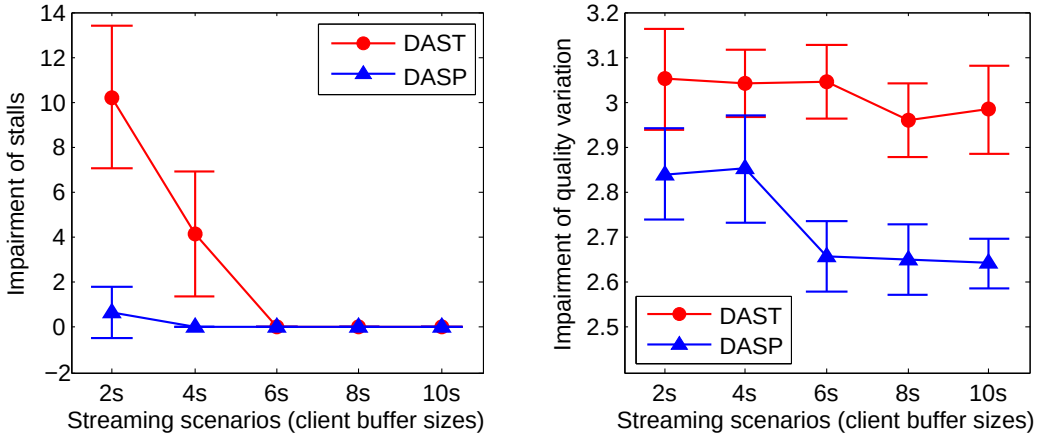
4 *s*, 6 *s*, 8 *s*, and 10 *s*. Figure 5.9 reports the averages and the standard deviations of ten runs of each streaming scenario for multiple streaming sessions.

Figure 5.9a shows that the rebuffering performance of OLAC significantly improves as the buffer size increases. Specifically, by increasing buffer size from 2 *s* to 4 *s*, the impairment of stalls in DAST decreases by 60%, while DAST and DASP achieves no rebuffering events for buffer sizes above 4 *s* and 2 *s*, respectively. Similarly, for each 2 *s*-increase in buffer size, the impairment of quality variation in DAST and DASP reduces by up to 3% and 7% (Figure 5.9b), respectively. Moreover, the deviation results show that OLAC performs stably. With respect to the impairment of startup delay, the results are very similar to the scenarios for multiple streaming session in Section 5.4.3.

5.5 SUMMARY

To further enhance the streaming performance in the context of low latency, we propose OLAC, a novel approach to dynamic streaming that effectively controls a small client buffer required for low-latency streaming services. In order to minimize the reception delay of video segments, the OLAC architecture introduces a virtual client buffer and an advanced adaptation controller that jointly avoid buffer instability at the client and compensate for inaccurate selection of the video bitrate. The adaptation algorithm deployed in OLAC benefits from the low-delay feedback provided by the server-side adaptation controller, thus increases the accuracy of bitrate adaptation. Our approach is designed to be flexible in operation with different transport-layer protocols, so as to enable greater improvement from the underlying transport protocols.

We implement the proposed dynamic streaming architecture and algorithm on top of two transport-layer protocols (TCP and PRRT), and compare their perfor-



(a) Impairment of stalls

(b) Impairment of quality variation

Figure 5.9: User experience modeled by impairment functions for multiple streaming sessions with increasing client buffer sizes. Averages and standard deviations of ten runs of each streaming scenario are indicated.

mance against two state-of-the-art solutions with respect to impairment functions that model user-perceived video quality. We run experiments at a wireless network with an emulated broadband access. Experimental evaluations show that OLAC significantly improves user-perceived video quality in scenarios with single and multiple competing flows. Specifically, we achieve only very few stalls in the PRRT configuration and at least 64% lower impairment of stalls compared to the other two solutions in the TCP configuration. In addition, OLAC achieves at least 20% lower impairment of quality variation for dynamic streaming with buffer sizes as small as the segment duration.

6

RELATED WORK

As Internet-based video streaming dramatically develops, much effort has been put on understanding and improving users' experience of streaming video. This chapter discusses the existing literature on those aspect of adaptive streaming that are related to our work. We begin with a survey of the latency in adaptive streaming, which outlines a landscape of the latency achieved by state-of-the-art bitrate adaptation. We proceed by reviewing the literature on the methods of throughput estimation, based on which most adaptation solutions make their bitrate decisions. We then focus on the studies addressing adaptation algorithms, striving for the optimization of the streaming performance with various approaches. Afterwards, we highlight the research efforts on the potential benefits from transport protocols, followed by video encoding schemes. Finally, we end this chapter with the investigation of a global optimization.

6.1 LATENCY LANDSCAPE IN ADAPTIVE STREAMING

Video delivery over the Internet has evolved rapidly over the past few years. The last decade has seen the transition of video streaming from UDP-based to HTTP/TCP-based technologies. Recently, there has been a significant number of studies on HAS and HAS-based commercial applications, and MPEG-DASH has emerged as a standard for Internet-based video streaming. Most of recent studies and applications focus on streaming solutions that typically consider playback buffer sizes of 10 to 30 seconds or even more. In contrast, the attention to those that specifically target live or low-latency streaming is significantly less.

6.1.1 *Industry*

Commercial video streaming services apply extensive buffering at the streaming client and a conservative bitrate selection, in order to obtain continuous video playback with acceptable throughput utilization [24, 144, 145].

Akhshabi *et al.* [24] experimentally evaluate the bitrate adaptation mechanisms of three HAS clients developed or used by the commercial service providers. The examination focuses on the reaction to persistent and short-term throughput changes, the behaviors of two competing clients under a shared network bottleneck, as well as the latency in the case of live streaming. They observe that the client buffer sizes are approximately $30\,s$ and one could reach $300\,s$ for a $2\,s$ video segment.

Müller *et al.* [144] compare their proposed implementation of MPEG-DASH against three proprietary streaming solutions with respect to the selected bitrates

and the buffer occupancy. The experiments are performed in an emulated mobile 3G environment using throughput traces recorded under vehicular mobility. The results reveal that the solutions employ an at least 30 *s* buffer (i.e., 15 segments) to compensate high throughput fluctuations.

Zabrovskiy *et al.* [145] provide a performance evaluation of eight commercial and open-source HAS players in terms of e.g., average video bitrate, startup delay, rebuffering, and bitrate switches. The players experience rebuffering and/or low average bitrate in a scenario with step-wise and abrupt throughput changes, despite applying buffer sizes in a range of approximately from 12 *s* to 40 *s*. A segment duration of 4 *s* is adopted in their experiments.

6.1.2 *Academia*

While the industry employs conservative adaptation solutions for commercial streaming services, the academia contributes to abundant methods for improving the QoE in adaptive streaming. The great majority of them are based on the consideration that, a client playback buffer is on the order of tens of seconds [25, 146, 26].

Timmerer *et al.* [146] investigate ten adaptation algorithms proposed in the literature. Their objective measurements are conducted within a controlled environment based on a throughput trace, and have two versions of the segment duration (2 *s* and 10 *s*). The buffer sizes of the tested algorithms vary approximately from 30 *s* to 80 *s* and their playback buffers have a mean buffer level of above 15 *s*, except that two methods exhibit a relatively small buffer size of 10 *s* and 12 *s* in the case of the 2 *s* segment duration, but resulting in a high number of rebuffering.

Karagkioules *et al.* [26] address a comparative study of five adaptive streaming algorithms running over various mobile network traces. The case study differentiates between two sizes of the client buffer; 16 *s* (4 segments) and 92 *s* (23 segments) target two application scenarios: live or short clips and VoD or long movies, respectively. Additionally, the buffering size (i.e., the number of segments received in the buffer before the playback can start initially) and the rebuffering size (i.e., the number of segments received in the buffer before the playback can resume after a buffer underflow event) are both set equal to 2 segments for their experiments.

In the context of live content, Thang *et al.* [25] study the behaviors of five typical adaptation methods using two throughput cases: a simulated sudden throughput drop and an actual throughput trace. They show the deficiencies of the five methods when a client can maximally buffer 3, 5, and 10 video segments (corresponding to 6 *s*, 10 *s*, and 20 *s*, respectively).

6.1.3 *Reduced Latency*

As outlined above, numerous studies in industry and academia go to the adaptation solutions that do not address the low-latency requirement. There are only few research directions (e.g., [33, 147, 148, 149, 150]) toward a reduced latency. In the sequel we point out some of them.

Lohmar *et al.* [33] identify and analyze the main delay components of HAS in live services. To be robust towards network throughput fluctuations, they suggest a buffering size of 2 segment durations, although the minimal target is one. Con-

sequently, they estimate a latency for HAS on the order of 5 segment durations. However, the analysis does not explicitly consider the actual characteristics of the network, e.g., available throughput and network delay. Therefore, the result may over- or underestimate the required latency.

Evensen *et al.* [147] explore the benefits of collaboratively using multiple network interfaces in HAS. They introduce an adaptive, segmented video streaming system, which dynamically divides a video segment into smaller logical sub-segments with varying sizes, and dynamically assigns and requests sub-segments to the network interfaces according to the throughput ratio of the network links. The adaptation algorithm proposed in the system selects for the next segment the highest bitrate that can be received before the playback deadline, based on the aggregated throughput and the RTT perceived in the past. The buffer sizes vary from 1 to 6 segments (using 2 s video segment) in experiments, while the buffering size is fixed to 1 segment. The results show that dynamic size-adjusting and request-scheduling of sub-segments are crucial to fully utilize the aggregated link capacity.

Miller *et al.* [150] focus on a HAS-based live streaming which keeps a target latency within 5 s for the video segment duration of 2 s. According to throughput predictions and prediction error estimations, the proposed algorithm computes the probabilities of missing the playback deadline for the individual segments. The highest bitrate for the next segment is identified such that the probability is bound by a controlled threshold. An upward switching of the bitrate is only performed if the fraction of bitrate switches does not exceed a predefined upper bound. The algorithm relies on the short-term TCP throughput predictions on multiple time scales from 1 s to 10 s.

Wei *et al.* [148] demonstrate that the server-push feature of HTTP/2 [151] allows a reduction of the latency in live streaming by employing short segment durations without the HTTP-request explosion issue. They experiment with 1 s segment duration for the push strategy and observe a live latency of more than 3 s. However, the study does not evaluate other QoE metrics than the latency, and the overhead of the video compression due to the small segment. Similar to the idea of reducing the latency, Huysegems *et al.* [149] further explore new HTTP/2 features to improve the live experience in HAS. They use the HTTP/2 features such as stream termination, request/response multiplex, stream prioritization, and server push, to support a reduction in the segment duration to the sub-second level. The evaluation demonstrates that an average latency can reach from 1.05 s to 10.48 s while using five segment durations from 133 ms to 2 s. The encoding overhead of segment duration 133 ms increases by roughly 10% compared to segment duration 2 s. It should be noted that, these effort is orthogonal and complementary to our work.

6.2 THROUGHPUT ESTIMATION

The available network throughput in the future is unknown and variable. A measurement of network throughput often represents the throughput information in the past. An adaptation system requires a technique to estimate the available throughput, such that a suitable video bitrate is selected to adapt the varying throughput.

There exists a large body of literature on throughput measurement and estimation in ABS. The simplest way is to use the measured throughput right after having

fully received a video segment, usually called instant throughput, as the throughput estimate for the next segment. The drawback of using instant throughput as an estimate is that the adaptation suffers from short-term fluctuations. A popular technique to cope with short-term fluctuations is using smoothing methods on measured samples such as averaging [24, 29, 152, 153, 150, 154]. However, this may lead to delayed responses to abrupt throughput drops. In addition, a throughput estimate can be obtained by other means and tools e.g., based on probing [132, 85, 60, 155], machine learning [133, 153, 156], stored data (lookup table) [157], and server support [158].

An average approach is adapted by the dash.js player[1], which uses the mean of last three measured throughput samples as the throughput estimate for next video segment. Akhshabi *et al.* [24] employ an EWMA in their proposed adaptation algorithm. To avoid delayed reaction to significant buffer decreasing incurred by abrupt throughput drops, they use a buffer threshold to decide whether the client should switch to the lowest bitrate. Finding an appropriate trade-off between the responsiveness and smoothness for the averaging is challenging. Sobhani *et al.* [154] take the advantage of the Kaufman's Adaptive Moving Average (KAMA) [159], which dynamically calculates the smoothing factor in EWMA based on the current throughput trend. Jiang *et al.* [29] suggest the harmonic mean for throughput estimation, in order to address the outlier issue in the averaging if the reception of a segment experiences a very high or low throughput. Subsequently, the adaptation benefits from the smoothness of throughput estimation with respect to quality variations.

Riiser *et al.* [157] introduce a throughput-lookup service based on Global Positioning System (GPS), in order to estimate the throughput availability for HAS. The principle of the GPS-based throughput-lookup service is to constantly monitor the throughput and geographical location of streaming clients, and to store this data to a central service. Using such a service allows streaming clients to query for estimated available throughput for given locations based on past observations. The experiments, both simulations and real-world tests, show that the service can be used to predict throughput fluctuations and network outages and is beneficial to avoid buffer underflows in streaming over mobile networks.

Mirza *et al.* [133] propose a machine learning method for the TCP throughput prediction. Tian and Liu [153] adopt this method to predict the achievable throughput at the beginning of a streaming session, since there is no TCP throughput history for the prediction. The method uses the Support Vector Regress (SVR) algorithm [160] to train a TCP throughput model with dataset concerning packet loss rate, queuing delay, file size, and the corresponding actual TCP throughput. To predict the current TCP throughput for ABS, one just feeds the model with the current measured packet loss, delay, and the selected segment size.

Sun *et al.* [156] develop a throughput estimation framework using observed throughput from past video sessions. The framework uses an offline clustering step to identify sessions with similar throughput patterns based on session features such as ISP, region, server identifier, and client's IP address. For each cluster, it estimates the initial throughput (i.e., the available throughput at the beginning of a streaming session) as the median throughput of the sessions in this cluster. To estimate the throughput during a session, it trains a Hidden Markov Model (HMM) for

1 https://github.com/Dash-Industry-Forum/dash.js/wiki.

each cluster to capture the state transitions within the session. The proposed framework is evaluated against five existing estimation approaches in both trace-driven simulation and commercial deployments. The results demonstrate its benefits on the estimation accuracy and the total rebuffering time.

Prasad *et al.* [132] survey four major techniques in available throughput measurement using probes and review the throughput estimation tools that implement these techniques. Probing-based throughput measurement estimates the available throughput by periodically sending probing packets to the network and analyzing the characteristics of the probing gap such as sending rate, delay, and size of probing packets. Such active probing methods may provide more accurate estimation compared to passive methods as described above, but their major drawbacks are two-fold. First, the estimation may not be accurate any more if the probing packets traverse other network paths than the video data. Second, the additional probing packets will cost the network resource. Therefore, passive methods are generally used for throughput estimation in ABS.

To address the issues of active probing throughput measurements, Mok *et al.* [85] propose an estimation method by integrating the probing measurement into the video stream. The method employs a proxy (or the server side) to adjust the sending rate of video packets by varying the inter-departure time of the packets in each probing round. Based on the measurements, the proxy calculates the loss rate, the average RTT, and the variance of RTT, and determines the preferred sending rate by comparing these values with their lower and upper bounds. To achieve a timely estimation, the method probes the network with a set of selected sending rates that correspond with the available video bitrates, instead of probing with all possible sending rates. As all probing methods, such a passive probing method requires additional supports of the server (or a proxy) and/or the client.

In the presence of competing clients, the ON-OFF streaming pattern may interfere the throughput estimation at the application layer, causing inaccurate estimate, as discussed in Section 2.2.3. It is worth noting that Li *et al.* [60] present an alternative probing method for throughput estimation to tackle this problem. Unlike throughput estimation methods that output a direct estimated value for the current available network throughput, the proposed method adopts a probing mechanism similar to TCP congestion control — the Additive-Increase / Multiplicative-Decrease (AIMD) algorithm. Namely, it constantly increments the estimated value by an additive increase term until the measured throughput is smaller than the throughput estimate; in case of an overestimate, it imposes a multiplicative decrease on the estimated value. To yield a stable throughput estimate, a smoothing function over the estimate values is required.

Streaming clients cannot directly estimate the correct throughput when receiving pushed video segments. Cherif *et al.* [155] introduce to use WebSocket [161] messaging to improve throughput measurements, while an adaptation solution reduces the startup delay using HTTP/2 server-push. The basic idea is to calculate the throughput on the client from the elapsed time between WebSocket messages at the start and the end of sending a video segment. They transport the WebSocket messages over HTTP/2, which allow a client and a server to use a single TCP connection for both HTTP and WebSocket streams. As a result, the probing messages will traverse the same network path as the video stream. Compared to the traditional

WebSocket (over a dedicated TCP connection for throughput estimation), WebSocket over HTTP/2 provides higher precision in such a probing-based estimation.

Xiao *et al.* [158] observe that the throughput of a streaming session depends on not only the network conditions between the server and the client but also the network resource allocation of the servers. The distribution of streaming clients together with the network capacity of servers is beneficial in estimating the trend of available throughput variation in the long term. Provided the availability of the server support, it can be integrated into an existing estimation approach for improving the accuracy.

6.3 ADAPTATION ALGORITHM

Recently, a wide range of adaptation algorithms have been proposed to support HAS. The adaptation algorithms make bitrate decisions primarily based on throughput measurement or throughput estimate and/or client's playback buffer occupancy. In this section, we review the literature on adaptation algorithms, which employ such as control theory [81, 153], machine learning [162], dynamic programming [163], queuing theory [164, 165, 166], optimization techniques [31], fuzzy logic [154], and various heuristics [78, 167, 29, 60, 30]. As we will see, the most existing algorithms do not stabilize or do not effectively stabilize the client buffer dynamics, and thus typically adopt a large buffer size of e.g., more than $30\,s$ for $2\,s$ video segments, leading to a high streaming latency. In contrast, we propose an bitrate adaptation for the effective stabilization of the buffer dynamics, in order to support low-latency adaptive streaming.

6.3.1 *Throughput-based Approach*

A throughput-based adaptation method depends only or mainly on the throughput to select the bitrate for the next video segment. It is challenging to optimize the buffer size using this method. On one hand, the throughput estimate is prone to inaccurate or even erroneous. On the other hand, the throughput may significantly fluctuate due to network dynamics. Therefore, this method typically suffers from a huge buffer size and/or severe bitrate variations, even though various algorithms use heuristics to mitigate this situation.

The VLC media player plugin (DASH VLC plugin), developed by Müller and Timmerer [78], enables the VLC media player[2] to play DASH-based streaming. The plugin employs a simple throughput-based approach for bitrate selection that takes the mean of all throughput samples measured in the past as the estimated value and selects the highest bitrate from the bitrate set less than or equal to it. Most commercial streaming players appear to apply such a typical approach for bitrate selection [29]. Thus, we take this approach as a reference approach for the performance comparison in the present thesis.

Liu *et al.* [167] present a conservative algorithm for bitrate selection based on the throughput measured during the reception of the last segment and the minimum buffer level threshold for bitrate up-switching. The algorithm deploys a step-wise

2 VLC media player. https://www.videolan.org/vlc/index.html

increasing but quick decreasing mechanism for bitrate switching to prevent buffer underflows. To produce smoothed throughput measurements, the algorithm adopts video segments of 10 *s*. An evaluation is performed using ns-2[3] simulation, without a baseline approach. The averages of client buffer levels vary from 54 *s* to 110 *s* in scenarios of competing artificial background traffic with various sending rates.

Jiang *et al.* [29] propose the adaptation algorithm FESTIVE, targeting three performance metrics: maximizing the utilization of the network capacity, minimizing the number and magnitude of bitrate switches, and maximizing application-layer fairness between competing clients. FESTIVE aims to heuristically optimize these metrics based on three key steps. First, the available throughput is estimated by using the harmonic mean over a fixed number of last segment receptions. Second, based on the estimation, FESTIVE selects a target video bitrate and performs a stepwise bitrate switch. The execution of an up-switching is delayed to avoid frequent bitrate switching. The delayed up-switching may lead to the under-utilization of the network capacity that is controlled by resolving the trade-off between the utilization and the number of bitrate switches. Third, competing clients may not sense the presence of each other in the network and predict the wrong available throughput, because buffer overflow causes the OFF-period during ON-OFF streaming behaviors. FESTIVE randomizes the request time of video segments in the case of buffer overflow to overlap the ON-period and to sense the presence of other competing clients. The performance of FESTIVE is evaluated using 2 *s* video segments and around 30 *s* buffer size in a custom emulation framework. The results reveal that FESTIVE outperforms four commercial solutions with respect to all three considered performance metrics. We adopt FESTIVE as a baseline approach for the performance evaluation presented in Chapter 4.

6.3.2 *Buffer-based Approach*

Huang *et al.* [30] argue that the throughput estimation is unnecessary during the playback phase. They observe that accurate throughput estimation at the application layer is challenging in video streaming when the network throughput is highly variable. Their investigation suggests an alternative adaptation algorithm (BBA) simply based on the buffer information. BBA uses a lower buffer threshold (called *reservoir*) for the minimum bitrate and a upper buffer threshold (called *upper reservoir*) for the maximum bitrate to avoid underflow and overflow, respectively. It defines the region between the reservoir and the upper reservoir as a flexible region for bitrate switching called *cushion*, and linearly maps buffer occupancy to bitrate for switching in the cushion. BBA is deployed into a large commercial streaming service for experiments. A 10–20% improvement in the rate of rebuffering is observed compared to the default algorithm of the streaming service, when the experiments adopt 4 *s* video segment and a 240 *s* playback buffer. Since BBA is the first buffer-based approach for bitrate selection and its performance was evidenced by millions of real users in a commercial service, we use BBA as another baseline approach for the performance evaluation presented in Chapter 4.

Spiteri *et al.* [31] offer a theoretical justification for using buffer-based adaptation algorithms, by showing that a near-optimal algorithm within a utility optimization

3 The Network Simulator - ns-2. https://www.isi.edu/nsnam/ns/

framework requires only buffer information and no throughput estimate. The authors formulate bitrate adaptation as a utility optimization problem that rewards an increase in the average video bitrate and penalizes potential rebuffering time. Using Lyapunov optimization method [168], they derive an algorithm BOLA that decides a bitrate for video segments by solving the optimization problem. The derivation is based on the assumption that the buffer size is infinite. Thus, BOLA performs better with larger buffers, although dynamic buffer sizing is integrated into the algorithm to control the achievable performance. The authors limit the buffer size to 25 *s* in experiments of 3 *s* video segment and evaluate the algorithm using trace-driven simulation of mobile network. The performance of BOLA is within 84–95% of the optimal utility and better than two algorithms introduced in the literature with respect to the considered utility metric. BOLA is now part of dash.js since version 2.0.0.

Buffer-based methods leverage the client playback buffer to avoid the challenge on throughput estimation. Since these methods make bitrate decisions only based on the buffer information, they also require a sufficiently large buffer to absorb the fluctuation of the buffer occupancy incurred by the throughput variations and the bitrate mismatch, as well as to map the available bitrates to the buffer levels. Accordingly, the buffer usually need to accommodate 8–60 video segments.

6.3.3 *Buffer- and Throughput-based Approach*

As outlined above, adaptation methods solely using throughput or buffer occupancy achieve the suboptimal QoE. Many adaptation algorithms attempt to optimize the QoE, taking both into accounts. However, they are not suitable for low-latency adaptive streaming due to a lack of explicit stabilization of client buffer dynamics. Using a buffer size of more than 30 *s* (equivalently 15 video segments) is generally observed in their performance evaluation.

Yin *et al.* [32] develop an optimization problem of bitrate adaptation for QoE maximization and propose an adaptation algorithm using Model Predictive Control (MPC). The algorithm can directly optimize a formally defined QoE objective by solving the optimization problem. In the algorithm, the bitrate for the current segment is selected based on a throughput estimation for the next few segments as well as the buffer occupancy. But, its performance depends on the accuracy of such a throughput estimation. The algorithm also requires significant offline optimization to be performed outside of the client for an exhaustive set of scenarios. Last, the streaming model on which the optimization problem relies does not consider the network delay and the timing information influenced by it. This often results in erroneous optimization especially in the case of low-latency streaming.

Batalla *et al.* [164] introduce an adaptation algorithm that decides the video bitrate based on the estimated probability of rebuffering events. The estimation of the probability depends on a queuing model of the playback buffer using as input the characteristics of segment download time. Through online computations, the required threshold for buffer occupancy is determined to ensure a given rebuffering probability. The algorithm selects the highest bitrate that satisfies the buffer occupancy threshold. However, the adaptation algorithm suffers significant computational overhead. Beben *et al.* [165] exploit a pre-computed map for buffer

occupancy to avoid heavy online computation, although the modification lessens the effectiveness of the adaptation.

Sobhani *et al.* [154] propose an adaptation mechanism using fuzzy logic. In response to throughput variations, the mechanism takes into consideration the available throughput estimated by means of KAMA on past throughput measurements as well as the buffer dynamics predicted by Grey Prediction Model [169] based on recent buffer levels. The proposed system is evaluated in an emulated test bed with competing flows and is compared to four baseline adaptation algorithms. Each segment represents 6 s video and is encoded in 17 bitrates from 100 $kbps$ to 6000 $kbps$, while the buffer level can reach above 35 s in experiments.

Mao *et al.* [162] argue that bitrate adaptation based on preset rules in the form of fine-tuned heuristics fails to achieve the optimal performance across a broad set of application scenarios. Consequently, they present Pensieve, a system for bitrate adaptation using Reinforcement Learning (RL). Pensieve observes a set of states including the throughput measurements and the download time for the last few video segments, the available sizes of the next video segment, the current buffer level, the number of segments remaining in the video, and the selected bitrate of the last video segment, and feeds these values to a neural network model, which outputs a bitrate decision. The decision policy is derived from training the neural network through Asynchronous Advantage Actor-Critic (A3C) algorithm [170]. To speed up the training process, a simple simulation environment is used that models the dynamics of the client playback buffer. Pensieve is deployed on the server-side to simply guide client bitrate selection, thereof, easily supporting a broad range of heterogeneous clients; while direct client-side deployment needs a lightweight method to reduce the computational and memory overhead.

6.3.4 *Approach with Buffer Stabilization*

Despite using throughput and buffer occupancy as inputs, most methods are unable to support adaptive streaming with a small buffer size, as discussed above. Some methods determine a video bitrate such that the buffer occupancy can be controlled around a target level, thereof, being able to reduce the buffer size.

In the presence of competing clients, the ON-OFF streaming pattern may lead to bitrate oscillation and unfairness, as discussed in Section 2.2.3. To tackle this problem, Li *et al.* [60] present Probe AND Adapt (PANDA), an adaptation algorithm by devising two different steps other than conventional throughput-based adaptation algorithm: throughput estimation and segment scheduling. PANDA probes the available throughput using an AIMD-like algorithm, where the estimate value slowly increases but aggressively decreases in case of the overestimation. In the scheduling step, PANDA determines the requesting time for the next segment by driving the buffer level towards a minimum reference level. The performance of PANDA is however not explicit with respect to the number of bitrate switches. To reduce the quality variation, Li *et al.* [163] use dynamic programming to solve a utility optimization problem over a finite number of segments, and integrate the algorithm into PANDA to determine the bitrate based on the throughput estimate provided by PANDA. For the integration, an EWMA is used to smooth out the raw

throughput estimation. The algorithm is evaluated in the ns-2 simulation and the buffer size limits between 20 *s* and 50 *s*, using 2 *s* video segment.

In the context of live streaming, De Cicco *et al.* [81] investigate a commercial High-Definition (HD) streaming service and identify that the service underutilized the available throughput due to its conservative adaptation heuristic. They present a server-side adaptation architecture to avoid delays of the client feedback, thereof, providing a fast response to the adaptation dynamics. The main component of the architecture is the controller (QAC) for adaptive live streaming system designed by employing feedback control theory, without using any adaptation heuristics. QAC chooses a PI controller to steer the occupancy of a sender buffer and attempts to keep the sender buffer at a target level by selecting the video bitrate. In the control loop, the sender buffer occupancy is taken as the feedback signal and the throughput measurement is modelled as a disturbance. The key advantage of using feedback control is to get a predictable system dynamics that can fulfill the design goal such as settling time and steady state errors. As a result, the proposed solution is able to maintain the buffer occupancy at the client around 15 *s*. The improvement compared to the commercial service is evidenced in scenarios including step-like change of the throughput, square-wave varying throughput, and multiple competing flows. We take QAC as a representative of the server-side adaptation solution and use it as a baseline approach in the performance evaluation discussed in Chapter 5.

The study by Tian and Liu [153] introduces an adaptation algorithm which computes adjustment factors using buffer occupancy as feedback signal and selects a video bitrate based on the adjusted throughput measurement or estimation. The design goal of the algorithm is to achieve stable buffer occupancy around a target buffer level and a smoothly increasing of video bitrate. The responsiveness of the algorithm to highly varying throughput is demonstrated, when a target buffer level is set to 20 *s*. The segment duration is not specified in the publication. The proposed algorithm is similar in spirit to our work in adaptation algorithm. However, the algorithm does not take into account the network delay, which is essential to the design for low-latency streaming. Moreover, the algorithm has a set of configuration parameters that does not allow for a straightforward tuning of the algorithm for particular network environments.

Yadav *et al.* [166] model the streaming client as a queuing system, which allows to calculate the expected buffer occupancy given a video bitrate, the estimated throughput, and current buffer occupancy. Using this queuing model, they present QUETRA, an adaptation algorithm which selects a video bitrate such that the buffer occupancy converges to 50% of the maximum buffer level. As a result, the client aims to keep the buffer occupancy away from the zero and maximum level, in order to avoid playback stalls due to buffer underflows and the OFF-period of the ON-OFF streaming pattern. For efficiency reason, a pre-computation of expected buffer occupancy is required in the implementation of the algorithm. Through experiments with various buffer sizes (30–240 *s*) and segment durations (2–5 *s*), it demonstrates that the algorithm outperforms four considered baseline algorithms in four different network profiles and seven video profiles.

These prior works reveal that buffer stabilization is important for low-latency streaming. However, the main limiting factors of these works are two-fold. First,

the algorithms do not explicitly incorporate the handling of the network delay in their designs. Second, it does not allow for a straightforward configuration of the algorithm parameters.

6.4 TRANSPORT PROTOCOLS

Video streaming has tight latency constraints, and data arriving too late is effectively lost. Late arrivals of video data are detrimental to the QoE, as error occurrences of video data are, which are generally due to packet losses. Streaming applications run on top of transport protocols and ideally trust the transport layer to minimize induced delays and to deliver an appropriate degree of reliability and timeliness. These features offered by the transport protocols essentially rely on the error and congestion control mechanisms implemented. Below, we will overview the research efforts on both mechanisms. Our work is complementary to these efforts and will benefit from a better error control and/or a better congestion control.

6.4.1 *Error Control*

TCP prefers reliability to timeliness and is designed to support *fully reliability* using an Automatic Repeat reQuest (ARQ) method. In ARQ, the sender repeats the transmission of data segments until obtaining acknowledgments from the receiver, and initiates the retransmissions upon a timeout. Suchlike error control inevitably comes at the price of unpredictable delay, and thus is infeasible for strict delay-constraint flows. There are efforts on the error control at the application layer — e.g., Stream Control Transmission Protocol (SCTP) [171], Datagram Congestion Control Protocol (DCCP) [172], Predictably Reliable Real-time Transport (PRRT) [38], Quick UDP Internet Connections (QUIC) [173], and TCP Hollywood [174] — to support the transport service features required by video delivery.

PRRT [38] introduces the concept of *predictably reliability* for delay-constrained and loss-tolerant communications services. PRRT implements adaptive reliability control under delay, reliability and throughput constraints, using so-called hybrid ARQ, which combines ARQ and Forward Error Correction (FEC) to mutually compensate for each other's drawbacks. As a result, PRRT achieves capacity-approaching error control on bidirectional packet-loss channels [139].

QUIC [173] is a transport protocol built atop UDP, and is designed to provide security and reliability along with reduced connection and transport delay (versus TCP). QUIC implements retransmissions and congestion control at the application layer. Key features of QUIC include dramatically reduced connection establishment time, improved congestion control, multiplexing without head-of-line blocking, and FEC. A performance evaluation [175], however, shows that there is no significant benefit to the QoE in ABS over QUIC.

On the other hand, novel streaming protocols are limited to be widely deployed due to the transport-layer ossification [176, 177]: streaming applications is restricted to use either TCP or UDP at the transport layer, neither of which is well-suited to their needs. TCP Hollywood [174] reduces the overall delay, by removing head-of-line blocking and offering *partial reliability*. It implements out-of-order delivery and inconsistent retransmissions to meet the application-specific latency bounds. The

modification of TCP Hollywood compared to the standard TCP is constrained for the ease of implementation and wide deployment.

6.4.2 *Congestion Control*

Congestion control prevents the network from being overwhelmed by excess data and serves competing flows for a fair share of the network. As a general-purpose transport protocol, TCP is inherently equipped with a congestion control mechanism. Most variants of TCP to date, however, are not optimized for video flows, and many streaming services are exploring various congestion control algorithms for improvements.

Akamai[4] adopts FAST TCP [39] for improving the throughput of video flows, instead of mainstream TCP variants such as TCP CUBIC [143] and TCP New Reno [178], which use packet loss as congestion signal. Loss-based congestion controls tend to completely fill the network buffer at a bottleneck link and induce a large queuing delay that adversely affects the performance of all flows on the network. In contrast, FAST TCP uses queuing delay (typically characterized by RTT) as congestion signal and aims towards high speed wide-area networks. The deployment of FAST TCP shows the enhancement in average video bitrate and rebuffering [179].

Recently, Google and YouTube deploy a variant of TCP congestion control — Bottleneck Bandwidth and Round-trip propagation time (BBR) [180] — on their services. BBR models the network path using the throughput and RTT at which the network delivers the most recent flight of outbound data. By dynamically estimating the maximum throughput and the minimum RTT, BBR finds the optimal operation point to respond to actual congestion, rather than packet loss.

Another variance of TCP is Compound TCP [181], which detects network congestion based on both packet loss and queuing delay. Compound TCP is the default congestion control used by Microsoft Windows. It characterizes low variations of network delay as the sign of the under-utilization for the network, and improves the fairness and the throughput by being aggressive only when the bottleneck link of the network is underutilized.

When adopting a congestion control algorithm, it is important to avoid creating interactions between congestion control and video bitrate adaptation, as Section 2.2.3 have discussed that the interactions may cause a downward spiral in video quality.

6.5 VIDEO ENCODING SCHEMES

Recent advancement in camera, display, and image processing technology has generated a paradigm shift from traditional 2D video to MVV technology. Although state-of-the-art video technologies still encircle ubiquitous 2D video, the next big step toward in video technology is destined to be MVV display and distribution. In the following, we outline video encoding schemes according to two cases: single-view (2D) video and multi-view video.

4 Akamai Techonologies. https://www.akamai.com/

6.5.1 *Single-View Video*

The recent rapidly developing ABS commonly uses single-layer coding like Advanced Video Coding (H.264/AVC) and High Efficiency Video Coding (H.265/HEVC). Nevertheless, the notion of adaptive quality for video streaming over Internet was first introduced, when video quality was adapted by using layered coding [182, 46]. A modern example of layered coding is the Scalable Video Coding (SVC) standard, which is an amendment to AVC standard and provides temporal, spatial, and image quality scalability [183].

With single-layer coding, each video segment is encoded into multiple versions according to video bitrate, each of which represents the same video as an independent and separate file but with different image quality. One adapts the video quality by switching the version i.e., by requesting a different file. Layered coding, on the other hand, encodes the video segment into a base layer and one or more enhancement layers in a hierarchical and cumulative way. The base layer represents the video with the lowest video bitrate, and each enhancement layer contains only the additional information for the next higher bitrate compared to the preceding layer. As receiving the more layers, the client reconstructs the video with the higher video bitrate. With such a manner, layered coding offers more resilience toward rebuffering and quality variation than single-layer coding, because it allows progressive video reconstruction — i.e., the client always first downloads the base layer, and optionally requests enhancement layers when enough throughput is available.

Though, there are two trade-offs that need to be considered in case layered coding is employed in an ABS solution. First, layered coding introduces extra encoding overhead. For example, SVC requires about 20% more bits to achieve the same quality as AVC, and this overhead is content dependent [184]. Also, the overhead increases along with the higher video bitrate [185]. Hence, video streaming today prefers single-layer coding over layered coding. In addition, since the bandwidth cost dominates the cost of streaming services, single-layer coding is currently more favored in the industry [184]. Second, the flexibility of requesting enhancement layers comes at the cost of increased requesting traffic, as several requests are needed per video segment. This should be taken into account in scenarios with high network delay. It implies that layered coding adapts more easily to highly variable throughput (such as in mobile scenarios) when a small client buffer size is used.

6.5.2 *Multi-View Video*

MVV is an emerging video form enabled by advances in multi-camera capturing systems, stereoscopic display technologies, as well as image-based modeling and rendering algorithms. With MVV, a scene is simultaneously recorded by multiple cameras from different viewpoints (angles). The application domains of MVV is diverse, and one example is Free Viewpoint Video (FVV), which uses multi-camera systems to visualize the scene from arbitrary viewpoints [186]. FVV allows a user to interactively navigate natural scenes (captured videos of real-world scenes) similarly to the way synthetic scenes (computer graphics-generated scenes) are interacted with. To support a seamless transition and a similar image quality across all views, view-rendering algorithms synthesize virtual views of real-world scenes

at locations between and around originally-captured viewpoints by warping and blending a number of captured camera streams.

The aforementioned 2D video coding standards — although not specifically designed for the compression of MVV — are readily applicable to encode MVV by treating each camera stream independently as a conventional 2D video. This strategy, however, makes MVV distribution consume several times larger throughput than 2D video, since it contains multiple separate video streams captured by multiple cameras. Furthermore, the throughput consumption will increase, as the number of views increases. In practice, such a linearly increasing overhead is redundant, because all views represent the same scene but from different perspectives. It implies that the inter-view statistical dependency also increases along with the increase of view numbers. Taking advantage of the redundancies between the different views in MVV, an efficient inter-view prediction is usually adopted to reduce the throughput overhead. For instance, the Multi-view Video Coding (MVC) standard [187, 188] specifically targets efficient compression of multi-camera video recordings by exploiting the inter-view, spatial and temporal dependencies to improve the rate-distortion performance. MVC is largely based on the existing H.264/AVC standard and extends it with the optimized compression on inter-view and intra-view prediction dependencies.

In this thesis, we focus merely on bitrate adaptation using single-layer coding, however it should be noted that many aspects of video encoding profoundly influence the design and the performance of bitrate adaptation. For example, the bitrate variation caused by the VBR encoding as discussed in Section 2.2.4 affects the buffer level the client needs to preserve, and the granularity of the available video bitrates for adaptation determines the effect of bitrate switches.

6.6 GLOBAL OPTIMIZATION

Most bitrate adaptation attempts to optimize purely the performance of a single client, since it lacks of a global view of all clients and network conditions. Even a near-ideal bitrate adaptation is not sufficient, because there is significant variability in network and CDN performance [189]. A global optimization [84] is proposed to alleviate these concerns by coordinating actions across multiple clients. For instance, a global optimization is able to proactively allocate a better performing CDN to clients before they suffer from congestion links. Results show that the global controller provides an improvement of average video bitrate by 70%. CDN federation and peer-assisted hybrid CDN are two emerging approaches to augment existing CDN infrastructure that have recently attracted significant industry attention. A study [190] analyzes the potential benefits of both approaches based on content, regional and peak-time effects, viewing behavior, and interest predictability. These prior works are complementary and beneficial to our contributions.

7

CONCLUSIONS AND FUTURE WORK

Internet-based streaming already became the main form of video distribution and its ecosystem is currently evolving extremely fast. The classical broadcasting services on the other hand, are being transformed, complemented, and partially replaced by emerging online-television services that aim to deliver superior video quality to users, while offering high interactivity, customization, and interconnection. Due to the absence of QoS guarantees, Internet-based streaming needs to adapt its services to the varying network conditions. In particular, while wireless and mobile video traffic are persistently increasing their share of Internet traffic, highly variable connectivity of wireless networks, especially in mobile environments, intensifies the variability of network conditions severely. Significant progress has been made in optimizing the streaming performance and adaptation solutions. The viewing experience however, is not comparable to classical broadcasting: the most notorious impairments include frequent rebuffering events, high latency of video playback, as well as low and/or heavily varying image quality. Additionally, the rising demand for reduced latency and increased throughput, as well as the soaring user-expectations for a better viewing experience make the issues more challenging.

In contrast to existing efforts that focus on adaptation solutions employing excess buffer sizes in the range of several tens of seconds, we propose a novel approach for adaptive streaming that minimizes the required buffer sizes on clients in order to fulfill the requirements of live video broadcasting. Following this design goal, we first analyze the elements influencing the streaming latency and identify the buffering as the dominant element. We introduce an analytical model of adaptive streaming for buffer size and derive the theoretical lower bound of the buffering size and an approximate minimum, in order to determine a reasonable low latency for streaming systems. Based on the model, we implement a simulation test-bed for benchmark comparisons. Then, we develop an adaptation algorithm for low-latency adaptive streaming, targeting the upper bound of the latency from the user's perspective. The algorithm models buffer dynamics in a way motivated by the analytical model for buffer size and performs quality-optimized bitrate selection based on the stabilization of client buffer dynamics. It is evaluated in simulated and realistic network environments and outperforms the corresponding baseline algorithms with respect to the considered performance metrics. Finally, we succeed in further improvements for low-latency streaming, by designing a server-side adaptation architecture to minimize the feedback delay in the control loop of the bitrate adaptation and optimizing a transport-layer mechanism to support real-time streaming with predictable reliability and high throughput utilization. The advancement of both steps is beneficial for stabilizing the buffer dynamics at the client. Through

intensive experiments, significant improvements of our solutions are demonstrated in terms of user-perceived quality.

This thesis is addressing adaptation solutions in the field of low-latency adaptive streaming and our results show that buffer stabilization in an adaptation design is a principal criterion to push the streaming latency to its limit. We believe the design principle will continue helping low-latency streaming solutions to deal with the increasingly challenging environments in the future. There are still many open questions in this field, and more research efforts are needed for applications based on this work to become commonplace in the future Internet. We envision several directions for future work based on the findings of this thesis. First, a coordination between the global adaptation and the individual adaptation on clients has the potential of greatly boosting the QoE of all users as well as a single user, e.g., by allowing load balancing and resource allocation based on large-scale measurement feedback. Second, the evolution of adaptation solutions will need to be accompanied by additional work on advanced video coding and the optimization of video parameters, e.g., regarding the bitrate variation across video segments and the granularity of available bitrates. Third, recent trends towards VR and AR as well as MVV drive the extension of adaptive streaming in these fields, where further optimization techniques are needed to reduce the vast throughput requirements e.g., by elaborately selecting the required content and only distributing those to clients. Lastly, the design guide of adaptation solutions benefits from a better understanding of the QoE for adaptive streaming, including system, context, and human factors. An important area of future research should therefore strive to build a holistic QoE model and a corresponding monitoring system covering all relevant QoE factors.

A

APPENDIX

A.1 DERIVATION OF APPROXIMATION FOR MINIMUM BUFFERING SIZE

We derive the minimum buffer level B_{min}^{deg} of Equation 3.24 based on the number of the segments k which are completely received during a network degradation with a duration D^{deg}.

$k = 0$: No video segment can be completely received during D^{deg}. The time required for the reception of a complete segment within D^{deg} is $\Delta t + \frac{r}{C^{deg}} T_c$, and $k = 0$ is the case that this time is bigger than D^{deg}, i.e., $D^{deg} < \Delta t + \frac{r}{C^{deg}} T_c$. However, for continuous playback, the video in the buffer needs to last until the reception of at least one segment completes. Therefore, the reception finishes after the network degradation. Let y be the time required to receive the portion of a segment (or the entire segment) after the network degradation. We have

$$\left(D^{deg} - \Delta t\right)_+ \cdot C^{deg} + y \cdot C_a^{deg} = r \cdot T_c \,. \tag{A.1}$$

The function $(\cdot)_+$ of the term $\left(D^{deg} - \Delta t\right)_+$ takes into account the case, in which the reception of the segment starts after the network degradation if $D^{deg} \leq \Delta t$; namely, in this case y represents the time for receiving the entire segment after the network degradation. Then the video required in the buffer for continuous playback prior to the network degradation starts can be expressed as

$$\begin{aligned} B_{min}^{deg} &= \max\left\{D^{deg}, \Delta t\right\} + y \\ &= \max\left\{D^{deg}, \Delta t\right\} + \frac{r \cdot T_c - C^{deg} \cdot \left(D^{deg} - \Delta t\right)_+}{C_a^{deg}}, , \\ & D^{deg} < \Delta t + \frac{r}{C^{deg}} T_c \end{aligned} \tag{A.2}$$

where the term $\max\{\cdot\}$ considers the case of $D^{deg} \leq \Delta t$.

$k = 1$: Only one segment is completely received within D^{deg}. So we have $\Delta t + \frac{r}{C^{deg}} T_c \leq D^{deg} < 2 \cdot \left(\Delta t + \frac{r}{C^{deg}} T_c\right)$. Note that the client may initiate a request to receive an additional segment within D^{deg} but completes the reception after the network degradation. Let

$$D_{res}^{deg} = D^{deg} - k \cdot \left(\Delta t + \frac{r}{C^{deg}} T_c\right) \tag{A.3}$$

denote the remaining time of D^{deg} after the entire reception of all k segments during the time period D^{deg}. Similar to the case of $k = 0$, the time y required to finish the reception of the additional segment after the the network degradation fulfills

$$\left(D_{res}^{deg} - \Delta t\right)_+ \cdot C^{deg} + y \cdot C_a^{deg} = r \cdot T_c \,.$$

Then we can compute the minimum buffer level as

$$\begin{aligned} B_{min}^{deg} =& \Delta t + \frac{r}{C^{deg}} T_c + \left(\max\left\{D_{res}^{deg},\, \Delta t\right\} + y - T_c\right)_+ \,, \\ & \Delta t + \frac{r}{C^{deg}} T_c \leq D^{deg} < 2 \cdot \left(\Delta t + \frac{r}{C^{deg}} T_c\right) \end{aligned} \,, \tag{A.4}$$

where

$$D_{res}^{deg} = D^{deg} - \left(\Delta t + \frac{r}{C^{deg}} T_c\right)$$

and

$$y = \frac{r \cdot T_c - C^{deg} \cdot \left(D_{res}^{deg} - \Delta t\right)_+}{C_a^{deg}} \,. \tag{A.5}$$

The playback time of video data in the buffer should cover the time required for receiving all segments, including ones whose receptions complete within D^{deg} and the potential one whose reception finishes after the network degradation. Note that video data in the buffer increases by one segment duration T_c after the complete reception of a segment and the increase compensates the time required for receiving the following segments. Therefore, the computation of B_{min}^{deg} in Equation A.4 subtracts T_c from the time required for the following reception $\max\left\{D_{res}^{deg},\, \Delta t\right\} + y$. The function $(\cdot)_+$ ensures that B_{min}^{deg} is at least $\Delta t + \frac{r}{C^{deg}} T_c$, such that the client can receive one entire segment and fill the buffer with T_c seconds video data, if $\max\left\{D_{res}^{deg},\, \Delta t\right\} + y < T_c$ e.g., due to a sufficient large C_a^{deg}.

$k = 2$: The duration of the network degradation needs to be $2 \cdot \left(\Delta t + \frac{r}{C^{deg}} T_c\right) \leq D^{deg} < 3 \cdot \left(\Delta t + \frac{r}{C^{deg}} T_c\right)$, such that the client can complete the reception of exact two segments within D^{deg}. Analog to the case of $k = 1$, we have

$$D_{res}^{deg} = D^{deg} - 2 \cdot \left(\Delta t + \frac{r}{C^{deg}} T_c\right)$$

and

$$\left(D_{res}^{deg} - \Delta t\right)_+ \cdot C^{deg} + y \cdot C_a^{deg} = r \cdot T_c \,,$$

then

$$y = \frac{r \cdot T_c - C^{deg} \cdot \left(D_{res}^{deg} - \Delta t\right)_+}{C_a^{deg}} \,.$$

Because the increase of the buffer level (by T_c) due to the complete reception of a segment (i.e., the first segment in this case) compensates for the decrease of the

buffer level (by $\Delta t + \frac{r}{C^{deg}} T_c$) during the reception of the next segment (i.e., the second segment), the minimum buffer level prior to the network degradation is

$$\begin{aligned} B_{min}^{deg} =& \Delta t + \frac{r}{C^{deg}} T_c + \Delta t + \frac{r}{C^{deg}} T_c - T_c + \\ & \left(\max \left\{ D_{res}^{deg}, \Delta t \right\} + y - T_c \right)_+ \\ =& 2 \cdot \left(\Delta t + \frac{r}{C^{deg}} T_c \right) - T_c + \\ & \left(\max \left\{ D_{res}^{deg}, \Delta t \right\} + y - T_c \right)_+ , \\ & 2 \cdot \left(\Delta t + \frac{r}{C^{deg}} T_c \right) \leq D^{deg} < 3 \cdot \left(\Delta t + \frac{r}{C^{deg}} T_c \right) \end{aligned} . \tag{A.6}$$

$k = K$: Because exact K segments are totally received within D^{deg}, it holds that

$$K \cdot \left(\Delta t + \frac{r}{C^{deg}} T_c \right) \leq D^{deg} < (K+1) \left(\Delta t + \frac{r}{C^{deg}} T_c \right) . \tag{A.7}$$

Similarly, we have

$$D_{res}^{deg} = D^{deg} - K \cdot \left(\Delta t + \frac{r}{C^{deg}} T_c \right)$$

and

$$\left(D_{res}^{deg} - \Delta t \right)_+ \cdot C^{deg} + y \cdot C_a^{deg} = r \cdot T_c ,$$

then

$$y = \frac{r \cdot T_c - C^{deg} \cdot \left(D_{res}^{deg} - \Delta t \right)_+}{C_a^{deg}} .$$

Accumulating the increase and the decrease of the buffer level due to the reception of all K segments, we obtain the minimum buffer level as

$$\begin{aligned} B_{min}^{deg} =& \Delta t + \frac{r}{C^{deg}} T_c + \underbrace{\Delta t + \frac{r}{C^{deg}} T_c - T_c + \cdots}_{K-1} + \\ & \left(\max \left\{ D_{res}^{deg}, \Delta t \right\} + y - T_c \right)_+ \\ =& K \cdot \left(\Delta t + \frac{r}{C^{deg}} T_c \right) - (K-1) \cdot T_c + \\ & \left(\max \left\{ D_{res}^{deg}, \Delta t \right\} + y - T_c \right)_+ , \\ & K \cdot \left(\Delta t + \frac{r}{C^{deg}} T_c \right) \leq D^{deg} < (K+1) \left(\Delta t + \frac{r}{C^{deg}} T_c \right) \end{aligned} . \tag{A.8}$$

Now, we derive the number of the received segments K within D^{deg} based on the inequalities in Equation A.7. We can formulate K as follows based on $D^{deg} - K \cdot \left(\Delta t + \frac{r}{C^{deg}} T_c \right) \geq 0$ and $D^{deg} - (K+1) \left(\Delta t + \frac{r}{C^{deg}} T_c \right) < 0$, respectively:

$$K \leq \frac{D^{deg}}{\Delta t + \frac{r}{C^{deg}} T_c} , \tag{A.9}$$

$$K > \frac{D^{deg}}{\Delta t + \frac{r}{C^{deg}} T} - 1 \,. \tag{A.10}$$

Because K is a non-negative integer, by combining Inequation A.9 and Inequation A.10 we have

$$K = \left\lfloor \frac{D^{deg}}{\Delta t + \frac{r}{C^{deg}} T_c} \right\rfloor \,. \tag{A.11}$$

Considering the above cases, we can formulate the minimum buffer level required for continuous playback in case of the network degradation with a duration D^{deg} as

$$\begin{aligned} B_{min}^{deg} =& K \cdot \left(\Delta t + \frac{r}{C^{deg}} T_c \right) - (K-1)_+ \cdot T_c + \\ & \left(\max \left\{ D_{res}^{deg}, \Delta t \right\} + y - T_c|_{K>0} \right)_+ \end{aligned} \,, \tag{A.12}$$

where

$$K = \left\lfloor \frac{D^{deg}}{\Delta t + \frac{r}{C^{deg}} T_c} \right\rfloor \,,$$

$$D_{res}^{deg} = D^{deg} - K \cdot \left(\Delta t + \frac{r}{C^{deg}} T_c \right) \,,$$

$$y = \frac{r \cdot T_c - C^{deg} \cdot \left(D_{res}^{deg} - \Delta t \right)_+}{C_a^{deg}} \,,$$

and

$$T_c|_{K>0} = \begin{cases} T_c & K > 0 \\ 0 & else \end{cases} \,.$$

■

A.2 PROOF OF MINIMUM QUANTIZATION ERROR

In the following we give a proof of Inequality 4.6; i.e., we show that the bitrate selection of BDS (Equation 4.5) minimizes the two-fold quantization error: i.) deviation of the nominal bitrate (R) from the average bitrate of the segment ($\frac{S_i^R}{T_c}$) due to VBR encoding and ii.) deviation of the desired bitrate (r) from the nominal bitrate. Assume r_i is the desired bitrate for segment i that minimizes the buffer deviation. Cf. Equation 4.5, the desired bitrate r_i can be calculated as

$$r_i = \arg\min_{r \in \mathbb{R}} \left| b_{i-1}^e + T_c - \left(\frac{r \cdot T_c}{a_i} + \Delta t_i \right) - \beta_{ref} \right| \quad . \tag{A.13}$$

Then, we have

$$0 = b_{i-1}^e + T_c - \left(\frac{r_i \cdot T_c}{a_i} + \Delta t_i \right) - \beta_{ref} \ , \tag{A.14}$$

and thus obtain the desired bitrate r_i with

$$r_i = \left(b_{i-1}^e + T_c - \Delta t_i - \beta_{ref} \right) \cdot \frac{a_i}{T_c} \ . \tag{A.15}$$

On the other hand, Equation 4.5 for calculating the nominal bitrate R_i of segment i can be reformulated as

$$\begin{aligned} R_i &= \arg\min_{R \in \mathcal{R}} \left| b_{i-1}^e + T_c - \left(\frac{S_i^R}{a_i} + \Delta t_i \right) - \beta_{ref} \right| \\ &= \arg\min_{R \in \mathcal{R}} \left| b_{i-1}^e + T_c - \Delta t_i - \beta_{ref} - \frac{S_i^R}{a_i} \right| \\ &= \arg\min_{R \in \mathcal{R}} \left| \left(b_{i-1}^e + T_c - \Delta t_i - \beta_{ref} - \frac{S_i^R}{a_i} \right) \cdot \frac{a_i}{T_c} \right| \\ &= \arg\min_{R \in \mathcal{R}} \left| \left(b_{i-1}^e + T_c - \Delta t_i - \beta_{ref} \right) \cdot \frac{a_i}{T_c} - \frac{S_i^R}{T_c} \right| \end{aligned} \quad . \tag{A.16}$$

By substituting r_i from Equation A.15 into Equation A.16, we get

$$\begin{aligned} R_i &= \arg\min_{R \in \mathcal{R}} \left| b_{i-1}^e + T_c - \left(\frac{S_i^R}{a_i} + \Delta t_i \right) - \beta_{ref} \right| \\ &= \arg\min_{R \in \mathcal{R}} \left| r_i - \frac{S_i^R}{T_c} \right| \end{aligned} \quad . \tag{A.17}$$

This implies that the bitrate selection of BDS is equivalent to the bitrate selection for minimizing the deviation of the average bitrate from the desired bitrate. Namely, given a set of nominal bitrates of the video $\mathcal{R}$ and R_i computed with Equation 4.5, we have

$$\forall R \in \mathcal{R} \text{ and } R \neq R_i : \left| r_i - \frac{S_i^{R_i}}{T_c} \right| < \left| r_i - \frac{S_i^R}{T_c} \right| \ , \tag{A.18}$$

because of

$$\forall R \text{ and } R' \in \mathcal{R} : \ R \neq R' \iff S_i^R \neq S_i^{R'} \ . \tag{A.19}$$

■

A.3 PUBLICATIONS

This appendix contains the publication list of the author of the present thesis.

Included Publications

The main contributions included in this thesis draw heavily on the following publications in the field of video streaming:

- Yongtao Shuai and Thorsten Herfet. Towards Reduced Latency in Adaptive Live Streaming. In *Proceedings* of *the IEEE Consumer Communications and Networking Conference (CCNC)*, Las Vegas, USA, January 2018.
- Yongtao Shuai and Thorsten Herfet. On Stabilizing Buffer Dynamics for Adaptive Video Streaming with a Small Buffering Delay. In *Proceedings of the IEEE Consumer Communications and Networking Conference (CCNC)*, Las Vegas, USA, January 2017.
- Yongtao Shuai and Thorsten Herfet. A Buffer Dynamic Stabilizer for Low-Latency Adaptive Video Streaming. In *Proceedings of the IEEE International Conference on Consumer Electronics - Berlin (ICCE-Berlin)*, Berlin, Germany, September 2016.
- Yongtao Shuai and Thorsten Herfet. Improving User Experience in Low-Latency Adaptive Streaming by Stabilizing Buffer Dynamics. In *Proceedings of the IEEE Consumer Communications and Networking Conference (CCNC)*, Las Vegas, USA, January 2016.
- Yongtao Shuai, Goran Petrovic, and Thorsten Herfet. OLAC: an Open-Loop Controller for Low-Latency Adaptive Video Streaming. In *Proceedings of the IEEE International Conference on Communications (ICC)*, London, UK, June 2015.
- Yongtao Shuai, Goran Petrovic, and Thorsten Herfet. Open-Loop Rate Control for Adaptive Video Streaming. In *Proceedings of the IEEE Consumer Communications and Networking Conference (CCNC)*, Las Vegas, USA, January 2015.
- Yongtao Shuai, Goran Petrovic, and Thorsten Herfet. Server-Driven Rate Control for Adaptive Video Streaming using Virtual Client Buffers. In *Proceedings of the IEEE International Conference on Consumer Electronics - Berlin (ICCE-Berlin)*, Berlin, Germany, September 2014.
- Yongtao Shuai, Manuel Gorius, and Thorsten Herfet. Low-Latency Dynamic Adaptive Video Streaming. In *Proceedings of the IEEE International Symposium on Broadband Multimedia Systems and Broadcasting (BMSB)*, Beijing, China, June 2014.

Supplemental Publications

The following publications are related with video streaming, but are not considered in the presented thesis:

- Jochen Miroll, Yongtao Shuai, and Thorsten Herfet. Network Delay Estimation in Reverse Genlock Synchronized Display Walls. In *15. ITG Fachtagung für elektronische Medien (Dortmunder Fernsehseminar)*, Dortmund, Germany, February 2013.

- Manuel Gorius, Yongtao Shuai, and Thorsten Herfet. Dynamic Media Streaming with Predictable Reliability and Opportunistic TCP-Friendliness. In *Proceedings of the IEEE Consumer Communications and Networking Conference (CCNC)*, Las Vegas, USA, January 2013.

- Manuel Gorius, Yongtao Shuai, and Thorsten Herfet. Dynamic media streaming over wireless and mobile IP networks. In *Proceedings of the IEEE International Conference on Consumer Electronics - Berlin (ICCE-Berlin)*, Berlin, Germany, September 2012.

- Manuel Gorius, Yongtao Shuai, and Thorsten Herfet. Dynamic media streaming under predictable reliability. In *Proceedings of the IEEE International Symposium on Broadband Multimedia Systems and Broadcasting (BMSB)*, Seoul, Korea, June 2012.

- Manuel Gorius, Yongtao Shuai, and Thorsten Herfet. Predictably Reliable Media Transport over Wireless Home Networks. In *Proceedings of the IEEE Consumer Communications and Networking Conference (CCNC)*, Las Vegas, USA, January 2012. (Best Student Paper)

BIBLIOGRAPHY

[1] Cisco Visual Networking Index: Forecast and Methodology, 2016-2021. Whitepaper, Cisco, June 2017. https://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/complete-white-paper-c11-481360.pdf.

[2] Global Internet Phenomena, Latin Americca and North America. Report, Sandvine, June 2016. https://www.sandvine.com/hubfs/downloads/archive/2016-global-internet-phenomena-report-latin-america-and-north-america.pdf.

[3] Emerging Video Services. Report, Leichtman Research Group, Inc., July 2017. http://www.leichtmanresearch.com/research/notes09_2017.pdf.

[4] ETSI. ETSI EN 302 769 V1.3.1 - Digital Video Broadcasting (DVB); Frame structure channel coding and modulation for a second generation digital transmission system for cable systems (DVB-C2), 2015.

[5] ETSI. ETSI EN 302 307-1 V1.4.1 - Digital Video Broadcasting (DVB); Second generation framing structure, channel coding and modulation systems for Broadcasting, Interactive Services, News Gathering and other broadband satellite applications; Part 1: DVB-S2, 2014.

[6] ETSI. ETSI EN 302 755 V1.4.1 - Digital Video Broadcasting (DVB); Frame structure channel coding and modulation for a second generation digital terrestrial television broadcasting system (DVB-T2), 2015.

[7] B. Wang, J. Kurose, P. Shenoy, and D. Towsley. Multimedia streaming via TCP: An analytic performance study. *ACM Trans. Multimedia Comput. Commun. Appl.*, 4(2):16:1–16:22, May 2008.

[8] State of the Internet / Connectivity Report Q1 2017. Report, Akamai, 2017. https://content.akamai.com/uk-en-pg9141-q1-soti-connectivity.html.

[9] S. J. Berman, S. Canepa, D. Toole, and R. Van den Dam. Becoming a "Living" Media Partner for Your Consumers: A Cognitive Future for Media and Entertainment. Whitepaper, IBM Institute for Business Value, September 2017.

[10] ITU-T. Vocabulary for Performance and Quality of Service, Amendment 2: New Definitions for Inclusion in Recommendation ITU-T P.10/G.100, 2008.

[11] R. Pantos and W. May. HTTP Live Streaming. RFC 8216, Internet Engineering Task Force (IETF), August 2017.

[12] A. Zambelli. IIS Smooth Streaming Technical Overview. Whitepaper, Microsoft Corporation, March 2009.

[13] J. A. Bocharov. Smooth Streaming Transport Protocol. Specification, Microsoft Corporation, September 2017. https://docs.microsoft.com/en-us/iis/media/smooth-streaming/smooth-streaming-transport-protocol.

[14] HTTP Dynamic Streaming. Adobe Systems Incorporated, 2018. https://www.adobe.com/products/hds-dynamic-streaming.html.

[15] T. Stockhammer. Dynamic adaptive streaming over HTTP – standards and design principles. In *ACM Multimedia Systems (MMSys)*, 2011.

[16] I. Sodagar. The MPEG-DASH Standard for Multimedia Streaming Over the Internet. *IEEE MultiMedia*, 18(4):62–67, Oct 2011.

[17] ISO/IEC. 23009-1:2014 - Dynamic adaptive streaming over HTTP (DASH): Media presentation description and segment formats, 2014.

[18] A. Bateman, J. Smith, A. Colwell, M. Watson, and M. Wolenetz. Media source extensions™. W3C recommendation, W3C, November 2016.

[19] S. Faulkner, A. Eicholz, S. Moon, A. Danilo, and T. Leithead. HTML 5.2. W3C recommendation, W3C, December 2017.

[20] ISO/IEC. ISO/IEC 23009-19:2018 - Multimedia application format (MPEG-A) – Part 19: Common media application format (CMAF) for segmented media, 2018.

[21] ETSI. ETSI TS 103 285 v1.1.1 - Digital Video Broadcasting (DVB); MPEG-DASH profile for transport of ISO-BMFF based DVB services over IP based networks, 2015.

[22] ATSC. ATSC Standard: ATSC 3.0 System (A/300), 2017.

[23] ETSI. ETSI TS 102 796 V1.4.1 - Hybrid Broadcast Broadband TV, 2016.

[24] S. Akhshabi, S. Narayanaswamy, A. C. Begen, and C. Dovrolis. An experimental evaluation of rate-adaptive video players over HTTP. *Signal Proc.: Image Comm.*, 27(4):271 – 287, 2012.

[25] T. C. Thang, H. T. Le, A. T. Pham, and Y. M. Ro. An evaluation of bitrate adaptation methods for HTTP live streaming. *IEEE Journal on Selected Areas in Communications*, 32(4):693–705, 2014.

[26] T. Karagkioules, C. Concolato, D. Tsilimantos, and S. Valentin. A Comparative Case Study of HTTP Adaptive Streaming Algorithms in Mobile Networks. In *ACM NOSSDAV*, 2017.

[27] A. Burkitt-Gray. Deutsche Telekom promises European 5G rollout in 2020. http://www.capacitymedia.com/Article/3665079/Deutsche-Telekom-promises-European-5G-rollout-in-2020.

[28] M. Biegajewski. State of 5G deployment all over the world in 2017. http://www.rfbenchmark.eu/state-of-5g-deployment-world-2017/.

[29] J. Jiang, V. Sekar, and H. Zhang. Improving Fairness, Efficiency, and Stability in HTTP-Based Adaptive Video Streaming With Festive. *IEEE/ACM Transactions on Networking*, 22(1):326–340, 2014.

[30] T. Huang, R. Johari, N. McKeown, M. Trunnell, and M. Watson. A Buffer-based Approach to Rate Adaptation: Evidence from a Large Video Streaming Service. In *ACM SIGCOMM*, 2014.

[31] K. Spiteri, R. Urgaonkar, and R. K. Sitaraman. BOLA: Near-optimal bitrate adaptation for online videos. In *IEEE INFOCOM*, 2016.

[32] X. Yin, A. Jindal, V. Sekar, and B. Sinopoli. A Control-Theoretic Approach for Dynamic Adaptive Video Streaming over HTTP. In *ACM SIGCOMM*, 2015.

[33] T. Lohmar, T. Einarsson, P. Frojdh, F. Gabin, and M. Kampmann. Dynamic adaptive HTTP streaming of live content. In *IEEE WOWMOM*, 2011.

[34] Y. Shuai and T. Herfet. Towards Reduced Latency in Adaptive Live Streaming. In *IEEE CCNC*, 2018.

[35] Y. Shuai and T. Herfet. On Stabilizing Buffer Dynamics for Adaptive Video Streaming with a Small Buffering Delay. In *IEEE CCNC*, 2017.

[36] Y. Shuai and T. Herfet. A Buffer Dynamic Stabilizer for Low-Latency Adaptive Video Streaming. In *IEEE ICCE-Berlin*, 2016.

[37] Y. Shuai and T. Herfet. Improving User Experience in Low-Latency Adaptive Streaming by Stabilizing Buffer Dynamics. In *IEEE CCNC*, 2016.

[38] M. Gorius. *Adaptive Delay-constrained Internet Media Transport*. PhD thesis, Saarland University, December 2012.

[39] D. X. Wei, C. Jin, S. H. Low, and S. Hegde. FAST TCP: motivation, architecture, algorithms, performance. *IEEE/ACM Trans. on Netw.*, 14, 2006.

[40] Y. Shuai, M. Gorius, and T. Herfet. Low-Latency Dynamic Adaptive Video Streaming. In *IEEE BMSB*, 2014.

[41] Y. Shuai and T. Herfet. Server-Driven Rate Control for Adaptive Video Streaming using Virtual Client Buffers. In *IEEE ICCE-Berlin*, 2014.

[42] Y. Shuai, G. Petrovic, and T. Herfet. Open-Loop Rate Control for Adaptive Video Streaming. In *IEEE CCNC*, 2015.

[43] Y. Shuai, G. Petrovic, and T. Herfet. OLAC: an Open-Loop Controller for Low-Latency Adaptive Video Streaming. In *IEEE ICC*, 2015.

[44] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson. RTP: A Transport Protocol for Real-Time Applications. STD 64, Internet Engineering Task Force (IETF), July 2003.

[45] H. Schulzrinne, A. Rao, R. Lanphier, M. Westerlund, and M. Stiemerling. Real-Time Streaming Protocol Version 2.0. RFC 7826, Internet Engineering Task Force (IETF), December 2016.

[46] R. Rejaie, M. Handley, and D. Estrin. Quality Adaptation for Congestion Controlled Video Playback over the Internet. In *ACM SIGCOMM*, 1999.

[47] S. Floyd, M. Handley, J. Padhye, and J. Widmer. Equation-based congestion control for unicast applications. In *ACM SIGCOMM*, September 2000.

[48] C. Boutremans and J. Y. Le Boudec. Adaptive joint playout buffer and FEC adjustment for Internet telephony. In *IEEE INFOCOM*, March 2003.

[49] L. Zhang, S. Deering, D. Estrin, S. Shenker, and D. Zappala. RSVP: a new resource reservation protocol. *IEEE Communications Magazine*, 40(5):116–127, May 2002.

[50] C. Aurrecoechea, A. T. Campbell, and L. Hauw. A survey of QoS architectures. *Multimedia Systems*, 6:138–151, May 1998.

[51] X. Xiao and L.M. Ni. Internet QoS: a big picture. *IEEE network - the magazine of global internetworking*, 13(2):8–18, March 1999.

[52] Q. Zhang, W. Zhu, and Y. Zhang. Resource allocation for multimedia streaming over the Internet. *IEEE Transactions on Multimedia*, 3(3):339–355, September 2001.

[53] J. Carapinha, R. Bless, C. Werle, K. Miller, V. Dobrota, A. B. Rus, H. Grob-Lipski, and H. Roessler. Quality of service in the Future Internet. In *ITU-T Kaleidoscope: Beyond the Internet? - Innovations for Future Networks and Services*, Dec 2010.

[54] T. Hoßfeld, M. Seufert, M. Hirth, T. Zinner, P. Tran-Gia, and R. Schatz. Quantification of YouTube QoE via Crowdsourcing. In *IEEE International Symposium on Multimedia*, Dec 2011.

[55] F. Dobrian, V. Sekar, A. Awan, I. Stoica, D. Joseph, A. Ganjam, J. Zhan, and H. Zhang. Understanding the Impact of Video Quality on User Engagement. In *ACM SIGCOMM*, 2011.

[56] S. S. Krishnan and R. K. Sitaraman. Video Stream Quality Impacts Viewer Behavior: Inferring Causality Using Quasi-Experimental Designs. *IEEE/ACM Transactions on Networking*, 21(6):2001–2014, Dec 2013.

[57] S. Akhshabi, L. Anantakrishnan, Ali C. Begen, and C. Dovrolis. What happens when HTTP adaptive streaming players compete for bandwidth? In *ACM NOSSDAV*, 2012.

[58] T. Huang, N. Handigol, B. Heller, N. McKeown, and R. Johari. Confused, timid, and unstable: picking a video streaming rate is hard. In *ACM IMC*, 2012.

[59] T. Huang, R. Johari, and N. McKeown. Downton Abbey Without the Hiccups: Buffer-based Rate Adaptation for HTTP Video Streaming. In *ACM SIGCOMM*, FhMN Workshop, 2013.

[60] Z. Li, X. Zhu, J. Gahm, R. Pan, H. Hu, A. C. Begen, and D. Oran. Probe and Adapt: Rate Adaptation for HTTP Video Streaming At Scale. *IEEE Journal on Selected Areas in Communications*, 32(4):719–733, April 2014.

[61] S. Akhshabi, A. C. Begen, and C. Dovrolis. An experimental evaluation of rate-adaptation algorithms in adaptive streaming over HTTP. In *ACM Multimedia Systems (MMSys)*, pages 157–168, February 2011.

[62] ITU-T. H.262 : Information technology - Generic coding of moving pictures and associated audio information: Video, February 2012.

[63] ITU-T. H.264 : Advanced video coding for generic audiovisual services, April 2017.

[64] ITU-T. H.265 : High efficiency video coding, February 2018.

[65] J. Bankoski, J. Koleszar, L. Quillio, J. Salonen, P. Wilkins, and Y. Xu. VP8 Data Format and Decoding Guide. RFC 6386, Internet Engineering Task Force (IETF), November 2011.

[66] A. Grange, P. de Rivaz, and J. Hunt. VP9 Bitstream & Decoding Process Specification. Specification, Google Inc., March 2016.

[67] P. de Rivaz and J. Haughton. AV1 Bitstream & Decoding Process Specification. Specification, Alliance for Open Media, March 2018.

[68] S. Lederer, C. Müller, and C. Timmerer. Dynamic Adaptive Streaming over HTTP Dataset. In *ACM Multimedia Systems (MMSys)*, 2012.

[69] C. Liu, I. Bouazizi, and M. Gabbouj. Segment duration for rate adaptation of adaptive HTTP streaming. In *IEEE International Conference on Multimedia and Expo (ICME)*, July 2011.

[70] V. Adzic, H. Kalva, and B. Furht. Optimizing video encoding for adaptive streaming over HTTP. *IEEE Transactions on Consumer Electronics*, 58(2):397–403, May 2012.

[71] J. Lievens, S. M. Satti, N. Deligiannis, P. Schelkens, and A. Munteanu. Optimized segmentation of H.264/AVC video for HTTP adaptive streaming. In *IFIP/IEEE International Symposium on Integrated Network Management*, May 2013.

[72] M. Karczewicz and R. Kurceren. The SP- and SI-frames design for H.264/AVC. *IEEE Transactions on Circuits and Systems for Video Technology*, 13(7):637–644, July 2003.

[73] E. Setton and B. Girod. Video streaming with SP and SI frames. In *SPIE VCIP*, volume 5960, pages 2204–2211, July 2005.

[74] M. Seufert, 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*, 17(1):469–492, September 2014.

[75] 3GPP. TS 26.234 v9.1.0 - Transparent end-to-end Packet-switched Streaming Service (PSS); Protocols and codecs, 2009.

[76] ISO/IEC. ISO/IEC 14496-12:2015 - Coding of audio-visual objects – Part 12: ISO base media file format, 2015.

[77] ITU and ISO/IEC. ITU-T H.222.0 / ISO/IEC 13818-1:2015 - Generic coding of moving pictures and associated audio information – Part 1: Systems, 2015.

[78] C. Müller and C. Timmerer. A VLC media player plugin enabling dynamic adaptive streaming over HTTP. In *ACM Multimedia*, 2011.

[79] B. Rainer, S. Lederer, C. Müller, and C. Timmerer. A seamless Web integration of adaptive HTTP streaming. In *European Signal Processing Conference (EUSIPCO)*, August 2012.

[80] C. Müller, S. Lederer, J. Poecher, and C. Timmerer. Libdash - An open source software library for the MPEG-DASH standard. In *IEEE International Conference on Multimedia and Expo Workshops*, July 2013.

[81] L. De Cicco, S. Mascolo, and V. Palmisano. Feedback control for adaptive live video streaming. In *ACM Multimedia Systems (MMSys)*, 2011.

[82] K. Kim, B. Y. Cho, and W. W. Ro. Server Side, Play Buffer Based Quality Control for Adaptive Media Streaming. *Multimedia Tools Appl.*, 75(10):5397–5415, May 2016.

[83] O. El Marai and T. Taleb. Online Server-Side Optimization Approach for Improving QoE of DASH Clients. In *IEEE Global Communications Conference (GLOBECOM)*, pages 1–6, Dec 2017.

[84] M. K. Mukerjee, D. Naylor, J. Jiang, D. Han, S. Seshan, and H. Zhang. Practical, Real-time Centralized Control for CDN-based Live Video Delivery. In *ACM SIGCOMM*, 2015.

[85] R. K. P. Mok, X. Luo, E. W. W. Chan, and R. K. C. Chang. QDASH: A QoE-aware DASH System. In *ACM Multimedia Systems (MMSys)*, 2012.

[86] G. Cofano, 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. In *ACM Multimedia Systems (MMSys)*, 2016.

[87] E. Thomas. Applications and deployments of server and network assisted DASH (SAND). *IET Conference Proceedings*, pages 22 (8 .)–22 (8 .)(1), January 2016.

[88] A. Bentaleb, A. C. Begen, and R. Zimmermann. SDNDASH: Improving QoE of HTTP Adaptive Streaming Using Software Defined Networking. In *ACM Multimedia (MM)*, 2016.

[89] J. W. Kleinrouweler, S. Cabrero, R. van der Mei, and P. Cesar. A model for evaluating sharing policies for network-assisted HTTP adaptive streaming. *Computer Networks*, 109:234 – 245, 2016.

[90] D. Bhat, A. Rizk, M. Zink, and R. Steinmetz. Network Assisted Content Distribution for Adaptive Bitrate Video Streaming. In *ACM Multimedia Systems (MMSys)*, 2017.

[91] P. Le Callet, S. Möller, and A. Perkis. Qualinet White Paper on Definitions of Quality of Experience. Whitepaper, European Network on Quality of Experience in Multimedia Systems and Services (QUALINET), March 2013.

[92] ITU-T. Recommendation ITU-T P.10/G.100 : Vocabulary for Performance, Quality of Service and Quality of Experience, 2017.

[93] W. Lin and C.-C. J. Kuo. Perceptual visual quality metrics: A survey. *Journal of Visual Communication and Image Representation*, 22(4):297 – 312, 2011.

[94] S. Chikkerur, V. Sundaram, M. Reisslein, and L. J. Karam. Objective Video Quality Assessment Methods: A Classification, Review, and Performance Comparison. *IEEE Transactions on Broadcasting*, 57(2):165–182, June 2011.

[95] A. K. Moorthy, L. K. Choi, A. C. Bovik, and G. de Veciana. Video quality assessment on mobile devices: Subjective, behavioral and objective studies. *IEEE Journal of Selected Topics in Signal Processing*, 6(6):652–671, Oct 2012.

[96] Y. Chen, K. Wu, and Q. Zhang. From QoS to QoE: A Tutorial on Video Quality Assessment. *IEEE Communications Surveys Tutorials*, 17(2):1126–1165, Secondquarter 2015.

[97] A. Balachandran, V. Sekar, A. Akella, S. Seshan, I. Stoica, and H. Zhang. Developing a Predictive Model of Quality of Experience for Internet Video. In *ACM SIGCOMM*, 2013.

[98] M. Z. Shafiq, J. Erman, L. Ji, A. X. Liu, J. Pang, and J. Wang. Understanding the Impact of Network Dynamics on Mobile Video User Engagement. In *ACM SIGMETRICS*, 2014.

[99] U. Reiter, K. Brunnström, K. De Moor, M. Larabi, M. Pereira, A. Pinheiro, J. You, and A. Zgank. *Factors Influencing Quality of Experience*, pages 55–72. Springer International Publishing, Cham, 2014.

[100] ITU-T. Recommendation ITU-T P.800 : Methods for Subjective Determination of Transmission Quality, 1996.

[101] ITU-T. Recommendation ITU-T P.913 : Methods for the Subjective Assessment of Video Quality, Audio Quality and Audiovisual Quality of Internet Video and Distribution Quality Television in any Environment, 2016.

[102] VQEG and ITU-T. Tutorial for Objective perceptual assessment of video quality: Full reference television. Handbook, Video Quality Experts Group (VQEG) and International Telecommunication Union (ITU), 2004.

[103] P. Seeling and M. Reisslein. Video Transport Evaluation With H.264 Video Traces. *IEEE Communications Surveys Tutorials*, 14(4):1142–1165, Fourth 2012.

[104] J. R. Ohm, G. J. Sullivan, H. Schwarz, T. K. Tan, and T. Wiegand. Comparison of the Coding Efficiency of Video Coding Standards – Including High Efficiency Video Coding (HEVC). *IEEE Transactions on Circuits and Systems for Video Technology*, 22(12):1669–1684, Dec 2012.

[105] A. Pulipaka, P. Seeling, M. Reisslein, and L. J. Karam. Traffic and Statistical Multiplexing Characterization of 3-D Video Representation Formats. *IEEE Transactions on Broadcasting*, 59(2):382–389, June 2013.

[106] A. M. Eskicioglu and P. S. Fisher. Image quality measures and their performance. *IEEE Transactions on Communications*, 43(12):2959–2965, Dec 1995.

[107] Z. Wang and A. C. Bovik. A universal image quality index. *IEEE Signal Processing Letters*, 9(3):81–84, March 2002.

[108] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli. Image quality assessment: from error visibility to structural similarity. *IEEE Transactions on Image Processing*, 13(4):600–612, April 2004.

[109] M.H. Pinson and S. Wolf. A new standardized method for objectively measuring video quality. *IEEE Transactions on Broadcasting*, 50(3):312–322, 2004.

[110] F. Yang, S. Wan, Q. Xie, and H. R. Wu. No-Reference Quality Assessment for Networked Video via Primary Analysis of Bit Stream. *IEEE Transactions on Circuits and Systems for Video Technology*, 20(11):1544–1554, Nov 2010.

[111] M. Venkataraman and M. Chatterjee. Inferring video QoE in real time. *IEEE Network*, 25(1):4–13, January 2011.

[112] A. Khan, L. Sun, and E. Ifeachor. QoE Prediction Model and its Application in Video Quality Adaptation Over UMTS Networks. *IEEE Transactions on Multimedia*, 14(2):431–442, April 2012.

[113] M. A. Saad, A. C. Bovik, and C. Charrier. Blind Prediction of Natural Video Quality. *IEEE Transactions on Image Processing*, 23(3):1352–1365, March 2014.

[114] D. C. Robinson, Y. Jutras, and V. Craciun. Subjective Video Quality Assessment of HTTP Adaptive Streaming Technologies. *Bell Lab. Tech. J.*, 16(4):5–23, March 2012.

[115] O. Oyman and S. Singh. Quality of experience for HTTP adaptive streaming services. *IEEE Communications Magazine*, 50(4):20–27, April 2012.

[116] W. Song and D. W. Tjondronegoro. Acceptability-Based QoE Models for Mobile Video. *IEEE Transactions on Multimedia*, 16(3):738–750, April 2014.

[117] Y. Liu, S. Dey, F. Ulupinar, M. Luby, and Y. Mao. Deriving and Validating User Experience Model for DASH Video Streaming. *IEEE Transactions on Broadcasting*, 61(4):651–665, Dec 2015.

[118] E. Baik, A. Pande, C. Stover, and P. Mohapatra. Video acuity assessment in mobile devices. In *IEEE Conference on Computer Communications (INFOCOM)*, April 2015.

[119] S. Tasaka. Bayesian Hierarchical Regression Models for QoE Estimation and Prediction in Audiovisual Communications. *IEEE Transactions on Multimedia*, 19(6):1195–1208, June 2017.

[120] Z. Duanmu, K. Ma, and Z. Wang. Quality-of-Experience of Adaptive Video Streaming: Exploring the Space of Adaptations. In *ACM Multimedia (MM)*, 2017.

[121] T. De Pessemier, K. De Moor, W. Joseph, L. De Marez, and L. Martens. Quantifying the Influence of Rebuffering Interruptions on the User's Quality of Experience During Mobile Video Watching. *IEEE Transactions on Broadcasting*, 59(1):47–61, March 2013.

[122] A. Ahmed, Z. Shafiq, H. Bedi, and A. Khakpour. Suffering from buffering? Detecting QoE impairments in live video streams. In *IEEE International Conference on Network Protocols (ICNP)*, Oct 2017.

[123] How Consumers Judge Their Viewing Experience. Report, Conviva, 2015.

[124] How Akamai Defines and Measures Online Video Quality. Whitepaper, Akamai, 2016. https://www.akamai.com/us/en/multimedia/documents/content/white-paper/how-akamai-defines-and-measures-online-video-quality-white-paper.pdf.

[125] P. Ni, R. Eg, A. Eichhorn, C. Griwodz, and P. Halvorsen. Flicker Effects in Adaptive Video Streaming to Handheld Devices. In *ACM Multimedia (MM)*, 2011.

[126] M. N. Garcia, F. De Simone, S. Tavakoli, N. Staelens, S. Egger, K. Brunnström, and A. Raake. Quality of experience and HTTP adaptive streaming: A review of subjective studies. In *International Workshop on Quality of Multimedia Experience (QoMEX)*, pages 141–146, Sept 2014.

[127] N. Staelens, S. Moens, W. Van den Broeck, I. Marien, B. Vermeulen, P. Lambert, R. Van de Walle, and P. Demeester. Assessing Quality of Experience of IPTV and Video on Demand Services in Real-Life Environments. *IEEE Transactions on Broadcasting*, 56(4):458–466, Dec 2010.

[128] D. Zegarra Rodríguez, R. Lopes Rosa, E. Costa Alfaia, J. Issy Abrahão, and G. Bressan. Video Quality Metric for Streaming Service Using DASH Standard. *IEEE Transactions on Broadcasting*, 62(3):628–639, Sept 2016.

[129] H. Riiser, P. Vigmostad, C. Griwodz, and P. Halvorsen. Commute Path Bandwidth Traces from 3G Networks: Analysis and Applications. In *ACM Multimedia Systems (MMSys)*, 2013.

[130] X. K. Zou, J. Erman, V. Gopalakrishnan, E. Halepovic, R. Jana, X. Jin, J. Rexford, and R. K. Sinha. Can Accurate Predictions Improve Video Streaming in Cellular Networks? In *ACM HotMobile*, 2015.

[131] J. Jiang, V. Sekar, and H. Zhang. Improving fairness, efficiency, and stability in HTTP-based adaptive video streaming with FESTIVE. In *ACM CoNEXT*, 2012.

[132] R. Prasad, C. Dovrolis, M. Murray, and K. Claffy. Bandwidth Estimation: Metrics, Measurement Techniques, and Tools. *Netwrk. Mag. of Global Internetwkg.*, 17(6):27–35, 2003.

[133] M. Mirza, J. Sommers, P. Barford, and X. Zhu. A Machine Learning Approach to TCP Throughput Prediction. *IEEE/ACM Trans. Netw.*, 18(4):1026–1039, 2010.

[134] A. Balachandran, V. Sekar, A. Akella, S. Seshan, I. Stoica, and H. Zhang. A Quest for an Internet Video Quality-of-experience Metric. In *ACM HotNets*, 2012.

[135] Y. Liu, S. Dey, D. Gillies, F. Ulupinar, and M. Luby. User experience modeling for DASH video. In *IEEE Packet Video Workshop (PV)*, 2013.

[136] P. Romirer-Maierhofer, A. Coluccia, and T. Witek. On the Use of TCP Passive Measurements for Anomaly Detection: A Case Study from an Operational 3G Network. In *Traffic Monitoring and Analysis*, pages 183–197. Springer Berlin Heidelberg, 2010.

[137] Z. Hu, Y. Chen, L. Qiu, G. Xue, H. Zhu, N. Zhang, C. He, L Pan, and C. He. An In-depth Analysis of 3G Traffic and Performance. In *the 5th Workshop on All Things Cellular: Operations, Applications and Challenges*, 2015.

[138] M. Carbone and L. Rizzo. Dummynet revisited. *ACM SIGCOMM Comput. Commun. Rev.*, 40(2):12–20, 2010.

[139] M. Gorius, Y. Shuai, and T. Herfet. Predictably Reliable Media Transport over Wireless Home Networks. In *IEEE CCNC*, 2012.

[140] D. Rubenstein, J. Kurose, and D. Towsley. A study of proactive hybrid FEC/ARQ and scalable feedback techniques for reliable, real-time multicast. *International Journal for the Computer and Telecommunications Industry*, 24:563–574, 2001.

[141] M. Gorius, Y. Shuai, and T. Herfet. Dynamic Media Streaming under Predictable Reliability. In *IEEE BMSB*, 2012.

[142] M. Gorius, Y. Shuai, and T. Herfet. Dynamic Media Streaming with Predictable Reliability and Opportunistic TCP-Friendliness. In *IEEE CCNC*, 2013.

[143] S. Ha, I. Rhee, and L. Xu. CUBIC: a new TCP-friendly high-speed TCP variant. *ACM SIGOPS operating systems review*, 42:64–74, July 2008.

[144] C. Müller, S. Lederer, and C. Timmerer. An Evaluation of Dynamic Adaptive Streaming over HTTP in Vehicular Environments. In *ACM Workshop on Mobile Video (MoVid)*, pages 37–42, 2012.

[145] A. Zabrovskiy, E. Petrov, E. Kuzmin, and C. Timmerer. Evaluation of the Performance of Adaptive HTTP Streaming Systems. *ArXiv e-prints*, October 2017.

[146] C. Timmerer, M. Maiero, and B. Rainer. Which Adaptation Logic? An Objective and Subjective Performance Evaluation of HTTP-based Adaptive Media Streaming Systems. *ArXiv e-prints*, June 2016.

[147] K. Evensen, D. Kaspar, C. Griwodz, P. Halvorsen, A. F. Hansen, and P. Engelstad. Using Bandwidth Aggregation to Improve the Performance of Quality-adaptive Streaming. *Image Commun.*, 27(4):312–328, April 2012.

[148] S. Wei and V. Swaminathan. Low Latency Live Video Streaming over HTTP 2.0. In *ACM NOSSDAV*, 2014.

[149] R. Huysegems, J. van der Hooft, T. Bostoen, P. Rondao Alface, S. Petrangeli, T. Wauters, and F De Turck. HTTP/2-Based Methods to Improve the Live Experience of Adaptive Streaming. In *ACM Multimedia (MM)*, 2015.

[150] K. Miller, A.-K. Al-Tamimi, and A. Wolisz. QoE-Based Low-Delay Live Streaming Using Throughput Predictions. *ACM Trans. Multimedia Comput. Commun. Appl.*, 13(1):4:1–4:24, October 2016.

[151] M. Belshe, R. Peon, and M. Thomson. Hypertext Transfer Protocol Version 2 (HTTP/2). RFC 7540, Internet Engineering Task Force (IETF), May 2015.

[152] Y. Liu and J. Y. B. Lee. An Empirical Study of Throughput Prediction in Mobile Data Networks. In *IEEE Global Communications Conference (GLOBECOM)*, Dec 2015.

[153] G. Tian and Y. Liu. Towards Agile and Smooth Video Adaptation in HTTP Adaptive Streaming. *IEEE/ACM Transactions on Networking*, 24(4):2386–2399, Aug 2016.

[154] A. Sobhani, A. Yassine, and S. Shirmohammadi. A Video Bitrate Adaptation and Prediction Mechanism for HTTP Adaptive Streaming. *ACM Trans. Multimedia Comput. Commun. Appl.*, 13(2):18:1–18:25, March 2017.

[155] W. Cherif, Y. Fablet, E. Nassor, J. Taquet, and Y. Fujimori. DASH Fast Start Using HTTP/2. In *ACM NOSSDAV*, 2015.

[156] Y. Sun, X. Yin, J. Jiang, V. Sekar, F. Lin, N. Wang, T. Liu, and B. Sinopoli. CS2P: Improving Video Bitrate Selection and Adaptation with Data-Driven Throughput Prediction. In *ACM SIGCOMM*, 2016.

[157] H. Riiser, T. Endestad, P. Vigmostad, C. Griwodz, and P. Halvorsen. Video Streaming Using a Location-based Bandwidth-lookup Service for Bitrate Planning. *ACM Trans. Multimedia Comput. Commun. Appl.*, 8(3):24:1–24:19, 2012.

[158] M. Xiao, V. Swaminathan, S. Wei, and S. Chen. DASH2M: Exploring HTTP/2 for Internet Streaming to Mobile Devices. In *ACM Multimedia (MM)*, 2016.

[159] P.J. Kaufman. *Smarter Trading: Improving Performance in Changing Markets*. McGraw-Hill, 1995.

[160] A. J. Smola and B. Schölkopf. A tutorial on support vector regression. *Statistics and Computing*, 14(3):199–222, Aug 2004.

[161] I. Fette and A. Melnikov. The WebSocket Protocol. RFC 6455, Internet Engineering Task Force (IETF), December 2011.

[162] H. Mao, R. Netravali, and M. Alizadeh. Neural Adaptive Video Streaming with Pensieve. In *ACM SIGCOMM*, 2017.

[163] Z. Li, A. C. Begen, J. Gahm, Y. Shan, B. Osler, and D. Oran. Streaming Video over HTTP with Consistent Quality. In *ACM Multimedia Systems (MMsys)*, 2014.

[164] J. M. Batalla, P. Krawiec, A. Beben, P. Wisniewski, and A. Chydzinski. Adaptive Video Streaming: Rate and Buffer on the Track of Minimum Rebuffering. *IEEE Journal on Selected Areas in Communications*, 34(8):2154–2167, Aug 2016.

[165] A. Beben, P. Wiśniewski, J. M. Batalla, and P. Krawiec. ABMA+: Lightweight and Efficient Algorithm for HTTP Adaptive Streaming. In *ACM Multimedia Systems (MMSys)*, 2016.

[166] P. K. Yadav, A. Shafiei, and W. T. Ooi. QUETRA: A Queuing Theory Approach to DASH Rate Adaptation. In *ACM Multimedia (MM)*, 2017.

[167] C. Liu, I. Bouazizi, and M. Gabbouj. Rate Adaptation for Adaptive HTTP Streaming. In *ACM Multimedia Systems (MMSys)*, 2011.

[168] M. J. Neely. *Stochastic Network Optimization with Application to Communication and Queueing Systems*. Synthesis Lectures on Communication Networks. Morgan and Claypool Publishers, 2010.

[169] S. Liu and J. Y. L. Forrest. *Grey Systems: Theory and Applications*. Understanding Complex Systems 68. Springer-Verlag Berlin Heidelberg, 2011.

[170] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu. Asynchronous Methods for Deep Reinforcement Learning. In *International Conference on Machine Learning*, Jun 2016.

[171] R. Stewart. Stream Control Transmission Protocol. RFC 4960, Internet Engineering Task Force (IETF), September 2007.

[172] E. Kohler, M. Handley, and S. Floyd. Datagram Congestion Control Protocol (DCCP). RFC 4340, Internet Engineering Task Force (IETF), March 2006.

[173] R. Hamilton, J. Iyengar, I. Swett, and A. Wilk. QUIC: A UDP-Based Secure and Reliable Transport for HTTP/2. Internet-Draft draft-tsvwg-quic-protocol-02, IETF Secretariat, January 2016.

[174] S. McQuistin, C. Perkins, and M. Fayed. TCP Goes to Hollywood. In *ACM NOSSDAV*, 2016.

[175] D. Bhat, A. Rizk, and M. Zink. Not So QUIC: A Performance Study of DASH over QUIC. In *ACM NOSSDAV*, 2017.

[176] M. Handley. Why the internet only just works. *BT Technology Journal*, 24(3):119–129, Jul 2006.

[177] S. Hätönen, A. Nyrhinen, L. Eggert, S. Strowes, P. Sarolahti, and M. Kojo. An Experimental Study of Home Gateway Characteristics. In *ACM IMC*, 2010.

[178] T. Henderson, S. Floyd, A. Gurtov, and Y. Nishida. The NewReno Modification to TCP's Fast Recovery Algorithm. RFC 6582, Internet Engineering Task Force (IETF), April 2012.

[179] Improving Online Video Quality and Accelerating Downloads. Whitepaper, Akamai, 2015. http://www.akamai.com/dl/akamai/Akamai-Improving-Online-Video-Quality-and-Accelerating-Downloads.pdf.

[180] N. Cardwell, Y. Cheng, C. S. Gunn, S. H. Yeganeh, and V. Jacobson. BBR: Congestion-Based Congestion Control. *Queue*, 14(5):50:20–50:53, October 2016.

[181] K. Tan, J. Song, Q. Zhang, and M. Sridharan. A Compound TCP Approach for High-Speed and Long Distance Networks. In *IEEE INFOCOM*, April 2006.

[182] X. Li, S. Paul, and M. Ammar. Layered video multicast with retransmissions (LVMR): evaluation of hierarchical rate control. In *IEEE INFOCOM*, Mar 1998.

[183] H. Schwarz, D. Marpe, and T. Wiegand. Overview of the Scalable Video Coding Extension of the H.264/AVC Standard. *IEEE Transactions on Circuits and Systems for Video Technology*, 17(9):1103–1120, Sept 2007.

[184] H. Kalva, V. Adzic, and B. Furht. Comparing MPEG AVC and SVC for adaptive HTTP streaming. In *IEEE International Conference on Consumer Electronics (ICCE)*, Jan 2012.

[185] J. Famaey, S. Latré, N. Bouten, W. Van de Meerssche, B. De Vleeschauwer, W. Van Leekwijck, and F. De Turck. On the merits of SVC-based HTTP Adaptive Streaming. In *IFIP/IEEE International Symposium on Integrated Network Management (IM)*, May 2013.

[186] M. Tanimoto, M. P. Tehrani, T. Fujii, and T. Yendo. Free-Viewpoint TV. *IEEE Signal Processing Magazine*, 28(1):67–76, Jan 2011.

[187] P. Merkle, A. Smolic, K. Muller, and T. Wiegand. Efficient Prediction Structures for Multiview Video Coding. *IEEE Transactions on Circuits and Systems for Video Technology*, 17(11):1461–1473, Nov 2007.

[188] A. Vetro, T. Wiegand, and G. J. Sullivan. Overview of the Stereo and Multiview Video Coding Extensions of the H.264/MPEG-4 AVC Standard. *Proceedings of the IEEE*, 99(4):626–642, April 2011.

[189] X. Liu, F. Dobrian, H. Milner, J. Jiang, V. Sekar, I. Stoica, and H. Zhang. A Case for a Coordinated Internet Video Control Plane. *SIGCOMM Comput. Commun. Rev.*, 42(4):359–370, August 2012.

[190] A. Balachandran, V. Sekar, A. Akella, and S. Seshan. Analyzing the Potential Benefits of CDN Augmentation Strategies for Internet Video Workloads. In *ACM IMC*, 2013.