Llama: Towards Low Latency Live Adaptive Streaming

Tomasz Lyko, BSc Hons

School of Computing and Communications

Lancaster University

A thesis submitted for the degree of

Doctor of Philosophy

August, 2023

Declaration

I declare that the work presented in this thesis is, to the best of my knowledge and belief, original and my own work. The material has not been submitted, either in whole or in part, for a degree at this, or any other university. This thesis does not exceed the maximum permitted word length of 80,000 words including appendices and footnotes, but excluding the bibliography. A rough estimate of the word count is: 44776

Tomasz Lyko

Llama: Towards Low Latency Live Adaptive Streaming

Tomasz Lyko, BSc Hons.

School of Computing and Communications, Lancaster University

A thesis submitted for the degree of *Doctor of Philosophy.* August, 2023

Abstract

Multimedia streaming, including on-demand and live delivery of content, has become the largest service, in terms of traffic volume, delivered over the Internet. The ever-increasing demand has led to remarkable advancements in multimedia delivery technology over the past three decades, facilitated by the concurrent pursuit of efficient and quality encoding of digital media. Today, the most prominent technology for online multimedia delivery is HTTP Adaptive Streaming (HAS), which utilises the stateless HTTP architecture - allowing for scalable streaming sessions that can be delivered to millions of viewers around the world using Content Delivery Networks. In HAS, the content is encoded at multiple encoding bitrates, and fragmented into segments of equal duration. The client simply fetches the consecutive segments from the server, at the desired encoding bitrate determined by an ABR algorithm which measures the network conditions and adjusts the bitrate accordingly. This method introduces new challenges to live streaming, where the content is generated in real-time, as it suffers from high end-to-end latency when compared to traditional broadcast methods due to the required buffering at client.

This thesis aims to investigate low latency live adaptive streaming, focusing on the reduction of the end-to-end latency. We investigate the impact of latency on the performance of ABR algorithms in low latency scenarios by developing a simulation model and testing prominent on-demand adaptation solutions. Additionally, we conduct extensive subjective testing to further investigate the impact of bitrate changes on the perceived Quality of Experience (QoE) by users. Based on these investigations, we design an ABR algorithm suitable for low latency scenarios which can operate with a small client buffer. We evaluate the proposed low latency adaption solution against on-demand ABR algorithms and the state-of-the-art low latency ABR algorithms, under realistic network conditions using a variety of client and latency settings.

Publications

Tomasz Lyko et al. "Evaluation of CMAF in Live Streaming Scenarios". In: *Proceedings of the 30th ACM Workshop on Network and Operating Systems Support for Digital Audio and Video*. NOSSDAV '20. Istanbul, Turkey: Association for Computing Machinery, 2020, pp. 21–26. ISBN: 9781450379458. DOI: 10.1145/3386290.3396932

T. Lyko et al. "Llama - Low Latency Adaptive Media Algorithm". In: *2020 IEEE International Symposium on Multimedia (ISM)*. 2020, pp. 113–121. DOI: 10.1109/ISM.2020.00027

Tomasz Lyko et al. "Improving quality of experience in adaptive low latency live streaming". In: *Multimedia Tools and Applications* (July 2023). ISSN: 1380-7501. DOI: 10.1007/s11042-023-15895-9

Acknowledgements

I would like to thank my PhD supervisors: Dr Nicholas Race and Dr Matthew Broadbent, for their invaluable guidance and support throughout my PhD. I would also like to thank Mike Nilsson, Paul Farrow, and Steve Appleby for their input and assistance during this project. I also thank the UK Engineering and Physical Sciences Research Council (EPSRC) and British Telecom (BT) for funding this work.

And most importantly, I would like to thank my friends and family, for their unconditional support and encouragement throughout this entire time.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Multimedia streaming has become the main service delivered over the Internet, dominating the bandwidth of Internet Service Providers. Online multimedia streaming continues to grow, with Cisco predicting that it will account for 82% of all IP traffic by 2022, an increase of 7% from 2017 [21]. While the majority of multimedia streaming consists of on-demand delivery, the forecast predicts that live multimedia streaming will increase as well, to 17% of all video traffic by 2022. These forecasts indicate that the demand for multimedia streaming over the Internet will continue to grow, including both on-demand as well live broadcasts.

The ever-increasing demand for multimedia streaming has led to major advancements in multimedia delivery technologies, as well as multimedia compression, over the past three decades. With the intention of matching the increasing demand, multimedia delivery methods have evolved to become scalable and cost-effective. Today, multimedia content is segmented and served using standard HTTP servers, allowing for the use of scalability solutions such as Content Delivery Networks. This method, being a relatively recent change, consists of a trade-off between encoding efficiency and scalability.

While online multimedia delivery technologies have evolved with scalability and cost-effectiveness in mind, their main focus has been on-demand delivery, where the content is already fully generated prior to the streaming session start, as it is the common method of content delivery. In the context of live streaming, where the content is generated in real-time and only certain parts of the content are available at the beginning of a streaming session, this introduces new challenges which are further amplified in the context of low latency live adaptive streaming.

Currently, typical online live streaming broadcasts are significantly behind the traditional broadcasting methods, experiencing a latency of 60-90s as opposed to the latency of 4-6s of terrestrial and satellite television. This latency gap can have a significant negative impact on the perceived quality of online live transmissions as it can lead to situations in which the user can be notified of the content ahead from sources where latency is considerably lower. For example, a user could hear their neighbour cheering for a goal in a football match a minute before they can experience it on their screen, if the neighbour is watching the same match using terrestrial or satellite television while the user is watching an online broadcast.

Achieving low latency in live adaptive streaming requires a small client video buffer, which can impact the performance of the adaptation logic employed, in turn leading to a challenging balancing act between minimising the latency and maximising the Quality of Experience.

1.1 Multimedia Delivery over the Internet

The way in which multimedia is delivered to the user over the Internet has changed dramatically over the past three decades. The Internet, being a best effort-network, presents unique challenges to reliable multimedia delivery. First, the bandwidth of the link between the broadcasting source and the client is not static and can change significantly over time, as opposed to traditional broadcasting methods such as terrestrial or satellite television, where the bandwidth of the link remains constant. Second, the links between the broadcasting source and each individual client are not identical, meaning, one client might be capable of receiving the broadcast at a much higher quality than others, while in traditional broadcasting methods all clients receive the broadcast at the same quality, creating a uniform experience across the clients.

One of the major advancements in the past three decades has been multimedia compression, which, is a critical element in the evolution of multimedia delivery systems. Multimedia compression focuses on the encoding of video and audio in a manner that results in fewer bits required to represent the digital media elements. Over the years, the compression has improved significantly, in terms of quality as well as efficiency, mainly driven by the increase in computational power available in consumer electronics. Today, the most common encoding formats enable efficient use of bandwidth, as well as, support packet-switched networks such as the Internet.

Initial delivery methods consisted of the content being transmitted using a stateful architecture, between a server and a client. In order to begin a streaming session, the client would connect to the server, after which the server would begin to transmit video and audio. The client would control the streaming session by issuing commands to the server, such as Pause and Play. In this architecture, the server was required to keep the state of each client and streaming session, leading to scalability issues as the demand for multimedia streaming increased.

Subsequent advancements in multimedia delivery over the Internet aimed to solve the issue of scalability. Multicast broadcasts utilised the IP multicast to increase the efficiency of the transmitted data, as it allowed the broadcast to be transmitted by the server once and then propagated by the network to all of the subscribed clients. This delivery method introduced new challenges, as individual clients could not control the entire streaming session, leading to issues with the re-transmission of corrupt data. Additionally, IP multicast required specialised hardware, hindering its deployment on the Internet.

Peer-to-peer streaming was another technology designed to ease the scalability issues of multimedia delivery over the Internet. It consisted of a decentralised architecture, made up of equal peers capable of receiving and transmitting multimedia data in the streaming session. The server would serve the content to only a subset of peers, which then would consume as well as re-send the content to other peers. Similarly to multicasts, peer-to-peer suffered from issues with re-transmission of data. Additionally, the bandwidth between pairs of peers could differ significantly, with some unable to transmit the content in a timely manner leading to reliability issues down the chain.

1.1.1 HTTP Adaptive Streaming

HTTP Adaptive Streaming has become the most common method for multimedia delivery over the Internet. In this stateless architecture, most of the control is shifted to the client. The content is encoded into multiple quality bitrates, each requiring different bandwidth, and divided into short segments of equal duration. The quality bitrate can be changed at the segment level, as segments of different quality bitrates are interchangeable. The resulting segments are served on a standard HTTP server. In order to begin a streaming session, the client first fetches a manifest file, using a standard HTTP request, which describes the available content, including the segment duration as well as the quality bitrates and their bandwidth requirements. Once the manifest file has been fetched, the client proceeds to fetch the content, segment by segment, by issuing HTTP requests. Additionally, the client usually employs an ABR algorithm which monitors the network conditions and selects an appropriate quality bitrate for each segment. Stateless architecture combined with the utilisation of standard HTTP servers makes this solution scalable and cost-effective, as the content can be distributed using services such as Content Delivery Networks.

1.1.2 Low Latency Live Adaptive Streaming

Adaptive streaming enables the content to be delivered in both: on-demand as well as live mode. In live mode, the content is generated in real-time and hence only some of it is available at the beginning of a streaming session. The manifest file is extended to indicate the time at which each segment becomes available, that is, the time at which the segment is fully generated and reachable on the server. Just as in on-demand mode, the client requests content, segment by segment, however, in live mode the client issues a HTTP request for a segment only once the segment has become available on the server. Since segments are of equal duration, they become available on the server periodically, every segment duration. A streaming session begins when the server starts to produce content, after which the client can join the session, based on its target latency setting which dictates the desired time behind the live edge. In order to achieve low latency, the target latency setting must be minimised, resulting in the client following the stream much more closely. Additionally, to facilitate low latency settings, the segment duration must be significantly reduced, introducing encoding overhead as well as network overhead due to the increased number of segment requests.

1.2 Motivation

HTTP Adaptive Streaming presents new challenges to live broadcasts, which are further magnified in low latency scenarios. In adaptive streaming, the content is encoded into multiple quality bitrates, each with different bandwidth requirements, and divided into segments of equal duration. The client fetches the content, segment by segment, by issuing standard HTTP requests while utilising an ABR algorithm to select the appropriate quality bitrate of each segment. Additionally, the client employs a video buffer, which holds already fetched segments queued for playback, in order to give the client enough time to adapt the quality bitrate, at which the segments are fetched, to changes in network conditions without playback interruptions.

In live adaptive streaming, that is when the content is generated in real-time, the client suffers from high latency due to the required client video buffer. In order to fill the client buffer to a specific duration, the client needs to join a streaming session at least this duration since the session's beginning, as the client can only fetch segments that have already been generated. In live adaptive streaming, segments become available periodically, every segment duration. On the other hand, reducing the desired client video buffer size, which in turn will result in lower latency, can lead to degradation in the overall Quality of Experience as the ABR algorithm employed by the client might struggle to adapt the quality bitrate, when the network conditions deteriorate, in a timely manner, that is, before the client video buffer is fully depleted and consequently the playback is interrupted.

Low latency adaptive streaming further escalates the issue of the limited client video buffer. The target latency, the time between a segment being generated and presented to the user, must remain low, typically under six seconds. This results in an even more restricted client video buffer, presenting challenging conditions for the ABR algorithm as it now needs to be able to detect changes in network conditions and counteract them by changing the quality bitrate, in a significantly limited amount of time. Failure to do so will lead to a significant deterioration in the overall Quality of Experience as the playback might be interrupted. Current ABR algorithms have been designed with on-demand streaming in mind, as it is the main application of multimedia streaming over the Internet, and have not been tested in low latency scenarios. Live, especially low latency, streaming might require specialised ABR algorithms as they present unique conditions and constraints.

Recently, the Common Media Application Format (CMAF) has been standardised which can enhance low latency live adaptive streaming, as it allows for segments to be further divided into chunks of equal duration. Each chunk can be played out as soon as received by the client, reducing the periodical content generation time from a segment duration to a chunk duration. The quality bitrate can still be changed at the segment level only, reducing the encoding overhead significantly. When combined with HTTP/1.1 Chunked Transfer Encoding (CTE), the transport overhead can be significantly reduced as the client will continue to request each segment once, after which the first chunk can be fetched, and the remaining chunks will be transmitted by the server as they become available - without additional HTTP requests. The impact of CMAF on ABR algorithms, and by extension on the overall Quality of Experience in low latency live adaptive streaming has not been investigated.

1.3 Thesis Aims & Research Question

This thesis aims to explore the following research question:

In live adaptive streaming, what impact does the requirement of low latency have on the client buffer, and in turn on the design of the adaptation algorithm required to ensure optimal Quality of Experience?

The work required to answer this research question can be divided in the following four steps:

1. Investigate the impact of latency on the Quality of Experience in low latency live adaptive streaming. Challenging conditions presented by low latency in live adaptive streaming can have a significant impact on the performance of the ABR algorithm employed, in turn having a significant impact on the overall Quality of Experience of a streaming session. The relationship between latency and ABR algorithm performance needs to be investigated in order to determine the suitability of state-of-the-art adaptation solutions for low latency live adaptive streaming. Additionally, the results of such investigation can be utilised to inform design decisions of an ABR algorithm constructed specifically to operate in low latency conditions.

2. Investigate the impact of changes in playback bitrate on the Quality of Experience in adaptive streaming. Playback bitrate can be changed at segment level in adaptive streaming, meaning, it can be changed every segment duration by any number of encoding bitrates. This can result in highly irregular quality bitrate which can have great repercussions to the overall Quality of Experience. The number of possible quality bitrate patterns grows exponentially with the number of encoding bitrates employed, making it an extremely challenging research topic. Taking current literature into account, additional video quality bitrate patterns need to be investigated to improve our understanding of the relationship between the changes in quality bitrate and the overall Quality of Experience.

3. Design the necessary adaptation algorithm for low latency live adaptive streaming. Since low latency live adaptive streaming presents unique conditions and constraints to ABR algorithm performance, what adaptation logic needs to be employed in such conditions to improve the overall Quality of Experience while trying to minimise the average latency of a streaming session? The challenging balancing act of trying to minimise the average latency while trying to maximise the overall Quality of Experience of a streaming session might require a new ABR algorithm to be created, one that is compatible with prominent methods of low latency live adaptive streaming over the Internet. This thesis aims to develop a new ABR algorithm, designed to improve the Quality of Experience in low latency live adaptive streaming.

4. Evaluate the performance of the proposed low latency adaptation algorithm against state-of-the-art ABR algorithms. A newly proposed ABR algorithm must be evaluated against existing state-of-the-art adaptation solutions. This will require the testing of said ABR algorithms under multiple client configurations in a variety of realistic network conditions, as both factors can have a significant impact on the performance of ABR algorithms. This thesis aims to perform an extensive evaluation of the proposed ABR algorithm in order to determine its suitability for low latency live adaptive streaming scenarios, compared to the state-of-the-art adaption solutions.

1.4 Thesis Contributions

This thesis investigates low latency live adaptive streaming with the intention of improving the state-of-the-art delivery of multimedia generated in real-time and delivered to the client in low latency conditions. The main contributions of this thesis are as follows.

1. Investigation of low latency live adaptive streaming and the impact of latency on ABR algorithm performance. This thesis investigates the two main delivery methods of low latency live adaptive streaming, and how the reduced target latency affects ABR performance. In order to carry out this task, this thesis evaluates four prominent on-demand ABR algorithms, under a large number of target latency settings and network conditions, in low latency adaptive streaming when used in a regular DASH client and a CMAF client. In order to facilitate this extensive evaluation, a simulation model was developed which accurately imitates low latency live adaptive streaming using DASH and CMAF while allowing for faster than real-time evaluation of streaming sessions. The results of this evaluation will be used to determine whether CMAF chunks are beneficial to low latency live adaptive streaming, as well as, the impact of latency on the performance of prominent on-demand ABR algorithms. The simulation model was made available to the research community on GitHub [99], along with the throughput traces used to simulate realistic network conditions [128].

2. Investigation of video quality switches and their impact on the overall Quality of Experience in adaptive streaming. This thesis investigates the impact of video quality bitrate changes in adaptive streaming on the perceived Quality of Experience by users. In order to achieve this task, this thesis conducts a subjective study, using the standard methodology for video quality assessment, which evaluates 28 different video quality bitrate patterns using 14 pieces of content. The evaluated quality bitrate patterns depicted a variety of video quality impairments in order to determine their effect on the overall QoE. The key findings of this investigation were used to construct QoE-oriented design goals for a new ABR algorithm aiming to maximise the Quality of Experience.

3. Llama, a Low Latency Adaptive Media Algorithm. This thesis proposes a new ABR algorithm, designed to be employed in low latency conditions where the client video buffer is severely limited due to decreased latency. Llama was designed using the conclusions of the two investigations discussed above. First, the proposed ABR algorithm is designed to improve over on-demand ABR algorithms in low latency conditions, and second, the proposed ABR algorithm is designed using the key findings from the subjective testing evaluation. The design and implementation of the proposed ABR algorithm are detailed in this thesis.

Additionally, this thesis evaluates the proposed ABR algorithm against prominent on-demand, as well as, the recently published, state-of-the-art low latency ABR algorithms. This requires the evaluation of ABR algorithms under a large number of target latency settings and network conditions, in both DASH and CMAF implementations of low latency live adaptive streaming. To perform an extensive evaluation, compromising a large number of client settings and network profiles, all ABR algorithms have been implemented in the simulation model mentioned earlier, which allows for faster than real-time evaluation of streaming sessions. The achieved Quality of Experience by each ABR algorithm has been presented using a standardised methodology.

1.5 Thesis Structure

This thesis is presented in eight individual chapters. After the introduction, presented in this chapter, we discuss the background material in Chapter 2 which describes the evolution of multimedia streaming over the past 30 years.

Chapter 3 presents the related work, focusing on advancements in HTTP Adaptive Streaming, including low latency live adaptive streaming. Relevant work in the areas of Quality of Experience (QoE) and ABR algorithms is covered in this chapter.

In Chapter 4, we discuss the extensive subjective study carried out to further investigate the impact of video quality impairments on the perceived Quality of Experience. The chapter details the methodology behind the experiment as well as an in-depth analysis of the presented results, concluding with key findings.

Chapter 5 presents the proposed ABR algorithm, Llama, developed to operate in low latency live adaptive streaming. It describes the analysis of prominent on-demand ABR algorithms, based on which, in conjunction with key findings from the previous chapter, the design goals for a low latency ABR algorithm were constructed. The adaption logic of the proposed ABR algorithm is presented in detail.

In Chapter 6, we discuss the implementation of the simulation model used to evaluate the proposed ABR algorithm under a large number of settings and conditions. The model was verified against a real DASH player.

Chapter 7 presents an extensive evaluation of the proposed ABR algorithm, Llama. It is first evaluated against four prominent on-demand ABR algorithms, and later against three state-of-the-art low latency ABR algorithms, using the simulation model described in Chapter 6. The ABR algorithms have been tested using twelve latency settings, as well as, 7000 throughput traces in both modes of low latency live adaptive delivery, DASH and CMAF. Their performance is presented using individual QoE factors, as well as a standardised QoE model which combines all of the QoE factors into a single metric.

In Chapter 8, we conclude by discussing and summarising the contributions of this thesis. Additionally, potential future work is outlined.

Chapter 2

Background

In this chapter, we discuss the evolution of multimedia delivery over the Internet. The increasing demand for online multimedia streaming has led to rapid improvements in online multimedia delivery technology since its inception in the 1990s, facilitated by the improvements in media compression which we describe in the first part of this chapter. In the second part, we focus on the key technologies for multimedia delivery over the Internet developed over the past three decades.

Multimedia streaming refers to the method, in which media is being transmitted and consumed continuously, from a broadcasting source to a client, where the client is not required to store the entire multimedia element. The alternative approach to streaming is file downloading, where the entire multimedia file must be fetched by the client before it can be consumed. Multimedia streaming can be divided into two main modes of delivery: on-demand and live streaming. The former approach focuses on the delivery of already produced content while the latter focuses on the delivery of content generated in real-time, simulating a live television broadcast.

The Internet, being a best-effort network, presents significant challenges to the design and implementation of multimedia transmission. In traditional broadcast methods, such as terrestrial and satellite television, all clients receive the content in the same quality due to static bandwidth, leading to a uniform experience across users. This is difficult to achieve in multimedia delivery over the Internet, where the clients can vary significantly in terms of resources, including the computational power and bandwidth. Additionally, the client's bandwidth is not static and can increase or decrease during a multimedia stream, potentially leading to insufficient bandwidth for the transmission which will cause the playback to stall.

2.1 Multimedia Compression

Advancements in media compression played a key role in the evolution of multimedia delivery over the Internet. Media compression, also known as source coding, primarily focuses on the encoding of video and audio that results in fewer bits required to represent the digital media elements. Over the years, media compression has improved in terms of efficiency and quality, enabled by the increase in available computational power in consumer electronics. In 1988, the International Organization for Standardization (ISO) have created the Moving Pictures Experts Group (MPEG), tasked with the development of standards for coding and transmission of digital media. In this section, we briefly describe the main three standards created by the MPEG group, which, mainly focused on creating standards for decoding of media; allowing manufacturers to implement custom coding techniques.

2.1.1 MPEG-1

Created in 1993, MPEG-1 (ISO/IEC 11172) was the first standard developed by the group, primarily designed to allow the storage of video and audio on Compact Disc (CD) media which supported the maximum bitrate of 1.5 Mbps. The supported resolution and frame rate were the Source Input Format: 352x240 at 30 frames per second (NTSC) or 352x288 at 25 frames per second (PAL).

In this standard, the video is encoded by aggregating frames into groups of pictures (GOPs), and frames within each GOP are coded using the following frame types. Each GOP starts with an I-frame (Intra-Frame), also known as a keyframe, which can be decoded independently of other frames. I-frames are the largest as they contain complete information needed to decode the frame. Consecutive frames in the GOP are inter-frame coded into P-frames or B-frames. P-frame (Predicted-frame) contains less information than an I-frame as it only stores the difference between the current frame and the nearest previous I-frame or P-frame. B-frame (Bidirectional-frame) is the smallest in terms of size as it uses the nearest I-frames or P-frames in both directions as references. This layered approach to video encoding has been continued in future MPEG standards.

The MPEG-1 standard also specifies three layers of audio encoding with each consecutive layer increasing in complexity allowing for more efficiency at the same bitrate. Each layer supports channel encoding settings of mono, stereo, and dual (two independent mono channels), sampling rates of 32, 44.1, and 48 kHz, as well as the minimum bitrate of 32 kbit/s. The maximum bitrates supported by Layer I, Layer II, and Layer III were 448, 384, and 320 kbit/s respectively. MPEG-1 Layer II audio, better known as the MP3 format, is still widely used today in services such as music delivery over the Internet. Audio encoding employs psychoacoustics to compress audio, meaning, parts of the audio that cannot be heard by the human ear are discarded or reduced.

2.1.2 MPEG-2

In 1995, MPEG-2 (ISO/IEC 13818) was standardised with a much broader scope of application in mind, which allowed for the transmission of media over packet-switched networks, such as the Internet. It offers significant improvements over its predecessor, the MPEG-1 standard, in multiple areas. Video encoding now supported interlaced video as well as higher resolutions and bitrates.

MPEG-2 standard introduced video profiles and levels to support a broad scope of applications. An application can support certain profiles and levels, meaning, it can support only the relevant subset of the standard. Video profiles define supported features such as frame types and chroma format, while levels refer to resolutions, frame rates and maximum bitrates. The standard was later extended to support HDTV resolutions, an extension originally planned to be released as a new standard MPEG-3.

The standard also improves multiple aspects of audio encoding. MPEG-2 now supports multiple audio channels, up to 5.2 multichannel significantly improving over MPEG-1 standard's two-channel (stereo) limitation. It also introduced Advanced Audio Coding (AAC), a successor to the MP3 format. AAC supports more channels (up to 48) and more sample rates (up to 96 kHz), as well as higher coding efficiency when compared to its predecessor allowing for higher sound quality at the same bitrate. It employed a modified discrete cosine transform (MDCT) algorithm to achieve better compression efficiency.

2.1.3 MPEG-4

In 1998, the MPEG-4 (ISO/IEC 14496) standard was finalised, three years after its predecessor. It incorporates most of the features from the previous two standards while significantly expanding the scope of applications. It introduced new features including object-oriented media files, external Digital Rights Management (DRM), and Virtual Reality Modelling Language (VRML) support. The standard significantly improved coding efficiency over its predecessor and allowed for the encoding of mixed video, audio, and speech.

Regarding video encoding, the MPEG-4 standard introduced Advanced Video Coding (AVC), also known as the H.264 coder, to significantly improve compression efficiency as it allowed for the same video quality at half or less the bitrate achieved by the previous standards. AVC was designed to be transmitted over many different types of networks as the video coding layer is separated from the transport bitstream using the Network Abstraction Layer (NAL). AVC has been improved over the years and is currently the most common video encoding method used for video delivery over the Internet. One of the new features offered by AVC was Scalable Video Coding (SVC), where the video was encoded into multiple layers which could be transmitted independently over the network. The video could then be decoded by combining one or more layers, with more layers used resulting in higher video quality.

The MPEG-4 standard further improved audio encoding by introducing new formats and enhancing the Advanced Audio Coding (AAC) to allow for better sound quality at the same bitrate. MPEG-4 was designed to handle a larger variety of audio formats, from speech to music, and from lossy to lossless coding. The standard introduced Audio Object Types (AOT), with each audio format assigned a unique AOT, which define the tools and coding methods required to encode each audio format. The standard also introduced Audio Profiles, each supporting a subset of Audio Object Types, which define supported features as well as the tools and possible parameters required for audio encoding of the supported Audio Object Types.

2.2 Multimedia Delivery over the Internet

In this section, we describe the history of multimedia streaming over the Internet, focusing on the main technologies and approaches created in the past three decades. Multimedia delivery over the Internet has improved drastically over the years, not just in terms of the video and audio quality transmitted, but also in terms of efficiency and scalability. Initial media streaming relied on a stateful architecture where the server had to keep track of each client session. Today, media streaming is much more cost-effective as well as more efficient, mainly using a stateless architecture, allowing for broadcasts over the Internet to be watched by millions of viewers around the world concurrently. The developments in multimedia compression, described in the first section of this chapter, were an important factor in this advancement.

2.2.1 Real-Time Protocol Suite

In the 90s, numerous protocols in the Real-Time family have been standardised, including Real-Time Transfer Protocol (RTP[44, 113]), Real-Time Control Protocol (RTCP[113]), and Real-Time Streaming Protocol (RTSP[112]). All three protocols combined were used for low latency media streaming over the Internet.

Real-Time Transfer Protocol (RTP), standardised by the Internet Engineering Task

Force (IETF) in 1996, is a network protocol developed for real-time transmission of audio and video data over Internet Protocol (IP) networks. RTP can be used over the two main Transport Protocols, Transmission Control Protocol (TCP) and User Datagram Protocol (UDP), with the latter being the most common choice. It supports jitter compensation, out-of-order delivery, packet loss detection, and IP multicast. The protocol also supports multiple multimedia formats by defining profiles, with each profile assigned a payload format. RTP packet header includes the following key fields which allow for real-time streaming of media. The sequence number, which specifies each packet's order number allowing for detection of packet loss and the receiving of packets out-of-order. Timestamp, which contains the timing information, allows for synchronisation of media as well as jitter detection. CSRC identifiers, which contain the unique identifiers of contributing sources, allow for mixed media to be received from multiple sources.

Real-Time Control Protocol (RTCP), defined in the same standard, is used in combination with RTP. It provides each participant in a streaming session with Quality of Service (QoS) feedback which includes jitter statistics, packet loss, round-trip time (RTT) and packet delay variation. The QoS feedback is sent periodically by all participants and can be used by the application to adapt the stream parameters to changing network conditions.

Published in 1998 by the Internet Engineering Task Force (IETF), the Real-Time Streaming Protocol (RTSP) allows for real-time media transmission between a client and a server. The protocol is used to establish and control a streaming session. The actual media data is transmitted using another protocol, most of the time using RTP. The streaming session is controlled in a VCR-like manner, with the client capable of issuing the server commands, such as Play and Pause, to control the multimedia playback. The commands can also be issued by the media server. RTSP is similar to the Hypertext Transfer Protocol (HTTP), mainly in terms of syntax and operation, however with one key difference – it is a stateful protocol as opposed to the stateless HTTP protocol. RTSP uses the connection-oriented TCP as a transport protocol to maintain a reliable connection between the client and the server. The client can join a streaming session using an RTSP URL, after which it first issues a Describe command. The media server responds with a description of the available media streams. Then, the client issues a Setup command to start a streaming session – the server responds with a session identifier. The client begins the multimedia playback by issuing a Play command, after which the playback can be paused and resumed using Pause and Play commands. The client can issue additional commands, such as Set Parameter to modify the settings of a media stream. Finally, the client can end the streaming session by issuing a Teardown command.

IETF has published RTSP 2.0 in 2016 [111] which mainly addresses issues around NAT traversal of media streams. The protocol is still being used today – mostly in the first mile of video broadcasting, as well as in CCTV applications.

2.2.2 Multicast Broadcasts

Delivery of unicast video streams to a large number of viewers can be an inefficient use of bandwidth. For example, a stream delivered to 1000 clients will require the server to send the exact same video data 1000 times, resulting in high bandwidth consumption and leading to scalability issues due to potential bandwidth bottlenecks in the network. In 1986, IP multicast [29] has been standardised, an extension to the IP protocol which can be used to solve this issue. Multicasting allows the media server to send the data to the network once, after which the data will be propagated through the network to all of the subscribed clients. This can significantly improve bandwidth efficiency. However, multicasting introduces new issues when used for the delivery of multimedia over the Internet. The first issue is the lack of interactivity, the client cannot control the media playback as the same data is being transmitted to all clients. There is also no re-transmission mechanism, requiring the clients to employ new methods to deal with potential errors in data as the clients cannot request the lost or corrupted data to be sent again.

IP multicast utilises the Internet Group Management Protocol (IGMP [29, 38, 15]). In order to participate in a multimedia stream, the client will need to send a join message using the IGMP protocol to the network. Upon receiving the message, a multicast router will subscribe the client to the specified multicast group. Multicast groups are specified by IP addresses in the range between 224.0.0.0 and 239.255.255.255. Once subscribed, the client will begin to receive the multicast multimedia stream. The media server transmits the multimedia data to the network only, which is then propagated by the multicast routers to all of the subscribed clients. To quit a streaming session, the client needs to send a leave message using the IGMP protocol.

One of the most prominent multicast technologies was Mbone (Internet Multicast Backbone [34]), developed in 1992. It was free software that created a virtual network that ran on top of the Internet, capable of multicasting packets. At the time, most routers did not support IP multicast, hence Mbone employed a virtual network architecture to overcome the hardware limitations where multicast packets could be converted into traditional unicast packets through tunnelling. One of its key features is the mechanism to limit the scope of the transmitted data over multicast to prevent it from over saturating the Internet with multicast packets. This was implemented by setting a time to live (TTL) counter for each packet, which decreased by one each time the packet propagated through Mrouters. In 1994, the first live music concert was transmitted over the Internet to a large number of viewers using the Mbone technology.

2.2.3 Peer-To-Peer Streaming

Another technology utilised to help with scalability issues of traditional unicast multimedia streams was Peer-To-Peer streaming, with multiple schemes proposed for on-demand [45, 133, 46, 4] and live streaming [17, 70, 84, 150]. In Peer-To-Peer streaming, clients were receiving data from the media server or other clients participating in the streaming session. This means, that clients were now not only receiving data but also sending the data to other clients. This approach significantly reduces the bandwidth requirements of a media server, making it a much more cost-effective broadcasting method when compared to traditional client-server architectures. However, the Peer-To-Peer architecture introduces new challenges and limitations when employed for multimedia streaming over the Internet.

In Peer-To-Peer (P2P) networks all nodes (known as peers) are equal, creating a decentralised distribution of nodes, where each node contributes a share of their resources to the network. P2P networks became widespread in the early 2000s due to their applicability in file-sharing software. P2P architecture can be divided into unstructured and structured networks, as well as hybrid models. In unstructured networks, the node distribution is truly decentralised, and all peers are equal as there is no global management of the nodes. Nodes join the network by simply connecting to other nodes. This approach can cause scalability issues, as in large networks peers will need to search the network for required data by flooding the network with requests which will lead to a large portion of resources being spent by nodes on the management of search queries. Additionally, there is no guarantee for a search query to be resolved - leading to nodes being unable to fetch the required data. These issues can be solved by structured P2P networks, which impose a specific topology for all nodes. Each peer stores a list of its neighbour nodes to improve traffic routing efficiency. A distributed hash table (DHT) is also commonly used and stored by each peer. DHT stores the information regarding the ownership of each file using hashing, allowing each peer to quickly find a node serving the required file. Hybrid models further improve search efficiency by employing a centralised server that hosts information regarding other nodes and available files in the P2P network. Peers can fetch this information by contacting the server using a traditional client-server model, after which the required files can be fetched from other nodes using the P2P network.

One of the main issues in P2P streaming is the lack of Quality of Service (QoS) management. In unicast streams, the media server can provide reliable transmission of data to the client. Additional mechanisms can also be implemented to measure and manage the QoS between the client and the server. However, in P2P networks this is challenging as peers cannot provide reliable media streams due to significant differences in terms of available resources including the computational power and bandwidth. Any disruption in the media stream higher up the chain will negatively impact the peers down the chain. Another issue is the delay, since the media stream is shared in chains of peers, each subsequent peer in the chain will experience an increasing delay, making this approach unsuitable for low latency media streaming.

2.2.4 HTTP Adaptive Streaming

Prior to the adoption of HTTP Adaptive Streaming, HTTP Progressive Streaming was the middle ground between downloading the complete multimedia file before playback and real-time multimedia streaming. In progressive streaming, the client leverages the HTTP progressive download approach, where the client can request a specific byte-range of a multimedia file to avoid having to fetch the entire file before the playback can begin. Range requests feature was added to HTTP/1.1 by IETF in 2014 [39].

With the first prototype created in 2002 by the DVD Forum, HTTP Adaptive Streaming has significantly improved over the years, becoming the dominant technology for media streaming today, as well as, producing several standards and implementations along the way. The fundamental features of HTTP Adaptive Streaming are bitrate flexibility and high scalability.

Bitrate flexibility allows the media server to serve video and audio to a broad range of clients, supporting a wide spectrum of client characteristics such as computational power and available bandwidth. Multimedia content is firstly encoded into multiple bitrates, each offering a different quality while having different bandwidth requirements, and secondly partitioned into short-duration segments. The client can change the quality bitrate throughout the stream, at the segment level. The client adapts the quality bitrate mainly to counteract changes in the available bandwidth to avoid playback interruption, caused by the media buffer being fully depleted. An adaptive bitrate (ABR) algorithm is employed by the client to measure the network conditions and select appropriate quality bitrate in order to maximise the user's Quality of Experience (QoE).

High scalability is the second fundamental feature of HTTP Adaptive Streaming, achieved via the utilisation of standard HTTP server architecture which allows for compatibility with Content Delivery Networks (CDNs). Multimedia content can be encoded into multiple quality bitrates and stored on standard HTTP servers. Clients fetch the multimedia content by simply issuing HTTP requests for each segment at the desired quality bitrate. This client-driven approach eliminates the need for specialised media servers as the server does not need to maintain a session state for each individual client, further supporting compatibility with standard HTTP servers and CDNs. Content Delivery Networks play a key role in achieving high scalability, as they allow the media content to be delivered to users more efficiently by replicating the media stream from the origin server to thousands of edge servers located closer to the users. In the following sections, we describe the most influential HTTP Adaptive Streaming implementations.

2.2.4.1 Adobe HTTP Dynamic Streaming

Developed by Adobe, HTTP Dynamic Streaming (HDS[72]) supports the transmission of H.264 encoded videos over HTTP connections. The format supports both on-demand video as well as live streaming, and is compatible with Content Delivery Networks. Media content can be encoded into multiple bitrates, however, the H.264 codec must be used for encoding. HDS was supported only by the Adobe Flash Player, a browser plugin which enabled the viewing of multimedia elements and rich internet applications before the standardisation of HTML5. Since then, the plugin has lost adoption as most of its functionality was now included in HTML5, and hence supported natively by web browsers, with Adobe ending the support for the plugin at the end of 2020.

2.2.4.2 Microsoft Smooth Streaming

Smooth Streaming [148], developed by Microsoft, is another implementation of HTTP Adaptive Streaming, which was supported by a range of Microsoft services and devices, including Silverlight, Xbox as well as Windows Phone. This implementation utilises the Protected Interoperable File Format, standardised by Microsoft and based on the ISO/IEC base media file format. It differs from other implementations, as the encoded content is not segmented, but instead, HTTP progressive downloads are utilised which are more efficient. However, this approach requires specialised media servers which reduces its compatibility with Content Delivery Networks. Microsoft Smooth Streaming support both on-demand video as well as live streaming, including HD and 3D video.

2.2.4.3 Apple HTTP Live Streaming

Developed by Apple, HTTP Live Streaming (HLS[1]) is one of the two main implementations of HTTP Adaptive Streaming in use today. It was published by Apple as an IEEE RFC draft in 2017 (RFC 8216). The standard is widely supported, with native support present in most modern operating systems, and can be employed for both on-demand video as well as live streaming. However, HLS is the only HTTP Adaptive Bitrate implementation natively supported by Apple devices. It supports H.264 and H.265 formats for video encoding and the following formats for audio encoding: AAC, MP3, AC-3 and EC-3.

In HTTP Live Streaming, media content is encoded into multiple quality bitrates, after which it is segmented into fragments of equal duration. In 2016 the default fragment duration was reduced from 10s to 6s, however, it can be adjusted. The fragmented files are referenced in the index file, in the format of an extended M3U playlist, and stored together on a standard HTTP server. The client initiates a media stream by fetching the index file which indicates the available quality bitrate and where the file fragments can be found. Once the index file has been analysed by the client, it fetches the video, fragment by fragment, by issuing HTTP requests for consecutive fragments of desired quality bitrate. Quality bitrate can be changed by the client at fragment level, allowing for an implementation of an ABR algorithm to measure the network conditions and adapt the quality bitrate accordingly. Media encryption which allows for simple Digital Rights Management (DRM) and dynamic ad insertion are some of the key features of HTTP Live Streaming.

In 2019, Apple proposed an extension to the standard which focuses on enabling low latency in live streaming scenarios. It supports the concept of further segmenting fragments into sub-fragments, similarly to the CMAF Chunks which we describe later in this chapter. Sub-fragments can become available for download before the entire fragment is generated. This can approach can aid to reduce latency with little to no encoding penalty. Additionally, to eliminate network transfer overhead of sub-fragments, the extension introduces HTTP/2 Server Push support. In the earlier version of HTTP, the client would need to request a resource, after which it can be transmitted by the server to the client. HTTP/2 Server Push allows for data to be sent to the client by the server without the client requesting each resource individually. It is especially useful in environments with high round-trip time (RTT) and can be used with regular fragments as well as sub-fragments. The extension also tackles the problem of large playlist transfer overhead. When employing HLS for live streaming, the index file grows in size as the stream continues, with new fragments added to the index requiring the client to periodically fetch the index file. The extension introduces playlist delta updates, allowing the client to periodically fetch only the new parts of the index file to avoid having to fetch the entire file each time, significantly reducing the bandwidth used. Low-Latency HTTP Live Streaming is supported by Apple devices running the most up-to-date operating system; however, the proposed extension is not widely supported in other devices.

2.2.4.4 Dynamic Adaptive Streaming over HTTP

MPEG group has standardised an implementation of HTTP Adaptive Streaming in 2012 under the name of Dynamic Adaptive Streaming over HTTP (MPEG-DASH), the second main implementation in use today. It can be employed for both, on-demand video as well as live streaming. It is the only implementation of HTTP Adaptive Streaming that is codec agnostic, meaning, video and audio can be encoded using any encoding format. It is also the only implementation that is an international standard, with many companies from the streaming industry participating in the standardisation process. DASH is widely supported via Media Source Extensions in web browsers and continues to grow in adoption. The DASH Industry Forum (DASH-IF), consisting of companies from the streaming and media industries, oversees interoperability and adoption of the standard. The group created the open-source MPEG-DASH reference player, dash.js.

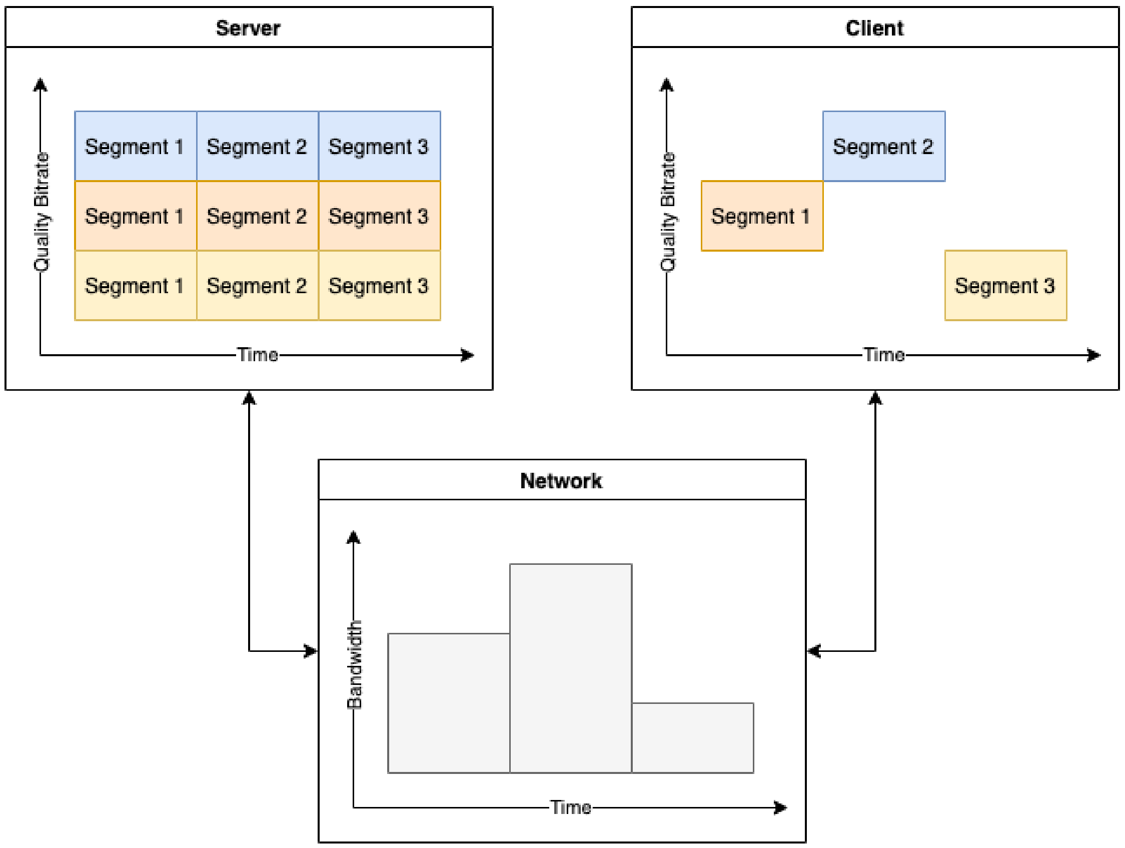

Figure 2.1: Overview of video streaming using DASH.

In DASH, a video is usually encoded into multiple bitrates, segmented and stored on a standard HTTP server along with a manifest file describing the content. The segments are of equal duration. The client first requests the manifest file, after which, it requests the video by issuing standard HTTP requests for each segment. The quality bitrate of each segment is selected by the ABR algorithm, usually deployed at the client side, which monitors network conditions and adjusts the quality bitrate accordingly to maximise the user's Quality of Experience. Quality bitrate can be changed at the segment level, that is, every segment duration. ABR algorithms usually measure network conditions by calculating the throughput of past segment downloads, the buffer level, or the buffer depletion rate. Figure 2.1 demonstrates the overview of video

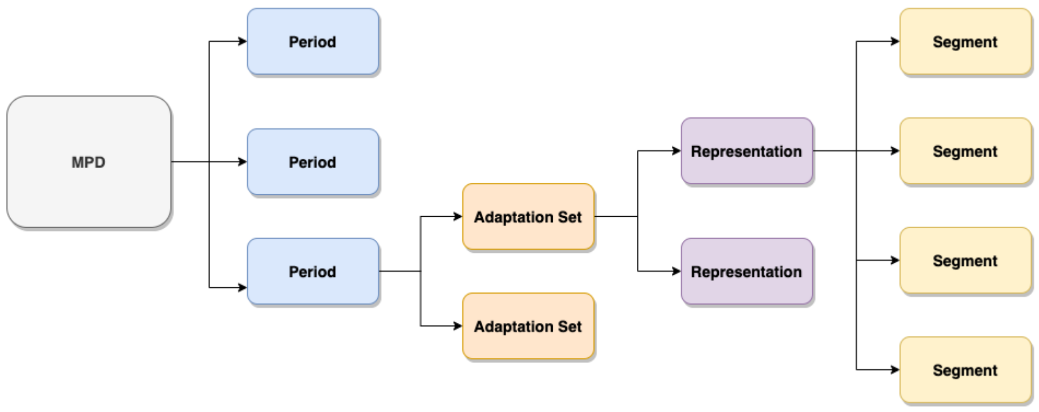

Figure 2.2: Structure of the MPD manifest file in DASH.

streaming using DASH, where the client requests the content segment by segment, adapting the quality bitrate of each segment to the perceived available bandwidth in the network.

Manifest files, specified as Media Presentation Descriptions (MPD), describe the stream parameters as well as the available multimedia content along with the available quality bitrates in an Extensible Markup Language (XML) format. Stream configuration includes parameters such as minimumBufferTime, which specifies the client target buffer level, and suggestedPresentationDelay, which specifies the target latency in the case of live streaming. Available video content is described using a hierarchy of Periods, Adaptation Sets, Representations and Segments, as seen in Figure 2.2. A video can be divided into periods, each containing a different scene or chapter. Each period is further divided into adaptation sets, with video and audio elements being presented in separate adaptation sets. Each adaptation set specifies the encoding format used and is further divided into representations, which include the content encoded using multiple parameters. For example, a video can be encoded into multiple representations, each offering a different resolution and bitrate. Finally, the representations are divided into segments of equal duration. The client can switch between representations at the segment level.

When employed for live streaming, the manifest file also specifies each segment's

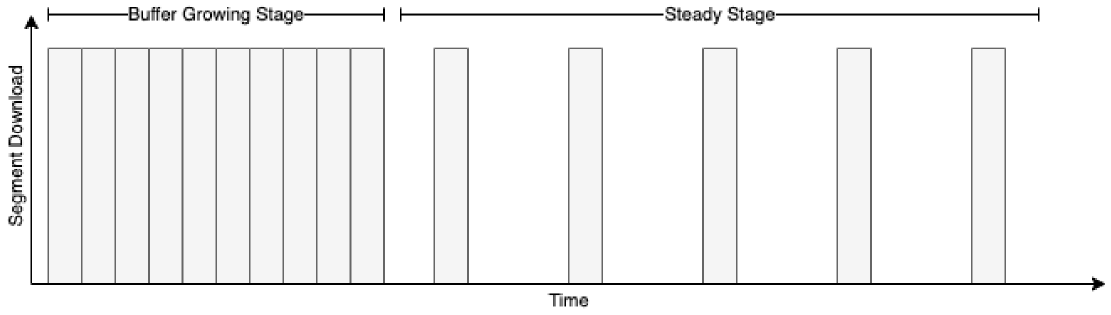

Figure 2.3: Client buffer stages in DASH.

availability time, allowing the client to request a new segment once it has been generated. The minimum latency in Live DASH is the duration of one segment. The client also needs a reasonable buffer of at least a few segments in order to be able to react to changes in network conditions before the player's buffer is fully depleted and the playback stalls. Achieving low latency in regular DASH would require short segments, resulting in significantly reduced encoding efficiency and increased number of HTTP requests needed, thereby increasing the transport overhead. Recently, the concept of further dividing segments into chunks has been introduced, using the Common Media Application Format (CMAF), which in combination with HTTP/1.1 Chunked Transfer Encoding can be utilised to achieve low latency in live DASH with little to no encoding as well as transport overhead. We describe this approach in more detail in the next section.

In on-demand delivery mode, as seen in Figure 2.3, at the beginning of a streaming session the client enters the buffer growing stage as it downloads the consecutive segments, back to back, until the configured minimum video buffer level is reached. Afterwards, the client enters the steady stage, where the client video buffer is depleted by one segment periodically, every segment duration, and therefore only one segment needs to be fetched in the background to fill the client video buffer back to the minimum level again. At this stage, the client does not utilise the available bandwidth fully if the segment download time is less than the segment duration, leaving the client idle

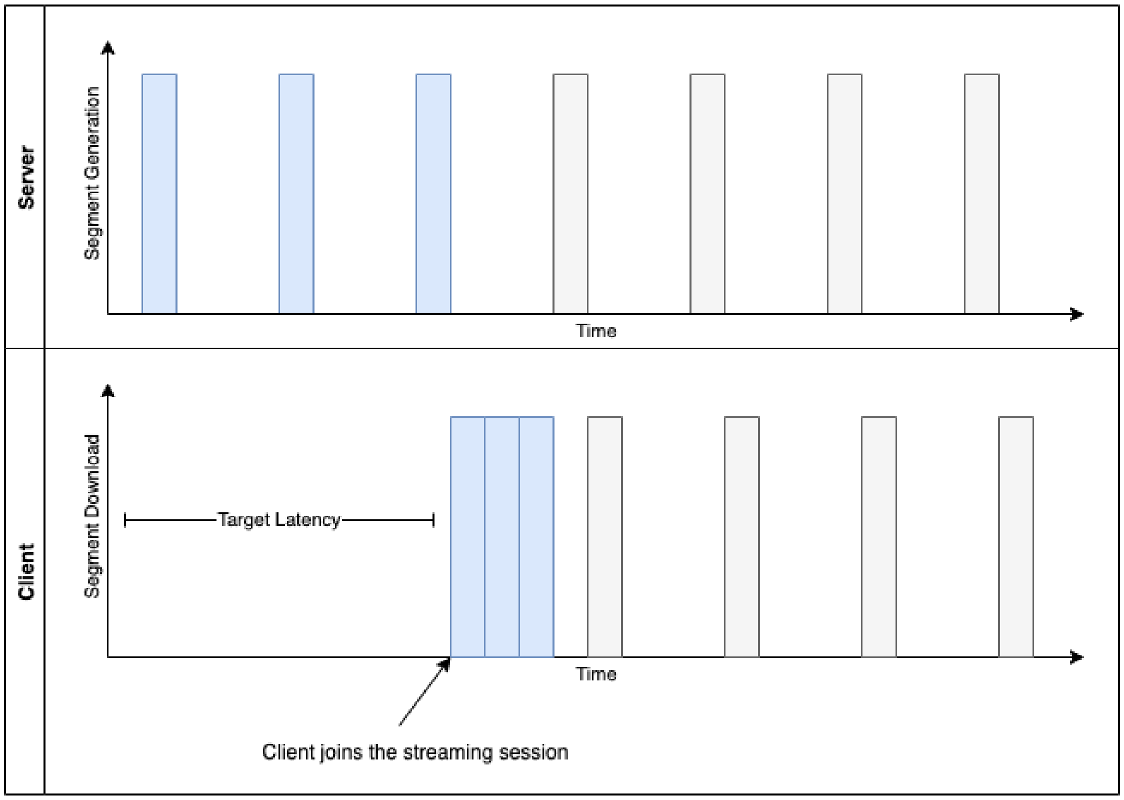

Figure 2.4: Live adaptive streaming using DASH.

between the download completion time and the current segment completing playback. In the case of multiple clients sharing the same bottleneck link, this leads to incorrect bandwidth estimation by ABR algorithms, resulting in clients either underestimating or overestimating the bandwidth and leading to unfair distribution of bandwidth between the clients. Additionally, incorrect bandwidth estimation can have a significant negative impact on the Quality of Experience.

Figure 2.4 demonstrates live streaming using DASH, where the content is generated in real-time. As the content is captured, segments are generated periodically every segment duration, becoming available on the server as specified in the manifest file. The client uses a Target Latency configuration parameter, which specifies how far behind the live edge the client should follow the stream. The client can join the streaming session when the duration equal to the Target Latency setting has elapsed since the beginning of the streaming session. Once the client joins the streaming session, it enters the buffer growing stage and begins to download the already available segments. The number of available segments at this point is dictated by the client's Target Latency setting, with higher settings resulting in a higher buffer level at the cost of higher latency. Once the initial segments have been fetched, the clients enter the steady stage, where it downloads segments as they become available periodically, every segment duration. For example, as seen in Figure 2.4, if the Target Latency setting is set to three segments, resulting in latency of three segment durations, the client will be able to download three segments back to back when it joins the streaming session, after which it will continue to download segments periodically, every segment duration.

2.2.4.5 Common Media Application Format

In 2018, the Common Media Application Format (CMAF[54]) has been standardised by the MPEG group. It is not another implementation of HTTP Adaptive Streaming, but rather an encoding and packaging format for segmented media. It introduces a common format for DASH and HLS, aiming to improve interoperability between the two implementations. DASH and HLS use different packaging formats, fragmented MP4 and MPEG-TS respectively. In order to serve content via both standards, the same video and audio data must be packaged into two different formats, doubling the required storage and packaging cost. CMAF offers a solution to this problem by introducing a common packaging format – ISOMFF fragmented MP4. The format, which introduces CMAF segments, is supported by both DASH and HLS, however, separate index files need to be created for both formats, an MPD file and an M3U playlist respectively. Another important feature offered by CMAF is the standardisation of Digital Rights Management (DRM) modes. It offers the following modes: AES-128 CTR, DRM used by DASH, and SAMPLE-AES CBC, used by HLS and later added to DASH.

Common Media Application Format introduces the concept of a CMAF chunk. Video can be divided into CMAF segments, which can now be further divided into smaller CMAF chunks. Chunks can be transmitted as soon as generated, as well as,

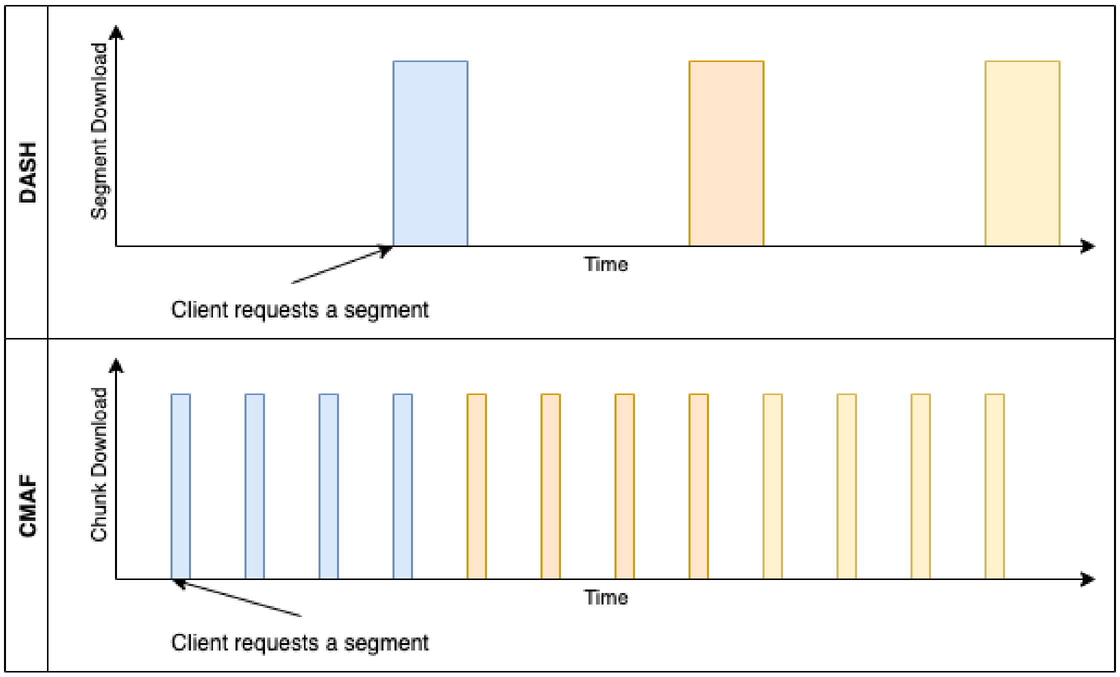

Figure 2.5: Comparison between live adaptive streaming using DASH & CMAF.

played out by the player as soon as received. This can reduce the minimum latency in live DASH from one segment duration to one chunk duration, allowing for low latency live streaming. Each segment contains one keyframe, in the first chunk, to reduce encoding overhead, meaning, the quality bitrate can only be changed at segment level just as in regular DASH. In order to minimise the transport overhead, CMAF chunks can be transmitted using HTTP/1.1 Chunked Transfer Encoding which allows for partial HTTP responses. The client requests a segment once its first chunk is ready by issuing a HTTP request, and the server responds with the first chunk, after which, it will transmit the remaining chunks once they are ready. A segment containing CMAF chunks can be requested as soon as the first chunk is created.

Figure 2.5 demonstrates the difference between live streaming utilising regular DASH segments and CMAF chunks combined with HTTP/1.1 Chunked Transfer Encoding. In this example, the client is close to the live edge, and hence fetches segments as soon as they become available. In the case of regular DASH segments, the client issues a HTTP request for each segment as soon as it becomes available, that is when the entire segment has been generated. When close to the live edge, this occurs periodically every segment duration. In the case of CMAF chunks combined with HTTP/1.1 Chunked Transfer Encoding, the client issues a HTTP request for each segment as soon as its first chunk becomes available, whereafter the remaining chunks are transmitted by the server as they become available - without additional HTTP requests. In this example, each segment consists of four chunks, meaning, the client can begin fetching each segment three chunk durations earlier when compared to regular DASH. HTTP/1.1 Chunked Transfer Encoding introduces new challenges, such as accurate estimation of the throughput of each individual chunk. Each segment requires a full HTTP request and response, with all chunk downloads belonging to the same HTTP response, hence the HTTP download time of the entire segment includes the idle periods between chunks being generated, leading to incorrect throughput estimation. HTTP/2 Server Push could also be used to reduce the network overhead of CMAF chunks, however, HTTP/1.1 Chunked Transfer Encoding is preferred due to better client and server support.

2.3 Summary

Multimedia streaming has become one of the largest services delivered over the Internet, driven by technological advancements throughout the past three decades. Media compression has evolved drastically, with the influence of the MPEG group, to support high resolutions, efficient encoding of video and audio, as well as, to enable the transmission of media over packet-switched networks, such as the Internet.

Delivery mechanisms have evolved in parallel, enabled by the progress in media compression, from stateful client-server architecture to scalable and stateless architectures which utilise Content Delivery Networks. Original delivery methods focused on specialised protocols, such as the Real-Time protocol suite. Today, the dominant delivery method of multimedia over the Internet is HTTP Adaptive Streaming (HAS);

a widely supported method thanks to the use of the HTTP protocol. In HAS, the content is encoded into multiple quality bitrates and fragmented into segments of equal duration. The client fetches the content, segment by segment, by issuing standard HTTP requests, and is capable of changing the quality bitrate at the segment level. An ABR algorithm is employed, usually at the client, to monitor the network conditions and adjust the quality bitrate in order to improve the user's Quality of Experience.

Chapter 3

Related Work

In this chapter, we discuss the relevant literature to the research work presented in this thesis. In the first section, we highlight the advancements in live adaptive streaming over the years, including the recent progress in low latency live adaptive streaming using DASH[119] and CMAF[54] – standards we have described in the previous chapter. In the second section, we describe the related work around Quality of Experience in adaptive streaming, an area which encompasses a large amount of relevant literature, including individual QoE factors, the dynamic relationships between them, as well as comprehensive QoE models which combine all of the individual factors in order to calculate an accurate prediction of the overall Quality of Experience of streaming sessions. In the third section, we discuss the related work in the space of ABR algorithms for adaptive streaming, describing the entire spectrum of adaptation solutions while describing in detail client-based ABR algorithms, the prominent type of adaptation solutions, including the state-of-the-art low latency ABR algorithms. In the final section, we summarise the entire chapter.

3.1 Low Latency Live Adaptive Streaming

In the early years of HTTP Adaptive Streaming, which we described in detail in the previous chapter, it has been utilised mainly for the delivery of on-demand content, that is, where the complete content is generated and encoded before a streaming session begins. While on-demand delivery is still the prominent application of HTTP Adaptive Streaming today, live streaming has become much more prevalent, where the content is generated in real-time. Most recently, low latency live streaming using HTTP Adaptive Streaming has become feasible, where the content is generated in real-time and delivered at low latency, that is, the time between the content being generated and then displayed at the client is severely minimised.

One of the earliest works in this area focused on the analysis of live streaming using the HTTP Adaptive Streaming standard. Lohmar et al. [80] defined four sources of delay that are specific to live HTTP Adaptive Streaming. The main one is the client video buffer, which we describe later. The remaining three sources of delay are as follows. Asynchronous fetching of media segments, referring to the time between a segment becoming available and the client issuing a HTTP GET request to fetch the segment. HTTP download time, which is the time required to fetch the requested segment. It is determined by the available bandwidth, which can change over time, between the client and the server. Segmentation delay, which introduces a delay of at least one segment duration.

In live HTTP Adaptive Streaming, the client video buffer is responsible for the storage of fetched video segments, queued for playback. The size of the client video buffer is determined by an ABR algorithm's ability to adapt to changing network conditions in a timely manner. It needs to be large enough to give the ABR algorithm enough time to measure changes in network conditions and adjust the quality bitrate before the client video buffer is fully depleted causing playback interruption. Hence, an insufficient client video buffer size will lead to frequent playback interruption, resulting in significant degradation to the overall Quality of Experience.

In order to reduce the client video buffer, and in turn the latency in live adaptive streaming, segments of shorter duration may be employed which can improve the performance of the ABR algorithm used. Additionally, shorter segments will reduce the segmentation delay. However, this will significantly increase the network overhead, as each segment needs to be requested by the client leading to an increase in the number of HTTP requests. Swaminathan et al. [122] demonstrated how HTTP/1.1 Chunked Transfer, which allows for partial HTTP responses, can be utilized to reduce latency in regular DASH without significant transport overhead.

Wei et al. [136] proposed the use of HTTP/2 Server Push, which allows the server to serve resources without the client having to explicitly request them, to reduce network overhead in low latency live adaptive streaming. Two push strategies were presented: All-Push and k-Push. In the former strategy, the client requests the first segment after which all of the remaining segments are pushed by the server without additional requests, while in the latter strategy the client issues a request every k segments. Both push strategies were shown to effectively reduce the network overhead in low latency live adaptive streaming. Xiao et al. [139] expanded the k-push strategy to create the adaptive-push scheme, which dynamically adjusts the value of k based on the observed bandwidth variation. Xu et al. [140] proposed another HTTP/2 Server Push delivery strategy, based on the k-Push, which is a QoE-driven dynamic push strategy where the value of k is optimised to maximise the Quality of Experience. Van Der Hooft et al. [131] showed how HTTP/2 Server Push can be used to improve the Quality of Experience of adaptive streaming in mobile networks, where the round-trip time is high. Huysegems et al. [55] demonstrated how HTTP/2 Server Push can reduce latency and improve video quality in live adaptive streaming in high round-trip time networks.

The approaches described above successfully reduce the additional network overhead caused by the use of short segments, however, they do not address the additional encoding overhead. Each segment needs to begin with a keyframe to allow for the interchangeability of quality bitrates at the segment level. Reducing the segment duration will require more keyframes to be employed, resulting in significant additional encoding overhead. Bouzakaria et al. [13] proposed the use of HTTP/1.1 Chunked Transfer Encoding combined with the ISO Base Media File Format (ISOBMFF) container to achieve low latency live adaptive streaming with minimal network and encoding overhead. ISOBMFF container was later utilised in the Common Media Application Format (CMAF) as it allows for the further division of segments into chunks with minimal encoding overhead. CMAF can reduce the segmentation delay in HTTP Adaptive Streaming to one chunk duration as segments can be divided into chunks [54].

Essaili et al. [35] demonstrated that the use of CMAF chunks combined with HTTP/1.1 Chunked Transfer can significantly reduce the latency, as well as, reduce the initial playback delay at the cost of increased frequency of rebuffering events, with more chunks per segment resulting in a bigger trade-off between the two Quality of Experience factors. This suggests that CMAF can be successfully employed to reduce the latency, however, it requires appropriate adaptation logic to make sure that the Quality of Experience is not degraded. Viola et al. [132] demonstrated the reduction in latency due to CMAF chunks combined with HTTP/1.1 Chunked Transfer, however, they also highlighted the issue of degradation in Quality of Experience as the number of chunks per segment tested increased, manifested by increasingly frequent rebuffering events.

While the combination of CMAF chunks and HTTP/1.1 Chunked Transfer Encoding can significantly reduce the latency in live adaptive streaming, it also introduces the problem of inaccurate bandwidth estimation, which is partially responsible for the degradation in Quality of Experience described earlier. As chunk delivery at the live edge is restricted by the encoder, estimation of network throughput is difficult for applications that have no direct visibility of the idle periods between chunks, meaning, the client will interpret each HTTP response as one continuous response where the response download time includes the idle periods between chunks being generated at the encoder. Bentaleb et al. [9] attempted to solve this problem by ignoring throughput measurements for chunks that contain idle time in their download times. Yadav et al. [142] proposed a solution to this problem with involves the combination of the existing chunk parser and filtering of downloaded chunk data. Ozcelik et al. [101] proposed a new bandwidth measurement heuristic that can measure the available bandwidth more accurately, in low latency live adaptive streaming with CMAF chunks and HTTP/1.1 Chunked Transfer Encoding.

3.2 Quality of Experience

Quality of Experience (QoE) is an important aspect of all user-oriented services, including HTTP Adaptive Streaming which offers great resilience to, even severe, variations in network conditions. In HTTP Adaptive Streaming, the video quality can be sacrificed in order to ensure uninterrupted playback of content at the client. Video quality can be changed every segment duration resulting in high flexibility and leading to many opportunities at which the client can alter the video quality to try and maximise the user's Quality of Experience.

However, the choice of appropriate video quality is not always straightforward, requiring the client to try and balance different QoE factors, such as the average video quality and the number of playback interruptions. For example, the client could request all of the segments at the lowest video quality, ensuring no interruptions to playback even if the network conditions deteriorate to a certain degree, however, this approach might result in low overall Quality of Experience, as the lowest video quality might be poorly perceived by the user. On the other hand, in another extreme scenario, where the client always picks the highest video quality, which can result in major interruptions to playback if network conditions deteriorate, might achieve low Quality of Experience as well. This is because even if the highest video quality is perceived well by the user, the major playback interruptions might be perceived poorly and even negate the positive QoE impact of the highest video quality.

In this section, we describe the research in the area of Quality of Experience in HTTP Adaptive Streaming. In the first part, we focus on the individual QoE factors which contribute to the overall Quality of Experience, and on the subjective testing work carried out to investigate how the individual QoE factors contribute to the overall Quality of Experience. In the second part, we describe QoE models designed for HTTP Adaptive Streaming, which combine the various QoE factors to produce a single QoE metric, aiming to predict the overall Quality of Experience of streaming sessions. QoE models are a useful tool for quantitative comparison of ABR algorithm performance in adaptive streaming, including low latency live delivery of content.

3.2.1 QoE Factors

Barman et al.[7] published the most recent survey on Quality of Experience in HTTP Adaptive Streaming. Based on the analysis presented in the paper, the Quality of Experience can be divided into the following individual QoE factors.

Start-up delay. It refers to the time between the user requesting the start of video playback, and the client beginning the video playback. This time depends on how many segments the client needs to fetch before the playback can begin, as well as, on the segment duration and download time of the segments.

Rebuffering in terms of duration, frequency and temporal location. Rebuffering occurs when the client video buffer is completely depleted, causing the playback to be interrupted as there are no more available segments to be played out by the player. This usually occurs when the download time of segments exceeds the segment duration, caused by deteriorations in the network bandwidth. Rebuffering can be measured in three ways. Rebuffering duration refers to the duration of rebuffering events, the time it takes for the client video buffer to receive another segment after the playback has been interrupted. Rebuffering frequency refers to the number of rebuffering events within a streaming session. The temporal location of rebuffering refers to the time at which a rebuffering event occurs, relative to the streaming session's run time.

Quality switching in terms of frequency and magnitude. In adaptive streaming, the quality of the video as well as audio can be changed every segment duration, by any number of supported encoding qualities. Frequency of quality switching refers to how many times the quality of video or audio has changed throughout a streaming session, while the magnitude measures by how many encoding qualities the video or audio has changed.

Quality down-switching and up-switching. Down-switching refers to the video or audio changing to a lower quality, while up-switching refers to changing to a higher quality.

Primacy and recency effects. Based on research in psychology, rebuffering or quality switching can have a different impact on how the user perceives it, depending on when the experience occurred, with primacy referring to the beginning of the streaming session and recency referring to the end of the streaming session.

The survey concluded that rebuffering events are the most annoying to users, with both the duration (as seen in [90], [143], [107], [5], [3], [31]) and frequency (as seen in [149], [134], [32], [5], [3], [31]) of rebuffering events being relevant, and hence concluded that an ABR algorithm should aim to minimize both. They also noted that some studies suggest that very short rebuffering events are not noticeable, and therefore, are less annoying to users[129].

Furthermore, the survey also mentions quality switching to be another important QoE factor, where a high number of quality switches can have a negative impact on the overall QoE, as seen in studies [130], [41] and [121]. They also noted that multi-level quality switches, where the quality changes across more than one quality level, can be detrimental to overall QoE [124, 121], suggesting that the magnitude of quality switching is also important. However, the impact of changes in video quality has been investigated in less detail than rebuffering, as most studies primarily focus on various forms of rebuffering. Additionally, since the quality can be changed every segment duration, the number of possible quality patterns grows exponentially with the number of supported encoding qualities, making it challenging to evaluate all forms of quality switching.

Ghadiyaram et al. [43] found that rebuffering events at the beginning of the stream were less annoying to users than at the end, and attributed it to the hysteresis effect first observed by Seshadrinathan et al. [114].

3.2.2 QoE Models

Measuring the Quality of Experience of multimedia sessions, including those delivered using HTTP Adaptive Streaming, is challenging as it requires the assessment of multiple QoE factors in order to produce a single metric, where the relationships between individual QoE factors are dynamic, and hence difficult to quantify. Quality of

Experience models can be classified into the following categories, as stated in [123], [145], and [67]: media-layer models, parametric packet-layer models, parametric planning models, bitstream-layer models, and hybrid models.

Media-layer models predict Quality of Experience by analysing audio and video signals. These models, also known as pixel-based models, can be deployed to evaluate unknown systems as they do not need prior knowledge about the system, such as the codec used for the encoding of video and audio. Media-layer models can be further divided into full-reference, reduced-reference and no-reference models. Full-reference models require full source information and produce the most accurate results out of all three types of media-layer models. Common examples of such models are Peak Signal to Noise Ratio (PSNR) and Structural Similarity Index Measure (SSIM) [50], as well as the Video Multimethod Assessment Fusion (VMAF[75]). The two proposed models in [134] utilise PSNR to create a model specifically designed for adaptive streaming. Reduced-reference models, such as [16], require limited source information. No-reference models do not require source information to predict quality. Examples of such models include BLIINDS[110] and NIQUE[93].

Parametric packet-layer models utilise packet header information to predict the Quality of Experience. These models do not analyse the video or audio included in the packet payload, resulting in solutions with low computational requirements. The packet header can contain useful metrics such as packet loss or bitrate, based on which the Quality of Experience can be estimated. Prominent examples of such models include the ITU-T Rec. P.564[60] and ITU-T Rec. P.NAMS[59]. The model presented in [90] and [92] proposed Quality of Experience estimation for adaptive streaming, based on several QoE factors: start-up delay, rebuffering frequency and rebuffering duration. The model presented in [109] focused on rebuffering as well, however, it did consider the temporal location of rebuffering. The model was further extended in [149] to include more QoE factors: start-up delay and video quality switching. Video quality alone was utilised in the model proposed in [51], which predicts Quality of Experience based on two QoE factors: quality switching and their amplitude, as well as the time spent at the highest video quality. The model proposed in [76] utilised several QoE factors, represented as functions: video quality switching, rebuffering frequency, frame rate, and rebuffering duration.

Parametric planning models utilise parameters such as the bandwidth and packet loss in the network or terminal to estimate the Quality of Experience. These models cannot be employed to evaluate unknown systems. Examples of such models are ITU-T Rec. G.1070 [58] and ITU-T Rec. G.107 [10].

Bitstream-layer models do not use pixel information, but instead utilise information associated with features of the encoded media, derived from the bitstream and packet layer information. Such features include bitrate, frame rate, motion vector, PLR, and Quantization Parameter. These models are of low computational complexity, making them a suitable choice for deployment to measure the Quality of Experience in real-time. However, most of these models only work with specific codecs. A prominent example of a bitstream-layer model is ITU-T Rec. P.1202 [61]. The model proposed in [117] predicts Quality of Experience where H.264 encoder is utilised, by encompassing several QoE factors: Quantization Parameter, average and maximum rebuffering duration, as well as rebuffering frequency. In [141] the proposed model predicts Quality of Experience based on instantaneous and cumulative QoE factors including video quality, quality switches, and rebuffering. The introduced model in [47] estimate the Quality of Experience based on the median and minimum video quality. In [130] and [129], the proposed model focuses on video quality and the switches in video quality in order to estimate the Quality of Experience. The model introduced in [107] encompasses more QoE factors to estimate Quality of Experience, including the video quality and its changes, recency effect, and rebuffering.

Hybrid models are combinations of any of the models above, resulting in typically more complex models with higher accuracy of Quality of Experience prediction. The following hybrid models have been developed over the years. The proposed model in [28] estimates Quality of Experience by utilising the average video quality, frequency of video quality changes, and standard deviation of video qualities, along with tunable parameters. In [19] the introduced model estimates the Time Varying Subjective Quality, using a Hammerstein-Wiener model with defined functions for input and output. The model demonstrated in [116] estimates Quality of Experience by considering the video quality of each segment, video quality changes, as well as, primacy and recency effects. The model shown in [32] estimates the Streaming Quality Index by utilising several QoE factors including rebuffering, start-up delay, and the video quality, calculated as a moving average. The proposed models in [5] and [6] employ a machine-learning approach that combines objective quality metrics with rebuffering and recency. The model introduced in [115] utilises a recurrent neural network to predict Quality of Experience. In [36] the demonstrated model estimates Quality of Experience using a wide range of QoE factors including objective quality metrics, rebuffering frequency as well as duration, and recency. The proposed model in [33] employs Expectation Confirmation Theory in order to predict Quality of Experience.

Recently standardised, ITU-T Rec P.1203 [62] is the most prominent hybrid model designed to estimate Quality of Experience in adaptive streaming. The model consists of the following three main modules. Pv, a video quality estimation module that calculates scores per second based on the visual element. Pa, an audio quality estimation module that calculates scores per second for the audio element. And finally, Pq, an integration module that combines the scores provided by the previous modules with additional factors, such as rebuffering. The video module encompasses four modes of operation, each consecutive mode requiring more information while providing a Quality of Experience prediction of higher accuracy. Mode 0 requires the following information: display resolution, frame rate, bitrate of each segment. This information can be supplied using the manifest file, as well as the general metrics of a streaming session. Mode 1 requires the same information as the previous mode along with the actual video and audio segments utilised in the streaming session. The segments are inspected using header information. Mode 2 requires the same information as the two previous modes along with 2% of media stream information and utilises deep packet inspection as well as partial bitstream parsing. Mode 3 requires the same information as Mode 1, along with entire media stream information, collected using bitstream parsing. The model estimates the Quality of Experience in the form of a mean opinion score on a 5-point scale designed for subjective testing using the absolute category rating (ACR) method, standardised in ITU-T Recommendation P.910 [57].

3.2.3 Summary

In this section, we have described the related work in the area of Quality of Experience in HTTP Adaptive Streaming. We have highlighted the contributing QoE factors, including the rebuffering and quality switching factors, and the difficulty of quantifying the relationship between the individual QoE factors and how they contribute to the overall Quality of Experience. Additionally, we have described relevant QoE models which aim to tackle this challenge to provide an accurate estimation of Quality of Experience in HTTP Adaptive Streaming. The prominent QoE hybrid model, ITU-T Rec P.1203, has been introduced in the years which offers a comprehensive evaluation of overall Quality of Experience in HTTP Adaptive Streaming. We employ this model in the evaluation presented in Chapter 7 of this thesis.

3.3 ABR Algorithms

In the previous section, we have examined the individual QoE factors which contribute to the overall Quality of Experience, as well as the challenging relationships between the various factors. In HTTP Adaptive Streaming an ABR algorithm, typically deployed at the client, is tasked with adapting the quality bitrate to changes in network conditions. Every segment duration, an ABR algorithm measures the ever-changing network conditions and selects the appropriate quality bitrate for the next segment to be fetched. The ABR algorithm aims to achieve the best overall Quality of Experience possible given the network conditions observed.

Bentaleb et al. [8] published the most recent survey of ABR algorithms designed for HTTP Adaptive Streaming. They outlined the main objective of an ABR algorithm as the maximisation of the user's Quality of Experience. This involves two complex tasks, the estimation of network conditions and appropriate bitrate selection. The ABR algorithm must be able to monitor the network conditions accurately, which can be done in a variety of ways such as throughput estimation or client video buffer level observation. Based on the network conditions observed, the ABR algorithm must select the appropriate quality bitrate, that is, quality bitrate which will maximise the overall Quality of Experience of the streaming session. This involves balancing the different individual QoE factors, which we described in the previous section of this chapter, including rebuffering in terms of frequency and duration, as well as, quality switching in terms of average quality bitrate, frequency and magnitude. In this section, we will examine ABR algorithms using a classification based on the taxonomy presented in the survey which groups them into client-based, server-based, network-assisted and hybrid adaptation solutions.

3.3.1 Client-based adaptation

In client-based adaptations the ABR algorithm is fully implemented at the client, that is, the client itself measures the network conditions and selects the appropriate quality bitrate alone. The server does not interfere with the client's decisions, and there is no cooperation between the client and the server. This is the most common approach, as it allows for scalability and cost-effectiveness due to the server not having to keep track of each client and their state, leading to standard HTTP servers being sufficient to serve the content – increasing compatibility with Content Delivery Networks. Client-based ABR algorithms are the most common type of adaptation and are an active area of research that has produced a broad scope of solutions over the years. Based on the taxonomy presented in the survey, client-based adaptation solutions can be divided into the following six categories: bandwidth-based, buffer-based, proprietary, mixed, machine-learning, and low latency ABR algorithms. In the following subsections, we discuss each category as well as highlight the prominent ABR algorithms.

3.3.1.1 Bandwidth-based ABR algorithms

Bandwidth-based ABR algorithms select video quality based on estimates of the available bandwidth, usually by calculating the throughput of the previously fetched segments. Adaptation solution presented in [78] utilises smoothed throughput estimation, along with step-wise increase and aggressive decrease approach, to select an appropriate quality bitrate. This approach has been further developed in [79], resulting in two ABR algorithms that support serial and parallel fetching of segments.

In [106] the proposed ABR algorithm employs throughput prediction in order to select appropriate quality bitrate. Throughput prediction is performed using the throughput of the most recently fetched segment, previous throughput estimate, as well as weighting factors which adjust the impact of the difference between the two measurements. This approach was further improved in LOLYPOP [86], an ABR algorithm designed for live streaming in mobile networks. The proposed ABR algorithm employs throughput predictions on multiple time scales using the Transmission Control Protocol, as well as relative prediction error distributions. Additionally, the ABR algorithm aims to maximise the Quality of Experience by heuristically maximising the average quality bitrate while minimising skipped segments and quality bitrate switches.

ABR algorithm proposed in [73], Panda, utilises smoothed bandwidth estimate along with a probe-and-adapt approach similar to TCP's congestion control[64]. It determines a target average data rate, based on which the appropriate video quality is selected. It monitors throughput and adjusts the target average data rate accordingly, as well as calculates the inter-request time for each segment to allow the buffer level to move towards the configured minimum buffer level. The introduced ABR algorithm was designed to overcome the fundamental problem of poor bandwidth utilisation by clients in HTTP Adaptive Streaming, in order to improve the Quality of Experience in scenarios where multiple clients share the same bottleneck link.

HTTP Adaptive Streaming suffers from poor bandwidth utilisation by clients. At the beginning of a streaming session, the client fetches the consecutive segments one after another without delay, until the configured minimum client video buffer is reached – this is known as the buffer growing stage of the streaming session. Subsequently, the client enters the steady state, where the client video buffer is depleted by one segment periodically, every segment duration, and therefore only one segment needs to be fetched, in the meantime, to fill the client video buffer to the minimum level again. At this stage, the client does not utilise the available bandwidth fully if the segment download time is less than the segment duration, leaving the client idle between the download completion time and the current segment completing playback. In the case of multiple clients sharing the same bottleneck link, this leads to incorrect bandwidth estimation by ABR algorithms, resulting in clients either underestimating or overestimating the bandwidth which can result in degraded Quality of Experience.

3.3.1.2 Buffer-based ABR algorithms

Buffer-based ABR algorithms select video quality based on the client video buffer level alone, usually only selecting a video quality once a specific buffer level threshold has been reached or utilising the buffer level in an optimisation problem. This approach is especially effective in conditions where throughput estimation is inaccurate or unavailable, requiring an alternative indicator of network conditions. Throughput estimation might be inaccurate in scenarios where multiple clients share the same bottleneck link due to the poor bandwidth utilisation by clients, as described in the previous section.

The introduced adaption solution in [95] aims to detect oscillations and compensate in such scenarios as the clients end up over-subscribing or under-subscribing the shared bottleneck link once they reach the steady state. The proposed ABR algorithm utilises a buffer model which is based on a mathematical function that limits quality bitrate levels according to the buffer level. To detect oscillations, the ABR algorithm uses an oscillation factor based on the quality bitrate switching variance. Once oscillations have been detected, the ABR algorithm employs a Compensation Algorithm to smooth it.

The proposed ABR algorithm in [53], BBA-0, selects the quality bitrate based on the current video buffer level. BBA-0 operates using a rate map, which calculates the required buffer level threshold for each quality bitrate. Additionally, the ABR algorithm defines the safety area as a reservoir, which dictates the minimum buffer level threshold needed to absorb network variation caused by segment download. The authors have found the buffer-based approach to be effective when the client is in the steady state, however, they also note that the bandwidth-based approach is more effective when the client is in the buffer growing stage.

The proposed adaptation solution in [120], Bola, utilises Lyapunov optimisation methods to minimize rebuffering and maximize video quality. The authors have formulated bitrate adaption as a utility maximisation problem which includes two QoE metrics: average quality bitrate and rebuffering, where the former increases the utility, and the latter decreases it. Bola is an online bitrate adaptation algorithm that incorporates Lyapunov optimisation, using the buffer level alone, to maximise the defined utility function. Bola is one of the default ABR algorithms implemented in the DASH reference player [25].

3.3.1.3 Proprietary ABR algorithms

Proprietary adaptations include ABR algorithms from proprietary adaptive streaming standards and implementations. The adaptation logic included in the proprietary standards is not transparent as it is a critical part of the commercial products built upon these standards. These include the adaptation solutions found in Adobe HTTP Dynamic Streaming (HDS) [72], Microsoft Smooth Streaming (MSS) [148], and Apple HTTP Live Streaming (HLS)[1]. In the previous chapter, we have discussed these proprietary streaming standards in more detail. Several studies have evaluated the performance of these proprietary ABR algorithms [22, 137, 96] and found the ABR algorithms to struggle in scenarios where multiple clients share the same bottleneck link.

3.3.1.4 Mixed ABR algorithms

Mixed ABR algorithms include those that take into account multiple metrics such as bandwidth estimation, buffer level, and segment duration when selecting video quality. The adaption solution proposed in [74] utilises an online algorithm combined with dynamic programming to solve an optimisation problem using both, the bandwidth estimate and buffer level. The optimisation problem is defined as an alpha-fairness utility function of any general QoE metric. The proposed solution was combined with

Panda [73], a bandwidth-based ABR algorithm we have described before, to improve its performance.

The proposed ABR algorithm in [118] utilises Fuzzy logic which takes into account both bandwidth estimate as well as the buffer level. It was designed to improve QoE fairness in scenarios where multiple clients share the same bottleneck link. The ABR algorithm leverages the buffer level to progressively download segments of higher quality bitrate than the estimated bandwidth to fully utilise the available bandwidth. It uses a Grey-model predictor to estimate future buffer level, allowing the ABR algorithm to react sooner to changes in network conditions than traditional ABR algorithms to effectively reduce rebuffering.

The introduced ABR algorithm in [27], ELASTIC, uses the feedback control theory and aims to fully utilise the available bandwidth in order to improve QoE fairness in scenarios where multiple clients share the same bottleneck link. Similarly to the previous ABR algorithm, ELASTIC fully utilises the link by selecting higher quality bitrate than the estimated bandwidth once the client reaches the steady state. The ABR algorithm employs the feedback linearization technique to select an appropriate quality bitrate in order to drive the buffer level towards a set point.

In [65] the proposed ABR algorithm, Festive, employs a random scheduler using the buffer level and estimates bandwidth using a harmonic mean. The harmonic mean is robust to outliers, which improves bandwidth estimation, while the random scheduler requests segments independently of the player's start time, which improves the fairness between multiple clients sharing the same bottleneck link. The proposed ABR algorithm consists of three components: chunk scheduling, bandwidth estimation and bitrate selection. Chunk scheduling utilises a random scheduler, which adds a small and random delay to segment requests. This approach results in the segments being downloaded periodically, but with a small difference in request timings. In scenarios where multiple clients share the same bottleneck link, this approach is beneficial as the clients do not start to fetch segments at the exact same time. Bandwidth estimation is performed using a harmonic mean, resulting in a smoothed estimate. Bitrate selection component compromises of two parts, stateful bitrate selection and delayed update, and is designed to ensure fair bandwidth allocation in scenarios where multiple clients share the same link, as well as, to improve quality stability.

The proposed ABR algorithm in [146], RobustMPC, utilises a model predictive control algorithm to solve an optimization problem for a number of segments ahead. It requires both throughput estimation and buffer level, optimizing towards a defined Quality of Experience model. The authors have proposed a sample QoE model which can be re-configured as well as improved in the future. It combines rebuffering, average quality bitrate, and quality switching QoE factors to produce a single metric. The ABR algorithm selects an appropriate quality bitrate for each segment by solving a QoE maximisation problem using a moving horizon of segments. It performs the following three tasks. First, the algorithm predicts the throughput for a number of segments ahead. Next, the ABR algorithm solves an optimisation problem, using the predicted throughput, the buffer level, and previous bitrate as input, to find the optimal quality bitrate for the next segment. Finally, the ABR algorithm applies the optimal quality bitrate. Additionally, the authors have developed FastMPC, a version of the algorithm that requires significantly less computational power at the expense of minimal performance penalty.

3.3.1.5 Machine-learning ABR algorithms

Machine learning adaptions utilise algorithms from the machine-learning research area, including reinforcement-learning and deep-reinforcement-learning. ABR algorithm proposed in [102] utilises reinforcement-learning in order to improve the Quality of Experience, especially in scenarios where multiple clients share the same bottleneck link. The proposed adaptation solution is a Q-Learning-based algorithm that is capable of learning and dynamically adapting its policy according to the network conditions present, aiming to maximise the Quality of Experience. It employs a Homo Egualis-like reward term in the reward function, allowing for clients to reduce their Quality of Experience in order to improve the QoE of under-performing clients, resulting in greater QoE fairness in multi-client scenarios. The learning process utilises both bandwidth estimation and buffer level.

Adaptation solution proposed in [85], Pensieve, employs a reinforcement learning approach to generate ABR algorithms. It does not use pre-programmed control rules or explicit assumptions, making it a flexible model which can be combined with a variety of Quality of Experience models and used in any environment. Pensieve utilises a neural network model which can be trained based on past streaming sessions to select the appropriate quality bitrate. The neural network maps past observations, consisting of throughput, buffer, and segment sizes, to quality bitrate decisions of future segments. It utilises the actor-critic A3C [89] as its reinforcement-learning algorithm with the training process involving evaluation of sessions using simulations runs across a large number of bandwidth profiles.

ABR algorithm presented in [40], D-DASH, employs a combination of deep learning and reinforcement learning to optimise the Quality of Experience in DASH streaming. It consists of deep neural networks and a reinforcement learning mechanism. D-DASH utilises a reward function that takes into consideration the main QoE factors: quality bitrate variations and rebuffering events. Its learning architecture compromises of two twin neural networks along with a replay memory to improve the stability and efficiency of the proposed learning algorithm. The computational cost of the proposed solution can be greatly reduced by performing a pre-training phase, allowing for its deployment in real DASH clients.

3.3.1.6 Low latency ABR algorithms

Low latency ABR algorithms have been designed specifically for low latency live adaptive streaming, where the focus on achieving as low latency as possible introduces new challenges. To achieve the low latency, the client buffer must be severely limited as the multimedia content is generated in real-time, meaning, only already generated segments can be fetched which become available periodically every segment duration, resulting in the client's potential maximum buffer level to be limited by the target latency. For example, when the client is configured to the target latency of 4s, and the segment duration is 1s, the potential maximum buffer level will equal to four segments or 4s. Therefore, low latency configuration reduces the ABR algorithm's time to react to changes in network conditions before the client buffer is depleted, leading to interruption in playback. In 2020, the following three low latency ABR algorithms have been published, the first adaptation solutions of this kind.

L2A[68] is based on Online Convex Optimization and aims to minimise the latency without negatively impacting the Quality of Experience. It does not require parameter tuning and can be deployed in any environment. The ABR algorithm does not require throughput estimation, making it a suitable adaptation solution for environments with a combination of CMAF chunks and HTTP/1.1 Chunked Transfer Encoding, where bandwidth estimation is difficult. It utilises Online Convex Optimization, an online optimisation method, to create an online learning framework that has low computational requirements. In this framework, the client learns to make decisions that aim to minimise the adversarial loss function. L2A defines the adversarial loss function as latency, indicated by the average buffer displacement. It makes decisions regarding the quality bitrate according to a probability distribution and derives a convex constraint for the upper threshold of the buffer.

LoL+[77] offers both heuristic and learning-based approaches, optimising towards a defined Quality of Experience equation. It consists of two modules: playback control and throughput estimation. The playback control module optimises the Quality of Experience in low latency live adaptive streaming environments, while the throughput estimation module focuses on providing accurate throughput measurements in environments where the combination of CMAF chunks and HTTP/1.1 Chunked Transfer Encoding is used. LoL+ defines a Quality of Experience model for both approaches which compromises of the following individual QoE factors: quality bitrate, quality switches, rebuffering duration, live latency, and playback speed. The heuristic-based algorithm builds upon FastMPC[146], and calculates the potential QoE for all possible decisions for a number of segments ahead, using the new QoE model which incorporates live latency and playback speed – QoE factors which are not present in the MPC algorithms. The second approach, the learn-based algorithm, utilises the Self Organising Maps (SOM) [69] technique used for unsupervised classification problems.

Stallion[48] uses a sliding window to measure the arithmetic mean and standard deviation of throughput and request latency. The algorithm measures the arithmetic mean of throughput and adjust it by subtracting the standard deviation of throughput multiplied by the throughput safety factor. Latency is measured, referring to the latency of HTTP requests, as the arithmetic mean and adjusted by adding the standard deviation of latency multiplied by the latency safety factor. This approach allows the ABR algorithm to detect fluctuations in bandwidth, as well as the delay of the link. The highest quality bitrate sustainable on the adjusted throughput and latency measurements is selected.

3.3.2 Server-based adaptation

In server-based adaptations, ABR algorithms utilise the server alone to measure the network conditions in order to select appropriate quality bitrate for each client in the streaming session. Quality bitrate is selected by the server only, requiring the server to keep the state of each client which reduces the scalability of the overall system as it requires specialised HTTP servers. However, this approach can significantly improve quality bitrate fairness among clients sharing the same link, leading to a more uniform Quality of Experience across the clients. Estimation of each client's available bandwidth at the server can be challenging, and incorrect measurements can lead to unsustainable quality bitrate selections causing interrupted playback, and in turn detrimental Quality of Experience.

Server-based adaptation can be performed by utilising traffic shaping methods as seen in [2] and [52]. Server-based adaption proposed in [26] focuses on adjusting the quality bitrate to ensure a stable buffer level of each client connected to the server. Other adaption solutions have been proposed such as [30] which modify the manifest file, and [14] where the segments can be fetched from multiple servers. Server-based ABR algorithms can be used to deploy efficient caching mechanisms as the content served is fully controlled by the server.

3.3.3 Network-assisted adaptation

Network-assisted adaptations consist of ABR algorithms that keep the adaption logic at the client but are enhanced by information from the network, which includes statistics regarding network conditions as well as suggested quality bitrates. This is done by a proxy node, deployed in the network, which monitors the network conditions and transmits the network-level information to the client. The client can utilise this additional information to aid better selection of appropriate quality bitrate and can ignore any quality bitrate suggestions made by the proxy node. This approach suffers from one major disadvantage, the additional network overhead, which can increase significantly as both the complexity of information and the update rate increase.

Network-assisted adaptation which focuses on improving the Quality of Experience of clients was proposed in [91] and [12]. The ABR algorithm proposed in [71] focuses on predicting clients' buffer levels based on the network traffic. Adaptation solution presented in [24] proposed a distributed system where the central node aims to improve the Quality of Experience of all clients. Further adaption solutions have been proposed in [103] and [66] to address the issue of fairness of Quality of Experience between clients. Common bottleneck problems are especially persistent in mobile networks with several ABR algorithms proposed over the years to address this issue such as [144], [147], and [49].

3.3.4 Hybrid adaptation

In hybrid adaptations, ABR algorithms consist of many nodes in the network collaborating to aid client quality bitrate selection. They can be classified into SDN-based and server-and-network-assisted ABR algorithms. SDN-based adaptations allow for global monitoring and control of the quality bitrate of clients in the network, which can be especially useful in a network where many clients share the same bottleneck, competing for bandwidth leading to unfair bandwidth allocation and unstable quality bitrate.

In [42] and [37] SDN-based adaptation solutions which focus on improving Quality of Experience fairness have been presented. SDN-based adaption with global network monitoring was presented in [97] and [135]. Adaptation solution proposed in [104] focused on reducing rebuffering by implementing a prioritisation process that can help clients with low client buffer. Server-and-network assisted adaptation focuses on utilising communication between the clients and servers with the intention of sharing metrics and feedback that can aid quality bitrate selection. Server-and-network assisted ABR algorithms have been proposed in [126] and [127].

3.4 Summary

In this chapter, we have discussed the related work to the research presented in this thesis. In the first section, we have examined the advancements in low latency live adaptive streaming, including the review of latency contributors in live adaptive streaming as well as the utilisation of the recently standardised Common Media Application Format (CMAF).

In the second section, we have discussed the relevant literature on the Quality of Experience in adaptive streaming. We have outlined the individual contributing QoE factors and described the studies investigating the dynamic relationships between them. Additionally, we have outlined the existing QoE models which combine all or some of the individual QoE factors in order to produce a single metric, which can be used to assess the overall QoE of streaming sessions in adaptive streaming.

In the third section, we have examined the existing ABR algorithms designed for adaptive streaming. These algorithms are tasked with the selection of an appropriate video quality bitrate for each segment, based on the estimates of the ever-changing network conditions, aiming to maximise the Quality of Experience of a streaming session. We have described existing adaptation solutions, including the state-of-the-art client-based low latency ABR algorithms, published the same year as the proposed adaptation solution in this thesis.

Chapter 4

Llama: QoE Analysis

Quality of Experience (QoE) is an essential component of HTTP Adaptive Streaming, however, the relationships between the different contributing QoE factors are complex and dynamic. As seen in the previous chapter, one of the main QoE factors is rebuffering, where the playback is interrupted due to the client buffer being fully depleted before the next consecutive segment is fully fetched. Rebuffering can have a significant negative impact on the overall QoE, depending on its duration, frequency, and temporal location [90, 143, 107, 5, 3, 149, 134, 32, 31].

Another crucial QoE factor is the video quality, which in HTTP Adaptive Streaming can be highly dynamic as it can be adjusted every segment duration. It is usually changed to counteract fluctuating network conditions to ensure uninterrupted playback. Understanding the impact of video quality variations is difficult as it encompasses a broad spectrum of possible encoding bitrates and quality switch patterns. Current literature agrees that frequent quality switches and multi-level switches can negatively impact the overall QoE [130, 41, 121, 124].

In this chapter, we further investigate the impact of video quality changes on the perceived Quality of Experience by performing an extensive subjective study. In the first section, we describe the aims and objectives of our investigation, including the video quality patterns selected for this experiment. In the second section, we detail the methodology of our subjective study, including the testing procedure and the developed online survey tool. In the third section, we present the results of our experiment, as well as, discuss our findings. In the last section, we summarise this chapter, concluding with key findings from the subjective study.

4.1 Aims and Objectives

We have designed this subjective study to further investigate how different video quality impairments impact the overall Quality of Experience. In this experiment, we focus solely on video quality variations as the current literature is clear on the negative impact of rebuffering on the overall QoE. Additionally, rebuffering events will increase the end-to-end latency by rebuffering duration in low latency live streaming scenarios when there is no playback catch-up mechanism employed at the client. Considering these two points we have decided to treat rebuffering as the primary negative QoE factor, meaning, a low latency ABR algorithm should prioritise avoiding rebuffering, as it increases the end-to-end latency, over trying to maximise the video quality.

As described earlier, video quality variations can span over a large variety of encoding bitrates and quality switch patterns. In order to carry out this subjective study within a reasonable time and resource budget, a limited number of video quality patterns was selected for testing. Video quality patterns were divided into seven groups, each designed to test a specific video quality impairment. Each group contains from three to five video quality patterns. Figure 4.1 presents all 28 video quality patterns from the following seven experiment groups: Spike, Drop, UnstableHigh, UnstableLow, GradualHigh, GradualLow, and DoubleDrop. The average video quality bitrate of each pattern is shown in the yellow box. Each pattern was constructed of seven segments, each 2s long, resulting in a playback duration of 14s. The video quality index corresponds to bitrates of {400, 800, 1200, 2400, 4800} (kbps) with encoding resolutions {426x240, 640x360, 854x480, 1280x720, 1920x1080}.

Each experiment group was designed to test a specific video quality impairment. Group ‘Spike’ deals with a single temporary increase in video quality, where the first pattern consists of all segments at the lowest quality level while the remaining patterns

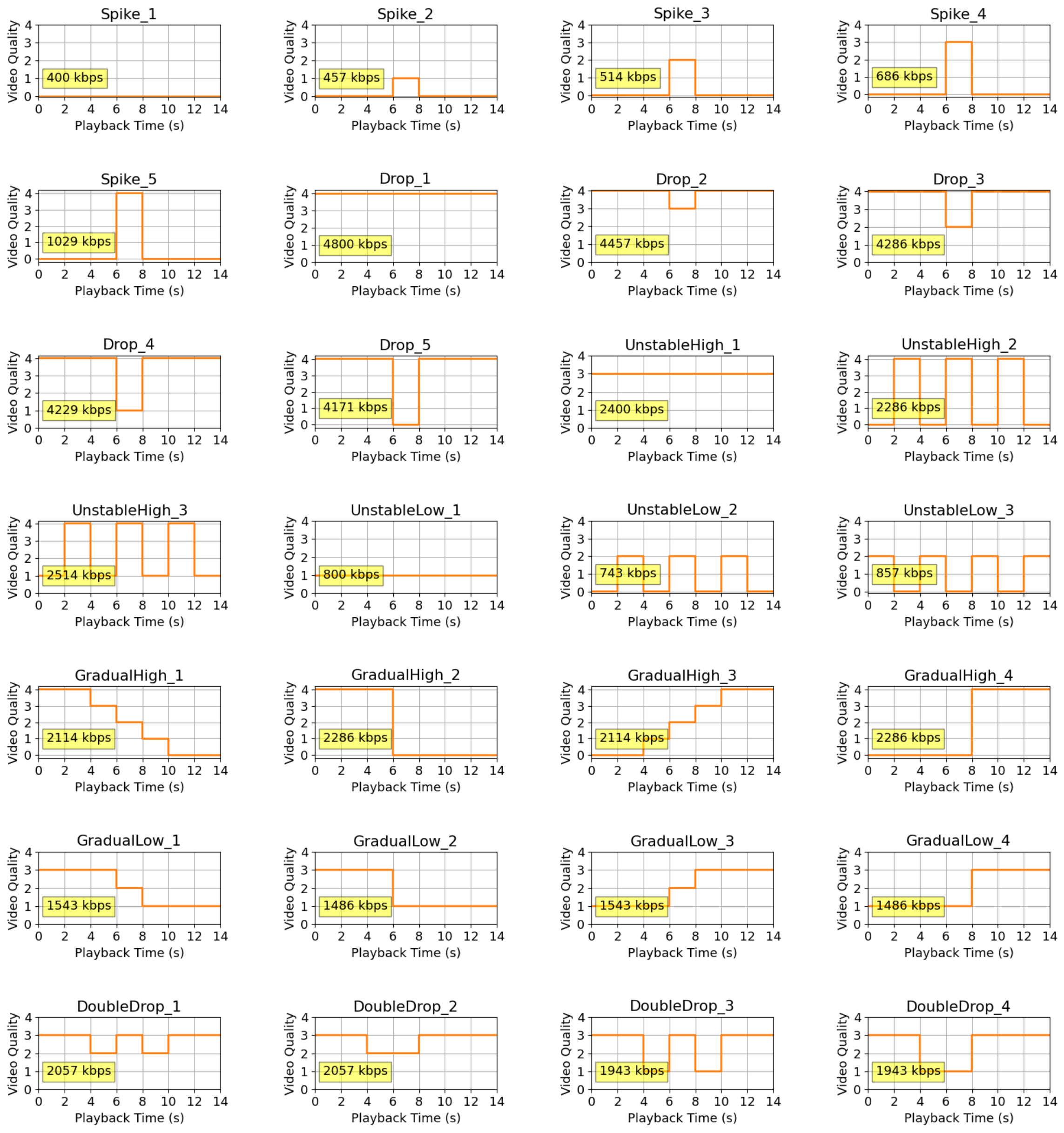

Figure 4.1: Video quality patterns used in the subjective test. Each graph shows a single pattern, with the video quality plotted against playback time, along with the resulting average bitrate shown in the yellow box.

consist of all segments at the lowest quality except the middle segment where the video quality increases by various degrees. The average bitrate varies significantly in this group, with the first pattern at 400kbps, and the last at 1028kbps. In this group, we investigate the impact of a temporary increase in video quality on the overall Quality of Experience, since it can significantly increase the average bitrate. Typically, an ABR algorithm will try to maximise the video quality presented to the user, increasing it when the network conditions improve, however, if the ABR overestimates the new available bandwidth, it will have to correct the video quality and reduce it to avoid rebuffering.

Group 'Drop' tests a single temporary decrease in video quality of various magnitudes. The first pattern consists of all segments at the highest quality level, while the remaining patterns consist of all segments at the highest quality except the middle segment where the video quality decreases by various degrees. This group consists of patterns with high variance in average bitrate, with the first pattern at 4800kbps and the last at 4171kbps. An ABR algorithm might need to decrease the video quality in order to avoid rebuffering when the network conditions worsen, however, in a case of an overreaction, that is, when the network conditions immediately improve or the video quality was reduced by too many levels, the drop in quality might have been unnecessary. In this group, we investigate the impact of such, unnecessary, temporary video quality drops on the overall Quality of Experience.

Group 'UnstableHigh' compares constant video quality against two forms of unstable quality, where the average bitrate of the two latter patterns is within 5% of the former. The first pattern offers constant second highest video quality. In the two remaining patterns, the video quality oscillates every segment duration between the lowest and the highest quality level in pattern 2, as well as, the second lowest and the highest quality in pattern 3. In environments with unstable network conditions, an ABR algorithm that uses a smoothed estimate of the available bandwidth will be able to provide a constant video quality, given it employs a sufficient client buffer, however, an ABR algorithm that uses a fine-grain throughput estimation method will change the video quality frequently, leading to highly unstable video quality. In this group, we investigate the impact of both approaches, where the network instability is high, on the overall Quality of Experience.

Group 'UnstableLow' also compares constant video quality against two patterns of unstable video quality, with the average bitrates of the latter being within 7% of the average bitrate of the former. In this group, the instability of video quality in the two latter patterns is significantly smaller. The first pattern offers constant second lowest video quality. The two remaining patterns consist of video quality oscillating every segment duration between the lowest and the middle quality level. The only difference between these patterns is that one starts and ends at the lowest quality level, while the other pattern starts and ends at the middle quality level resulting in a higher average bitrate. Similarly to the previous group, we investigate the impact of unstable video quality on the overall Quality of Experience, however, in this group the video quality instability is of a much smaller magnitude.

Group 'GradualHigh' focuses on comparing gradual and instant multi-level video quality switches, that is, where the video quality is switched between more than two consecutive quality levels. In the first two patterns, the video quality is switched from the highest to the lowest, where in the first pattern the switch is performed gradually, meaning, the video quality is switched by one level every segment duration, and in the second pattern the switch is performed instantly. The two remaining patterns contain reversed video quality switches, where the video quality is switched from the lowest to the highest, gradually and instantly. The average bitrate in this group ranges from 2114kbps to 2285kbps. An ABR algorithm can change the video quality every segment duration, by any number of encoding bitrates, meaning, the video quality could be changed from the lowest to the highest between two consecutive segments. In this group, we investigate the impact of instant multi-level video quality switches, when compared to gradual and slower switches over the entire encoding bitrate spectrum.

Group 'GradualLow' also focuses on comparing gradual and instant multi-level video quality switches, however, of much smaller magnitude. In the first two patterns, the video quality is changed from the second highest to the second lowest, where in the first pattern the change occurs gradually, and in the second pattern the change occurs instantly. In the two remaining patterns, the video quality is switched from the second lowest to the second highest, gradually in the first pattern, and instantly in the second one. The average bitrate ranges between 1485kbps and 1542kbps. In this group, we investigate the impact of instant multi-level video quality switches, when compared to gradual and slower switches, similar to the previous group, however, in this group the video quality is changed by only two levels.

Group 'DoubleDrop' tests video quality drops of various frequencies and the same duration, over two orders of magnitude. In the first two patterns, the video quality decreases by one level, for the period of one segment, which occurs twice, in the first pattern, while in the second pattern the video quality drops for the period of two segments, which occurs only once. In the two remaining patterns, the video quality decreases by two levels, for the duration of one segment, twice, in the first pattern, and for the duration of two segments, once, in the second pattern. The average bitrate in the first two patterns equalled to 2057kbps, while in the two remaining patterns, it equalled to 1942kbps. Patterns in this group simulate ABR behaviour, in which, the algorithm either, waits for the conditions to fully improve and then increases the video quality back again, or, tries to increase the video quality back too early which leads to another decrease in video quality followed by an increase back.

4.2 Methodology

In this section, we detail the methodology behind our subjective study. It consists of a developed online survey and follows a standard testing procedure to investigate how the video quality patterns, detailed in the last section, are perceived by users.

The subjective test was performed using an Absolute Category Rating (ACR) method, also known as the single stimulus method, according to the ITU-T Recommendation P.910 [57]. In the ACR method, participants are presented with one short test sequence at a time. Immediately after viewing the test sequence, the participants are asked to evaluate its quality by rating it. The following scales are proposed to record a participant's opinion of each test sequence. Five-level scale: {1 - Bad, 2 - Poor, 3 -

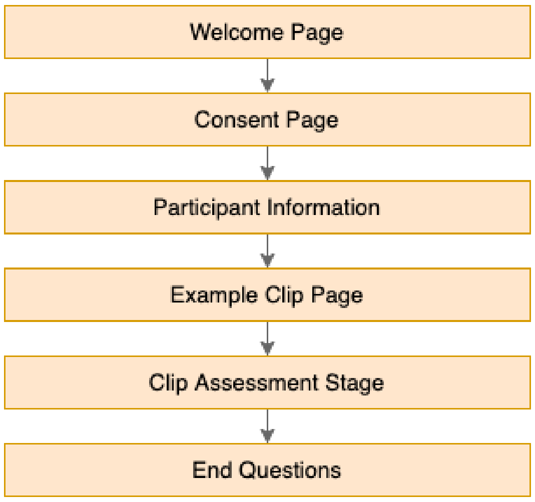

Figure 4.2: Structure of the online survey used in the subjective study.

Fair, 4- Good, 5- Excellent} and nine-level scale {1 – Bad, 2, 3 – Poor, 4, 5 – Fair, 6, 7 – Good, 8, 9 – Excellent}, with the latter scale recommended for assessment of low bitrate encoded content, hence the former was selected for our subjective test. ACR method also specifies that the participant must not see variations of the same content twice in the row, and that, only the scores of variations of the same content, all viewed by the same participant, can be fairly compared and analysed. Participant responses were processed according to ITU-R Recommendation BT.500-13 [56]. It specifies that for each test sequence, the following metrics need to be calculated: Mean Opinion Score (MOS), the Standard Deviation of the scores, as well as a (95%) Confidence Interval. The MOS of each test sequence is calculated as an arithmetic mean of all scores given by the participants for the given test sequence, where the score is mapped into an integer according to the scale, where 5 equals to the 'Excellent' response.

In order to carry out the experiment remotely, an online survey capable of presenting test sequences one at a time, without adding unintended quality impairments, was developed. The survey could be accessed by a specific URL, which was sent out along with invitations to potential participants, and completed anonymously. Figure 4.2 shows the structure of the survey. Upon starting the survey, a participant was subjected to a welcome page that described the study and gave an approximate completion time of 15 minutes. The next few pages asked the participant for consent (Appendix B) as well as to self-report their gender and age. A participant would then continue to the example page, which would train the participant to be able to play the test sequences, as well as demonstrate what various video qualities look like. After the example page, the participant would proceed to the subjective test, during which, the participant was presented with one test sequence per page along with the five-level scale to record the participant's response. A participant could proceed to the next test sequence only after rating the current test sequence. A participant could not go back to previous test sequences. Each test sequence was fully downloaded before the participant could begin playback to ensure that no unintended rebuffering occurred during playback. Once the test clip was fetched, the participant needed to click on the playback button to begin watching the clip. Upon clicking the playback button, the clip was played in full-screen mode, after which full-screen mode was exited and the scoring scale displayed. The survey restricted each participant to only one set of responses and recorded the operating system, web browser and screen resolution used by the participant. The last page asked the participants if they have watched the clips in full-screen mode and with sound on.

In order to create test sequences, based on the video quality patterns detailed in the previous section, high-quality unaltered video clips were required, to which the quality impairments could be applied to. A wide range of video clips was sourced by capturing content from the live BT Sport service which offers a variety of live sports content. The following set of 14 video clips was captured: {Tennis_1, Tennis_2, FormulaOne_1, FormulaOne_2, Rugby_1, Rugby_2, Football_1, Football_2, Squash_1, Squash_2, Rally_1, Basketball_1, Basketball_2, Basketball_3}. Sports content was selected as it is the most common type of content transmitted using low latency live streaming. Each video clip was 14 seconds long and encoded at 25 frames per second. Figure 4.3 shows a thumbnail image of each of the selected video clips. As per ITU-T Recommendation P.910, the Temporal and Spatial Information of each video clip was calculated and presented in Figure 4.4. In order to create a test sequence for a given video quality pattern, first, a video clip was encoded at the five bitrates and resolutions detailed in the last section,

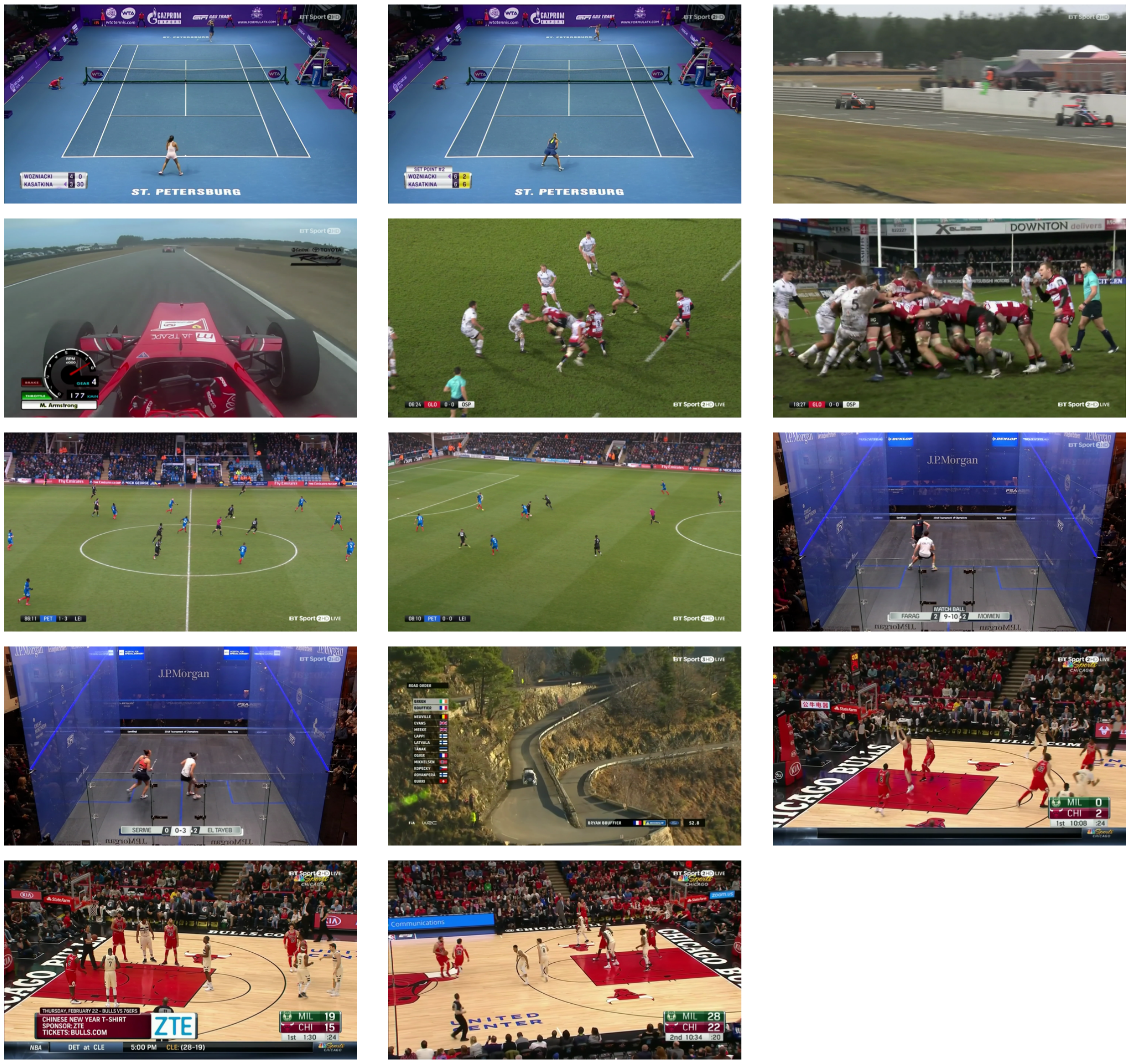

Figure 4.3: Thumbnails of selected video clips for the subjective study, in the following order (from left to right): Tennis_1, Tennis_2, FormulaOne_1, FormulaOne_2, Rugby_1, Rugby_2, Football_1, Football_2, Squash_1, Squash_2, Rally_1, Basketball_1, Basketball_2, Basketball_3.

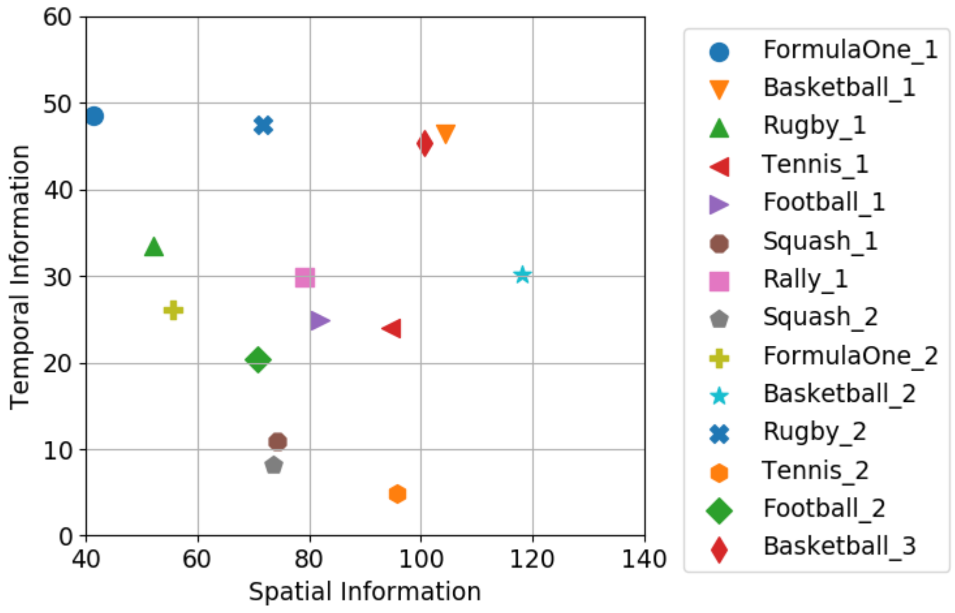

Figure 4.4: Spatial and Temporal Information of each video clip.

using VBR encoding, then segmented into 2s segments. Each segment was then scaled up to the highest resolution to avoid playback issues in some web browsers. Finally, a test sequence was created by concatenating, without re-encoding, the consecutive up-scaled segments, each of the quality level specified by the video quality pattern.

Six test sets were created by using three sets of test sequences, each with two orders of presentation. The two orders of presentation were created randomly, with one constraint which dictates that test sequences belonging to the same experiment group must not be presented consecutively. This ensures that the same video clip is not shown to a participant twice in a row, as per guidelines published in ITU-T Recommendation P.910. A single test set was presented to a participant, consisting of one of the three sets of test sequences, as well as, using one of the two orders of presentation. Three sets of test sequences were created, A, B, and C, each including video quality patterns from every group. Table 4.1 shows the video clips assigned to each experiment group and test set. Figure 4.5 displays the calculated objective metric, Video Multi-Method Assessment Fusion (VMAF)[75], of the resulting test sequences.

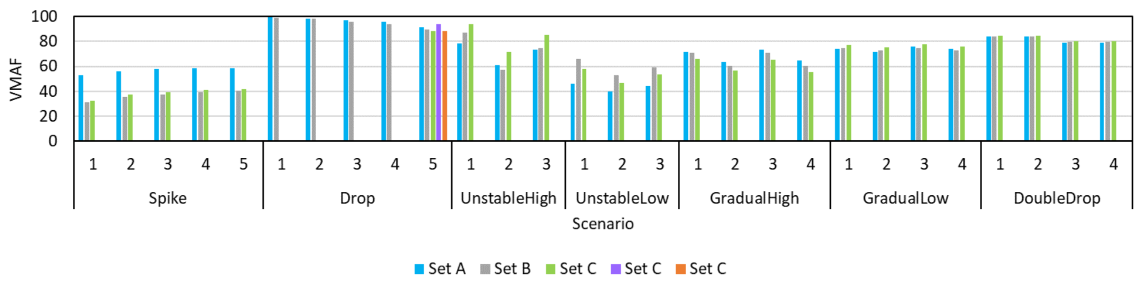

Figure 4.5: VMAF score of each test sequence used in the subjective study.

Table 4.1: Source clips assigned to each experiment group and test set.

Pattern	**Set A**	**Set B**	**Set C**
Spike_1-5	FormulaOne_1	Squash_2	Tennis_1
Drop_1-4	Basketball_1	FormulaOne_2	
Drop_5	Basketball_1	FormulaOne_2	Tennis_1, FormulaOne_1, Basketball_2
UnstableHigh_1-3	Rugby_1	Basketball_2	FormulaOne_1
UnstableLow_1-3	Tennis_1	Rugby_2	Basketball_2
GradualHigh_1-4	Football_1	Tennis_2	Rally_1
GradualLow_1-4	Squash_1	Football_2	Football_1
DoubleDrop_1-4	Rally_1	Basketball_3	Squash_1

As per guidelines published in ITU-T Recommendation P.910 regarding the ACR testing procedure, a fair comparison between Mean Opinion Scores of two test sequences can be made only if they were created using the same video clip. This means that to allow comparisons to be made between patterns within each experiment group, a common video clip was required for all of the patterns within each group. Two unique video clips were assigned to each of the seven groups to be presented with each video quality pattern in that group. In each group, one of these video clips was allocated to a first set of test sequences (A), and the other to a second set (B). This ensures that a participant was presented with test sequences created using the same video clip for all patterns belonging to a single group.

To allow some comparison between patterns belonging to different experiment groups, one more set of test sequences (C) was defined. Groups were further aggregated into two supergroups, first based on groups {Spike, Drop_5, UnstableHigh, and UnstableLow}, and the second on groups {GradualHigh, GradualLow, DoubleDrop}. Within each supergroup, each experiment group was assigned an additional video clip, which corresponds to a video clip from test sequence set A or B of another group within the same supergroup. For example, group DoubleDrop was assigned video clip Squash_1 which belongs to the test sequence set A of group GradualLow, allowing for comparison of Mean Opinion Scores between all 8 video quality patterns from the two groups. During our preliminary testing, we discovered that there was little benefit in more extensive comparisons of patterns Drop_1-Drop_4, and hence we decided to include only pattern Drop_5 within the first supergroup. Drop_5 is the only pattern that was assigned three additional video clips instead of just one, each corresponding to video clips assigned to groups {Spike, UnstableHigh, and UnstableLow}.

Once the online survey was developed, potential participants were invited using email invitations (Appendix C) with the Participant Information Sheet attached (Appendix A), sent out to students as well as staff. In total, 63 participants were recruited, with a median age group of 35-44. The age distribution of the recruited participants is shown in Figure 4.6. Participation in the study was voluntary and we did not offer rewards or incentives for completing the study to try to maximise the

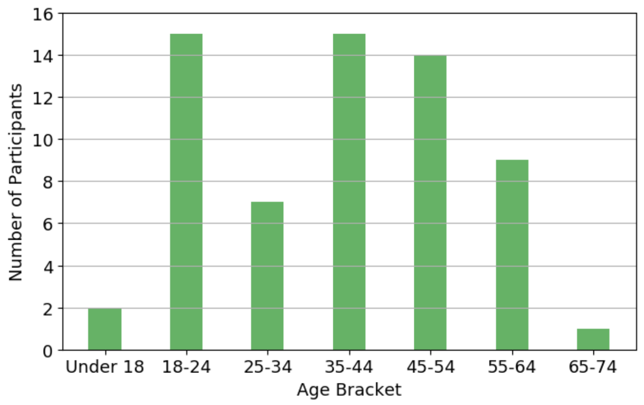

Figure 4.6: Age distribution of all recruited participants for the subjective study.

reliability of the responses given. 14% of responses were completed on a mobile device and 86% on a desktop or laptop. The responses were screened using the methodology described in ITU-R Recommendation BT.500-13 [56] which resulted in one response being rejected. Additionally, the responses were screened by comparing the survey completion time with the total playback duration of all clips. If the completion time was lower than the total playback time, it would indicate that the participant has not watched the test sequences before rating them and their response should be rejected; however, no such cases were found. This study was approved by the faculty's ethics committee.

4.3 Results and Discussion

In this section, we present the results of our subjective study, as well as, discuss the findings. In the first subsection, we compare the Mean Opinion Scores of video quality patterns within each experiment group individually. In the second subsection, we compare the Mean Opinion Scores of patterns within each supergroup. Throughout this section, the presented results include combined responses from both desktop/laptop and mobile devices as only 14% of responses were completed using the latter – not enough for a fair comparison between the responses from the two categories.

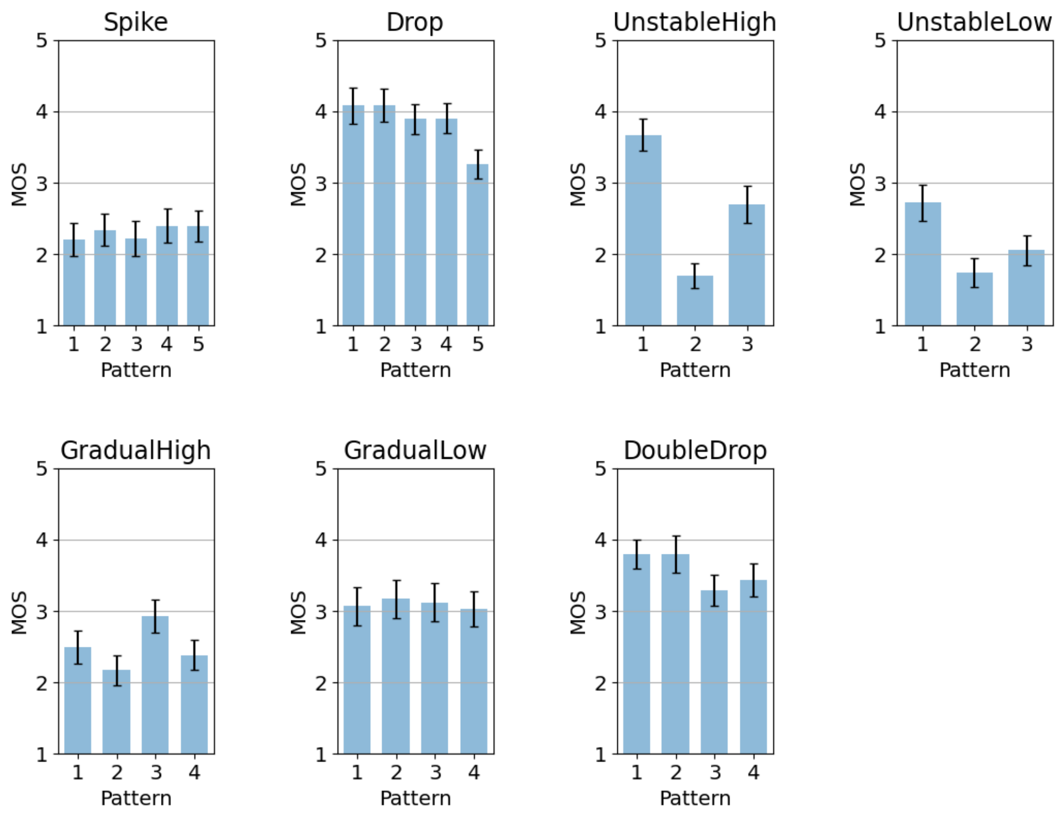

Figure 4.7: Mean Opinion Scores of common test sequences within each experiment group with Confidence Interval of 95% plotted in the error bars.

4.3.1 Comparison of patterns within groups.

In this subsection, we present and analyse the results of our subjective test, group by group. Figure 4.7 shows the average Mean Opinion Scores (MOS) of test sequences for each video quality pattern, divided by experiment group. The error bars present the Confidence Interval of 95%. As shown in Table 4.1, the same three video clips were used for each pattern within a group, except for group 'Drop' for which only two video clips were used for all of its patterns. Additional three video clips were used for pattern 'Drop_5', but the Mean Opinion Scores for these test sequences were omitted from Figure 4.7 to ensure a fair comparison between all five patterns within the group 'Drop'.

Group ‘Spike’ consists of video quality patterns where a temporary increase, of various magnitude, in video quality occurs. The improvement in video quality takes place halfway through the pattern and lasts for the duration of one segment (2s). Pattern 1 contains no video quality improvement as it remains constant at the lowest quality level. In patterns 2-5, all segments are encoded at the lowest quality level with the exception of the middle segment, which, is encoded at various higher quality levels, covering the entire video quality spectrum. The first chart in Figure 4.7 presents the Mean Opinion Scores of patterns within the group ‘Spike’. We can observe that the differences in MOS of all five patterns were statistically insignificant, as they were lower than the Confidence Interval of each pattern. This suggests that a temporary improvement in video quality does not improve the overall Quality of Experience, even if it results in a higher average bitrate as seen in pattern 5, which had more than double the average bitrate of pattern 1.

Patterns belonging to the group ‘Drop’ focus on testing various magnitudes of a temporary decrease in video quality, lasting a single segment duration and occurring in the middle of the test sequence. Pattern 1 consists of all segments at the highest quality level, resulting in an average bitrate of 4800bkps. In patterns 2-5, all segments are at the highest quality level, except the middle segment, which is encoded at a lower quality level, by various degrees to cover the entire video quality spectrum. Patterns 2, 3, 4, and 5 had an average bitrate of 4457, 4285, 4228, and 4171kbps respectively. We can observe differences ranging 0.01-0.19 in Mean Opinion Scores of patterns 1-4, however, these differences were statistically insignificant as they were smaller than the Confidence Interval of each pattern. On the other hand, pattern 5 achieved a significantly lower Mean Opinion Score than patterns 1-4, with differences between 0.63 and 0.82. These results suggest, that, a temporary decrease in video quality does not have a significant negative impact on the overall Quality of Experience unless the temporary decrease results in the lowest quality level.

Group ‘UnstableHigh’ compares constant video quality against two patterns of unstable video quality. In pattern 1, each segment was encoded at the second highest quality level, resulting in an average bitrate of 2400kbps. In pattern 2, the video quality oscillates every segment duration between the lowest and highest, with an average bitrate of 2285kbps. In pattern 3, the video quality oscillates every segment duration between the second lowest and the highest, resulting in an average bitrate of 2514kbps. We can observe a statistically significant difference between Mean Opinion Scores of all three patterns, with pattern 1 being scored 1.97 and 0.96 higher than patterns 2 and 3 respectively. This suggests that constant video quality, as presented in pattern 1, is much preferred over unstable video quality, as presented in patterns 2 and 3, with the smaller amplitude of oscillations in pattern 3 being preferred to the larger oscillations of pattern 2.

Group 'UnstableLow', similar to the previous group, compares constant video quality against two forms of unstable video quality. Pattern 1 presents all segments encoded at the second lowest quality level, with an average bitrate of 800kbps. Patterns 2 and 3 contain video quality oscillation every segment duration between the lowest and the middle quality level, with the former starting and ending at the lowest, while the latter pattern starts and ends with the middle quality. The average bitrate equals to 742 and 857kbps for patterns 2 and 3 respectively. We can observe statistically significant differences between Mean Opinion Scores of all three patterns. Patterns 2 and 3 were scored 0.98 and 0.66 respectively lower than pattern 1. This reinforces the conclusion from the previous group, suggesting that constant video quality is preferred over unstable video quality, with smaller oscillations in video quality being preferable over larger oscillations as pattern 3 was scored 0.4 higher than pattern 2.

Group 'GradualHigh' tests gradual and instant multi-level video quality switches. In pattern 1, the video quality gradually decreases from the highest to the lowest, decreasing by one level every segment duration, while, in pattern 2 the video quality changes instantly, decreasing by the entire bitrate spectrum in one segment duration. The two remaining patterns, 3 and 4, present the reversed video quality of the first two patterns, with the quality increasing from the lowest to highest, gradually in the former, and instantly in the latter pattern. The average bitrate of patterns with instant video quality switches was 171kbps higher when compared to the patterns where a gradual switch occurs. We can observe that the gradual video quality switches, presented in patterns 1 and 3 with MOS of 2.49 and 2.93 respectively, were preferred to the instant video quality switches, presented in patterns 2 and 4 with MOS of 2.17 and 2.38 respectively. Additionally, we can observe that the improving video quality, presented in patterns 3 and 4, was preferred over deteriorating video quality, presented in patterns 1 and 2.

Group 'GradualLow' further tests gradual and instant multi-level video quality switches, but with changes of smaller magnitude. In pattern 1 the video quality gradually decreases from second highest to second lowest, while in pattern 2 the video quality decreases instantly. In the two remaining patterns, the video quality increases from the second lowest to the second highest, with the increase being gradual in pattern 3, and instant in pattern 4. We can observe no statistically significant differences in Mean Opinion Scores of all four patterns. This suggests that gradual switches over two quality levels have little to no positive impact on the overall Quality of Experience when compared to instant switches.

Group 'DoubleDrop' compares patterns with video quality drops of various frequencies and the same duration. In patterns 1 and 2, the video quality decreases by one level, while in patterns 3 and 4 it decreases by two quality levels. The video quality drops for a duration of one segment, twice, in patterns 1 and 3, while it drops for a duration of two segments, once, in patterns 2 and 4. We can observe that differences between Mean Opinion Scores of patterns 1 and 2, as well as patterns 3 and 4, were statistically insignificant, suggesting that video quality deterioration frequency has no impact on the overall Quality of Experience, as long as, the duration is of the deterioration is the same. Additionally, we can observe that patterns 1 and 2 were scored higher than patterns 3 and 4, which was expected as the video quality drops by an additional level in the latter patterns.

4.3.2 Comparison of patterns within supergroups.

As per the ACT testing procedure described in ITU-T Recommendation P.910, we can only compare Mean Opinion Scores of test sequences created using the same video

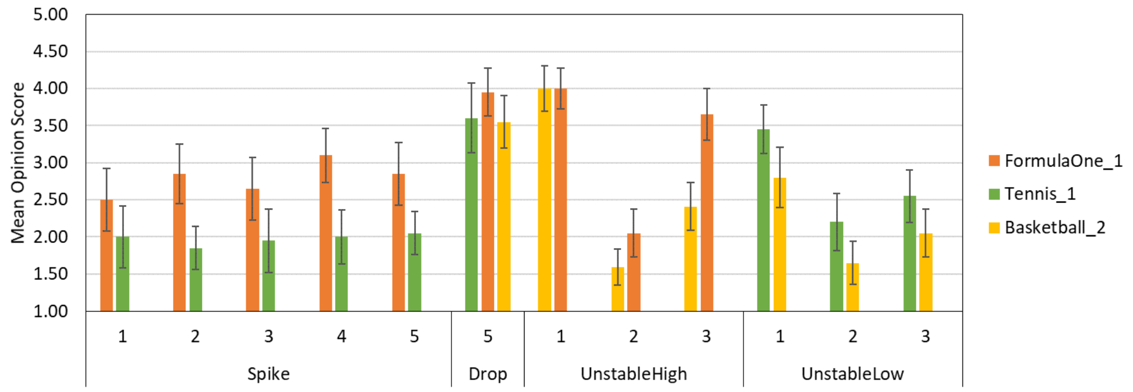

Figure 4.8: Mean Opinion Scores of all test sequences for patterns within the first supergroup with the 95% Confidence Interval plotted in the error bars. Each source clip is plotted in a different colour.

clip. Hence, in order to compare Mean Opinion Scores of video quality patterns belonging to different groups, we have created two supergroups: {Spike, Drop_5, UnstableHigh, and UnstableLow}, and {GradualHigh, GradualLow, DoubleDrop}. Within each supergroup, every group has an assigned video clip in common with another group allowing for comparison of Mean Opinion Scores of patterns in different groups. Figure 4.8 shows the Mean Opinion Scores of test sequences for all patterns within the first supergroup, with the 95% Confidence Interval plotted in the error bars. Mean Opinion Scores were aggregated by the video clip used to create the test sequences.

First, we compare pattern Drop_5 with the rest of the patterns in the first supergroup, using the common video clips. Drop_5 had an average bitrate of 4171kbps and was scored higher than most other patterns from groups Spike, UnstableHigh, and UnstableLow. This was expected as the second highest bitrate equalled to 2400kbps in our bitrate spectrum. However, we can observe the following exceptions. The difference between Mean Opinion Scores of patterns Drop_5 and UnstableHigh_1, where all segments were presented at the second highest video quality, was statistically insignificant, with the average bitrate of the latter pattern being 42% lower. Patterns Drop_5 and UnstableHigh_3 have similar MOS with video clip FormulaOne_1, however, Drop_5 had much lower MOS with video clip Basketball_2. Pattern UnstableLow_1, with an average bitrate 81% lower and where all segments were presented at the second lowest video quality, achieves only slightly lower MOS than Drop_5. This suggests that the benefit for Drop_5 of six high bitrate segments at the highest quality is lost by having one segment at the lowest quality, and that more effective use of throughput could be made by having all segments at the same quality, even if that quality is much lower.

Secondly, we compare patterns Spike_1-5 and UnstableHigh_1-3 using the common video clip FormulaOne_1. Patterns belonging to group UnstableHigh had an average bitrate ranging from 2285 to 2514kbps, while Spike patterns had an average bitrate in the range of 400-1028kbps. Pattern UnstableHigh_1, where all segments were presented at the second highest video quality, was scored 0.9-1.5 higher than patterns Spike_1-5. Pattern UnstableHigh_2, where the video quality oscillates every segment duration between the lowest and the highest video quality, was scored 0.45-1.05 lower than patterns Spike_1-5. Pattern UnstableHigh_3, where the video quality oscillates very segment duration between the second lowest and the highest video quality, was scored 0.55-1.15 higher than patterns Spike_1-5. From these results, we can observe that a higher average bitrate does not always correlate with better Quality of Experience.

Thirdly, we compare patterns Spike_1-5 and UnstableLow_1-3 using the common video clip Tennis_1. The average bitrate range equalled to 400-1028kbps and 742-857kbps for patterns belonging to groups Spike and UnstableLow respectively. Pattern UnstableLow_1, where all segments were presented at the second lowest video quality, was scored significantly higher than patterns Spike_1-5. Patterns UnstableLow_2 and UnstableLow_3 contain video quality oscillation every segment duration between the lowest and the middle quality, with the former starting and ending at the lowest, while the latter pattern starts and ends with the middle quality. We can observe a statistically insignificant difference between Mean Opinion Scores of patterns UnstableLow_2 and Spike_1-5, however, pattern UnstableLow_3 was scored higher than patterns Spike_1-5. These results reinforce the observation made in comparison between groups Spike and UnstableHigh, suggesting, that a higher average bitrate does not always lead to better

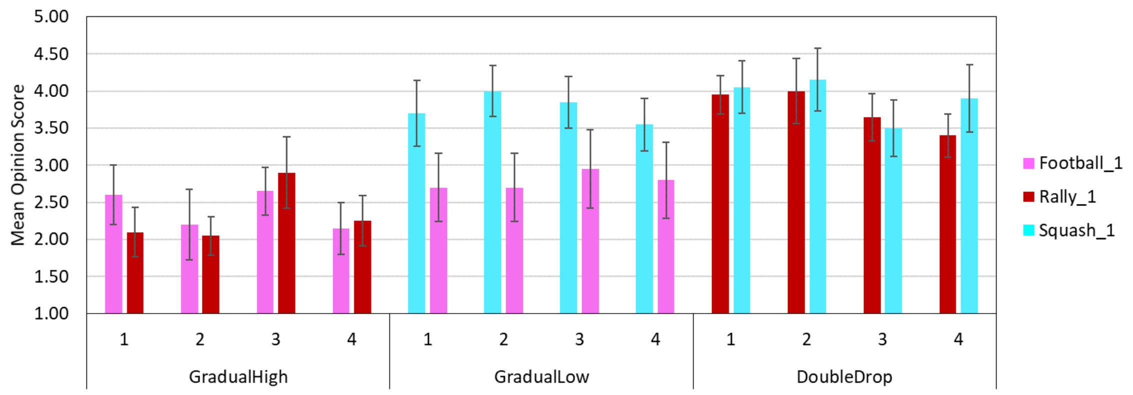

Figure 4.9: Mean Opinion Scores of all test sequences for patterns within the second supergroup with the 95% Confidence Interval plotted in the error bars. Each source clip is plotted in a different colour.

Quality of Experience.

Finally, we compare patterns UnstableHigh_1-3 and UnstableLow_1-3 using the common video clip Basketball_2. The range of average bitrate of patterns in groups UnstableHigh and UnstableLow equals to 2285-2514kbps and 742-857kbps respectively. The first pattern in each group offers constant video quality, the second highest and the second lowest in UnstableHigh and UnstableLow respectively. Both of these patterns were scored significantly higher than the remaining four patterns from both groups. This further reinforces the observation that constant video quality leads to better Quality of Experience than unstable video quality, even if it results in a significantly lower average bitrate. Patterns UnstableHigh_2 and UnstableLow_2 were scored similarly, despite the former having approximately three times higher average bitrate. Pattern UnstableHigh_3 was scored slightly higher than UnstableLow_3, with the latter pattern having approximately three times lower average bitrate. These results further reinforce the observation made in the previous compression, as unstable video quality can lead to significantly degraded Quality of Experience despite a much higher average bitrate.

Figure 4.9 shows the Mean Opinion Scores of all test sequences for patterns within the second supergroup with the 95% Confidence Interval plotted in the error bars. As with Figure 4.8, MOS can only be fairly compared between patterns using the same video clip as the base for the test sequences.

First, we compare patterns GradualHigh_1-4 and GradualLow_1-4 using the common video clip Football_1. Both groups contain patterns that test gradual and instant multi-level video quality switches, with the video quality switching across the entire spectrum in group GradualHigh, and across two quality levels in group GradualLow. Each pattern in group GradualLow has higher MOS than the corresponding pattern in group GradualHigh despite having approximately 32% lower bit rate, although none of the differences are statistically significant.

Secondly, we compare patterns GradualHigh_1-4 and DoubleDrop_1-4 using the common video clip Rally_1. Group GradualHigh tests gradual and instant multi-level video quality switches across the entire bitrate spectrum, with an average bitrate range of 2114-2285kbps, while group DoubleDrop evaluates video quality drops of various frequencies, with the average bitrate range of 1942-2057kbps. Patterns belonging to group DoubleDrop were scored higher than patterns of group GradualHigh, despite having approximately 10% lower average bitrate. This suggests that fewer variations of video quality are preferred, as there were fewer changes in quality in patterns belonging to group DoubleDrop.

Finally, we compare patterns GradualLow_1-4 and DoubleDrop_1-4 using the common video clip Squash_1. Group GradualLow tests gradual and instant multi-level video quality switches across two quality levels, with an average bitrate range of 1485-1542kbps, while group DoubleDrop evaluates video quality drops of various frequencies and duration, with the average bitrate range of 1942-2057kbps. We can observe no statistically significant differences between MOS of patterns GradualLow_1-4 and DoubleDrop_1-4, despite the former having approximately 25% lower average bitrate. These results reinforce the observation made in the previous comparison, showing that fewer video quality variations improve the Quality of Experience, regardless of the average bitrate.

Table 4.2: Key findings from the subjective study.

#	**Key Finding**
1	Average video quality bitrate alone is not an accurate predictor of the overall Quality of Experience.
2	Stable video quality can offer better Quality of Experience, even with a lower average video quality bitrate.
3	Multi-level video quality switches of high magnitude are better perceived when done gradually, one level at a time, instead of instantly.
4	Temporary decrease in video quality has no impact on the Quality of Experience, except in the case of switching to the lowest quality level.

4.4 Summary

In this chapter, we have described our subjective study, designed to improve our understanding of how users perceive various video quality patterns. We have presented and analysed the results in detail. Based on these observations, we can highlight the following key findings in Table 4.2.

First, the average video quality bitrate alone is not an accurate predictor of the overall Quality of Experience. In our results, we have observed instances in which a higher average bitrate resulted in no improvement to overall QoE, as seen in the MOS of patterns within the Spike group, where the MOS differences were statistically insignificant, and the average bitrate of pattern Spike_5 was more than double the average bitrate of pattern Spike_1. We have also observed that in some cases higher average bitrate leads to worse QoE, such as, in comparison between patterns UnstableLow_1 and UnstableHigh_2, where the former achieved higher MOS while having a significantly lower average bitrate.

Second, stable video quality can offer better Quality of Experience, even with a lower average bitrate. In our results, we can observe multiple instances in which stable video quality, where fewer quality switches occur, results in equal or higher QoE than patterns with an unstable video quality of higher or similar average bitrate. Patterns belonging to group UnstableHigh had similar average bitrates, however, pattern UnstableHigh_1, which offered constant video quality, was scored significantly higher than the remaining patterns. Pattern UnstableLow_1, which offers constant video quality while having a significantly lower average bitrate, was scored significantly better than pattern UnstableHigh_2.

Third, multi-level video quality switches of high magnitude are preferred when done gradually, one level at a time, instead of instantly. In our results, particularly for the group GradualHigh, we can observe that gradual quality bitrate switches across the entire bitrate spectrum were scored significantly higher than instant switches, even with the average bitrate of the latter being 171kbps higher. However, we can also conclude that gradual quality bitrate switches offer no improvement to the overall QoE over instant switches when the quality bitrate is changed by only two encoding bitrates, as seen in the group GradualLow.

Finally, a temporary decrease in video quality has no impact on the Quality of Experience, except in the case of switching to the lowest quality level. As seen in our results for group Drop, a temporary decrease in quality bitrate, that is when the quality bitrate is reduced for the duration of a single segment, has no impact on the overall QoE. However, we can also observe multiple instances in which switching to the lowest quality bitrate leads to a significantly worse QoE. In group Drop, only pattern Drop_5, where the quality bitrate drops to the lowest for the period of one segment, was scored significantly lower. When comparing two forms of unstable quality bitrate presented in patterns UnstableHigh_2 and UnstableHigh_3, where the former involves content presented at the lowest quality bitrate, we can observe that the latter was scored significantly higher.

Based on these four key findings, we can derive QoE-oriented design goals for our new low latency ABR algorithm in Chapter 5, aiming for more bandwidth-efficient video quality switching decisions and improved overall Quality of Experience for users.

Chapter 5

Llama: ABR Design

In this chapter, we detail the design behind Low Latency Adaptive Media Algorithm (Llama), a new ABR algorithm created for low latency live adaptive streaming. In the first section, we present the aims and objectives of the new ABR algorithm. In the second section, we establish the benchmark ABR performance in low latency conditions, by analysing the performance of prominent on-demand ABR algorithms in low latency scenarios, across multiple target latency settings and realistic network conditions. In the third section, we set out the design goals for a low latency ABR algorithm, utilising key findings from the subjective study presented in the previous chapter. In the fourth section, we discuss the design principles behind the new ABR algorithm and detail its adaptation logic, as well as discuss its conformance with low latency live adaptive streaming. In the final section, we summarise this chapter.

5.1 Aims and Objectives

Adaptive streaming, including live and low latency streaming, requires an ABR algorithm capable of adjusting the video quality bitrate to counteract the changes in network conditions when transmitting multimedia over the Internet, a best-effort network. Adaptation logic is a crucial component of each streaming session as failure to adjust the quality bitrate in a timely manner will lead to playback interruption, which along with frequent and unsustainable quality switches can lead to deterioration of the overall Quality of Experience (QoE) [7].

In on-demand adaptive streaming, that is, when the content is already fully generated at the start of the streaming session and hence all segments are available to the client for consumption, the client can employ a large video buffer to give the ABR algorithm enough time to detect changes in network conditions and adjust the quality bitrate appropriately. The client video buffer holds already-fetched video segments, ready and queued for playback. For example, when the maximum client video buffer is set to 60s, it can fetch up to 60s worth of consecutive segments, giving the ABR algorithm at most 60s to adapt the quality bitrate to counter the variations in available bandwidth, before the client video buffer is fully depleted and consequently the playback is interrupted.

In live adaptive streaming, that is, when the content is generated in real-time and only the already generated segments are available to the client at the start of the streaming session, the client video buffer is limited by the target latency setting of the client. When joining a live streaming session, the client will fetch the already generated segments according to its target latency setting, for example, with the target latency of 30s, the client will start by fetching the segment approximately 30s behind the live edge and continue fetching the consecutive segments, filling the client video buffer to a maximum of 30s, before moving to fetching new segments as they are generated in real-time, every segment duration. In this case, the user consumes the content 30s after it is generated, effectively experiencing a latency of 30s. In low latency live adaptive streaming, the target latency is further decreased to reduce the latency between content being generated and consumed by the user. This significantly reduces the maximum reaction time for an ABR algorithm, creating challenging operating conditions.

The main aim of an ABR algorithm is to maximise the overall QoE, which involves achieving a balance between all contributing QoE factors [8]. Low latency live adaptive streaming, being a relatively new advancement, presents new challenges for ABR algorithms. In low latency scenarios, the ABR algorithm not only needs to maximise the QoE but also to minimise the average latency in order to maintain the desired small target latency. The most significant impact of low latency on ABR algorithms is the reduced maximum response time, as in order to achieve the low latency the target latency setting must be low, which leads to the client video buffer being drastically limited. The objective of this chapter is to design an ABR algorithm specifically tailored to overcome this challenge, that is, an ABR algorithm that can operate with a small client video buffer, while, being able to maximise the QoE and minimise the average latency.

5.2 Design Analysis

In this section, we briefly analyse and discuss the performance of four prominent on-demand ABR algorithms when deployed in low latency live adaptive streaming. As described in the earlier section, low latency conditions make bitrate adaptation challenging as they can severely limit the size of the client video buffer. The lower the target latency the smaller the maximum potential client video buffer, reducing the ABR algorithm's maximum reaction time. On-demand ABR algorithms were designed to operate with a large client video buffer and have not been tested in live, especially low latency, scenarios. This section aims to investigate the impact of low latency on current ABR algorithms, as well as, set the benchmark ABR performance for evaluation of low latency ABR algorithms.

5.2.1 Methodology

In order to analyse the performance of ABR algorithms in low latency live adaptive streaming, an experiment that utilises a simulation model along with realistic network scenarios was conducted. The simulation model was used to allow for the testing of a large number of parameters in a reasonable time. It was developed in NS-3, a discrete-event network simulator. It supports live DASH as well as CMAF chunks, along with traffic shaping between the client and the server and extensive client configuration. We discuss the simulation model in more detail in Chapter 6.

Achieving low latency in live adaptive streaming can be done by utilising regular DASH or with CMAF, where segments are further divided into smaller chunks of equal duration. This experiment examines both delivery methods and their impact on ABR performance. When in DASH mode, the client requests the content from the server, segment by segment, issuing a request to fetch a segment only once the full segment has become available on the server, simulating live content being generated in real-time. In CMAF mode, the client requests a segment once the first chunk has become available, and downloads the first chunk, after which the remaining chunks are transmitted by the server as soon as they become available, simulating the delivery of CMAF chunks via HTTP/1.1 Chunked Transfer Encoding.

Following prominent on-demand ABR algorithms were implemented for this experiment: Panda[73], Festive[65], RobustMPC[146], and Bola[120]. These ABR algorithms have not been designed for live or low latency streaming. The ABR algorithms were tuned using the settings presented in their respective papers. As detailed in the previous section, the target latency setting will have a significant impact on ABR performance in low latency adaptive streaming, with lower target latency creating more challenging conditions. Hence, for this experiment four target latency settings were selected: 2s, 4s, 6s, and 8s, each being a multiple of the segment duration (2s). There is no catch-up mechanism employed in the simulation model, meaning, the latency during a streaming session will increase every time a rebuffering event occurs, by the rebuffering duration.

The content transmitted between the client and the server was the four minutes of the BigBuckBunny movie. The clip was encoded into the bitrates of {400, 800, 1200, 2400, 4800} (kbps) with resolutions {426x240, 640x360, 854x480, 1280x720, 1920x1080}. In the case of DASH, the clip was divided into 2s segments. For CMAF, the two-second segments were further divided into 0.5s chunks, resulting in four chunks per segment. In both methods, the quality bitrate can only be changed at the segment level, that is, every segment duration. However, a CMAF chunk can be played out as soon as received by the client.

With the intention of creating realistic network conditions between the client and the server, traffic shaping on the link between them has been applied, using throughput traces based on real network data. Throughput traces were derived from CDN logs of a real commercial live TV service. The traces were cut to match the duration of the clip tested (four minutes). Traces where the mean bitrate was higher than the top bitrate of our encoded content (4800kbps), and traces where all measurements corresponded to only one quality bitrate, were excluded. This resulted in 7,000 throughput traces. We describe the throughput traces in more detail in Chapter 7.

The performance of each ABR was recorded by calculating the predicted Mean Opinion Score (MOS) of each run, using the ITU-T Rec. P.1203 QoE model which combines the bitrate, resolution, and framerate of each segment along with the rebuffering frequency and duration into a single value between 1 and 5 [108]. The standalone implementation of the model was used, which can be found here [63]. Higher MOS indicates a better perceived quality of a streaming session. Additionally, the average Live Latency was recorded. It is calculated as the arithmetic mean of the latency recorded every segment, and indicates the latency of the streaming session. In low latency live streaming, the ABR algorithm must aim to minimise the latency to achieve the desired target latency. Since there is no catch-up mechanism employed at the client in the simulation model, during a streaming session the latency will increase if a rebuffering event occurs, by the rebuffering duration, and never decrease.

5.2.2 Results and Discussion

For each low latency adaptive delivery method, four ABRs were tested using all possible combinations of the four Target Latency settings and the 7000 throughput traces. The results are presented as the arithmetic means of MOS and Live Latency for all 7000 runs per each Target Latency setting and ABR algorithm combination, divided by the delivery method.

5.2.2.1 DASH

First, we present the analysis of the performance of the four ABR algorithms when used in a regular DASH client, with the segment duration configured to 2 seconds. Figure 5.1

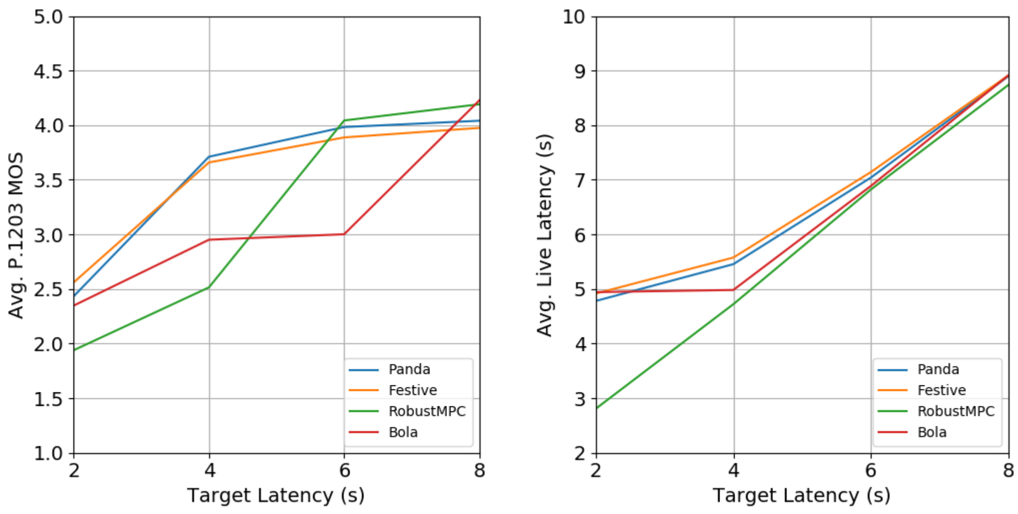

Figure 5.1: Average P.1203 MOS and Live Latency as a function of Target Latency for each ABR algorithm, when used in a DASH client.

shows the average P.1203 MOS and Live Latency across all 7000 throughput traces for each ABR algorithm and Target Latency setting, when used in the DASH client.

At the lowest Target Latency setting of 2s, that is just one segment duration, all four ABR algorithms performed poorly in terms of P.1203 MOS, with RobustMPC performing the worst, achieving significantly lower MOS than the three other ABRs. MOS equalled to 2.43, 2.56, 1.94, 2.35 for Panda, Festive, RobustMPC, and Bola respectively. In terms of the average Live Latency, all ABRs failed to match the Target Latency setting, with an average value of 4.9s for Panda, Festive, and Bola, and 2.8s for RobustMPC.

At the Target Latency of 4s, the P.1203 MOS for all four ABRs improved considerably, however, it was still on average 0.9 worse than their peak performance at the highest Target Latency setting. At 6s Target Latency, the MOS further improved. For Panda, Festive, and RobustMPC it was on average only 0.1 lower than their peak performance at the highest setting. For Bola, it was 1.23 lower. The average Live Latency for all four ABRs was on average 1.2s and 1s above the Target Latency of 4s and 6s respectively.

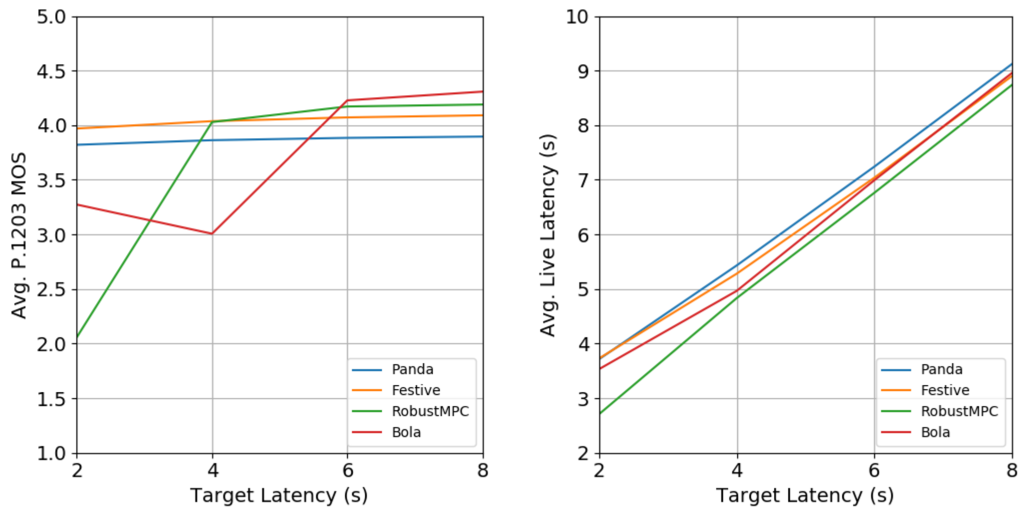

Figure 5.2: Average P.1203 MOS and Live Latency as a function of Target Latency for each ABR algorithm, when used in a CMAF client.

At the highest Target Latency setting, 8s, the peak performance of all four ABRs can be observed. The P.1203 MOS increased to 4.03, 3.97, 4.19, and 4.22 for Panda, Festive, RobustMPC and Bola respectively. However, the average Live Latency for all four ABRs was still above the target setting - by 0.9s on average.

The impact of the Target Latency setting on the performance of all four ABRs can be observed in these results. As the Target Latency increased, the average P.1203 MOS improved by 1.8 on average, however, the improvement was not linear, nor universal. Additionally, the gap between the average Live Latency and the Target Latency decreased, by 1.4s on average. These results indicate that low Target Latency settings have a significant negative impact on the overall Quality of Experience achieved by each tested ABR algorithm.

5.2.2.2 CMAF

Secondly, we present the analysis of the performance of ABR algorithms when used in a CMAF client. The segment duration was configured to 2s, with 0.5s chunks, resulting in four chunks per segment. Figure 5.2 shows the average P.1203 MOS and Live Latency across all 7000 throughput traces for each ABR algorithm and Target Latency setting, when used in a CMAF client.

Just as in DASH, as the Target Latency setting increased, the performance of each ABR algorithm improved. The P.1203 MOS improved from 3.81, 3.96, 2.06, and 3.27, at the lowest Target Latency setting, to 3.89, 4.09, 4.19, and 4.3, at the highest Target Latency setting, for Panda, Festive, RobustMPC, and Bola respectively. Panda and Festive showed minimal improvement as they have both achieved high P.1203 MOS at the lowest Target Latency setting. The range of average MOS for each Target Latency setting decreased as the Target Latency setting increased, from 1.9, at the lowest setting, to 0.41 at the highest setting, showing a high inconsistency in ABR performance at the lowest Target Latency setting.

In terms of the average Live Latency, for all four ABR algorithms, it was on average 1.1s above the Target Latency across all four settings. At the lowest Target Latency of 2s, the Live Latency was 0.9s lower than in the DASH client, on average across all four ABR algorithms. However, for the three other Target Latency settings, the average Live Latency was within ±0.1s of the Live Latency observed in the DASH client, on average across all four ABR algorithms.

While the achieved P.1203 MOS was high for Panda and Festive at the lowest Target Latency, their average Live Latency equalled to 3.7s, which is 1.7s above the Target Latency of 2s. Increased Live Latency can lead to better ABR performance in terms of P.1203 MOS as the client is operating with a larger potential buffer. This is due to the lack of a catch-up mechanism at the client, meaning, any rebuffering will increase the client's Live Latency for the remainder of the streaming session. This indicates that achieving comparable P.1203 MOS at an average Live Latency that is closer to the Target Latency setting would be more difficult for the tested ABR algorithms.

The results presented above indicate that low latency live adaptive media delivery utilising CMAF outperforms regular DASH. Table 5.1 shows the change in average P.1203 MOS for each ABR algorithm and Target Latency setting, when used in a CMAF client instead of DASH. CMAF chunks boosted ABR performance, especially at the lowest Target Latency setting, where the P.1203 MOS achieved by Panda and

Table 5.1: Change in P.1203 MOS for each ABR algorithm and Target Latency setting, when employed in a CMAF instead of a DASH client.

	Target Latency (s)			
	2	**4**	**6**	**8**
Panda	1.38	0.16	-0.1	-0.14
Festive	1.4	0.38	0.19	0.12
RobustMPC	0.12	1.51	0.13	0
Bola	0.92	0.06	1.22	0.08

Festive improved by 1.38 and 1.4 respectively, in comparison to DASH. However, the performance improvement was not universal across all ABR algorithms and Target Latency settings. Additionally, in some cases, the use of CMAF resulted in detrimental ABR performance, such as at the two highest Target Latency settings for Panda. Furthermore, CMAF resulted in reduced average Live Latency at the lowest Target Latency setting, however, it was still above the intended delay.

5.2.3 Summary

Bitrate adaptation in low latency live streaming is challenging due to the restricted maximum client video buffer. We have presented the analysis of the performance of four prominent on-demand ABR algorithms across multiple Target Latency settings, under realistic network conditions, in regular DASH as well as CMAF.

Tested ABR algorithms performed poorly in regular DASH at the two lowest Target Latency settings. However, as the Target Latency setting increased the performance of each ABR algorithm improved. In CMAF, while the ABR performance in terms of P.1203 MOS improved, the average Live Latency was still above the Target Latency setting. These results suggest that the tested ABR algorithms are unsuitable for low latency environments, hence, a new specialised ABR algorithm is required. Additionally, this new ABR algorithm needs to be compatible with CMAF, as the performance improvement across the tested ABRs was not universal.

Table 5.2: Design goals for a new low latency ABR algorithm.

#	**Design Goal**
1	The ABR algorithm must be capable of operating with a small client video buffer.
2	The ABR algorithm must be compatible with CMAF chunks.
3	The ABR algorithm must prioritise avoiding rebuffering.
4	The ABR algorithm must not prioritise the average quality bitrate.
5	The ABR algorithm must prioritise stable quality bitrate.
6	The ABR algorithm must perform multi-level quality bitrate switches of high magnitude gradually.
7	The ABR algorithm should not hesitate to decrease the quality bitrate temporarily if necessary.
8	The ABR algorithm should avoid switching down to the lowest quality bitrate if possible.

5.3 Design Goals

In this section, we discuss the design goals for a low latency ABR algorithm. Table 5.2 presents a constructed list of design goals, based on the results from the previous section as well as the key findings from the subjective study presented in the previous chapter.

The first design goal states that the ABR algorithm must be capable of operating with a small client video buffer. In low latency live adaptive streaming the client video buffer is restricted by target latency, meaning, in order to achieve low latency the target latency must be low, which, results in a small client video buffer. In the previous section, we analysed the performance of four prominent ABR algorithms across different target latency settings. We have found that the performance of all four ABR algorithms was significantly worse at the two lowest latency settings. It is crucial for a low latency ABR algorithm to perform well in environments where the target latency, as well as the resulting client video buffer, is low.

The second design goal dictates that the ABR algorithm must be compatible with CMAF. In the second part of the experiment presented in the previous section, four prominent ABR algorithms were tested when used in a DASH as well as CMAF client, across multiple target latency settings. The results indicate that CMAF can significantly improve ABR performance, especially at the two lowest target latency settings which are crucial to achieving low latency in live adaptive streaming. However, only two out of four tested ABR algorithms experienced a significant performance improvement. In order to benefit from CMAF chunks, a low latency ABR algorithm needs to be compatible with CMAF.

The third design goal states that the ABR algorithm must prioritise avoiding rebuffering. In adaptive streaming, rebuffering occurs when the client video buffer is fully depleted due to segments being fetched for longer than their duration. Current literature suggests that rebuffering is the most detrimental factor to the overall Quality of Experience in adaptive streaming. Additionally, in live adaptive streaming a rebuffering event will increase the client's average latency, especially when no catch-up mechanism is employed at the client as seen in the results presented in the previous section. Considering these two reasons, a low latency ABR algorithm must aim to avoid rebuffering if possible, to provide better QoE and to minimise the average latency.

The fourth design goal specifies that the ABR algorithm must not prioritise the average quality bitrate. This is based on the first key finding from the previous chapter: average video quality bitrate alone is not an accurate predictor of the overall Quality of Experience. Results presented in the previous chapter highlighted multiple cases in which higher average bitrate did not correlate with improvement in QoE. Contrarily, the results highlighted some cases in which a higher average bitrate resulted in worse QoE. Given these observations, it can be concluded that the average quality bitrate is not an accurate predictor of QoE, and therefore the ABR algorithm must not prioritise the average bitrate over other QoE factors.

The fifth design goal dictates that the ABR algorithm must prioritise stable quality bitrate, as per the second key finding: stable video quality can offer better Quality of Experience, even with a lower average video quality bitrate. Results from the previous chapter contained multiple instances in which the stable quality bitrate, a quality bitrate pattern with fewer switches, resulted in the same or better QoE than the unstable quality bitrate of the same or higher average bitrate. Based on this observation, the ABR algorithm must aim to provide a stable quality bitrate if possible, prioritising this QoE factor over some of the other QoE factors, such as the average bitrate.

The sixth design goal states that the ABR algorithm must perform multi-level quality bitrate switches of high magnitude gradually, that is, by changing the quality bitrate by a single encoding bitrate every segment duration. This is based on the third key finding: multi-level video quality switches of high magnitude are better perceived when done gradually, one level at a time, instead of instantly. In the previous chapter, we analysed two sets of quality bitrate patterns of the same average bitrate, where in the first the quality bitrate was switched gradually, and in the second the quality bitrate was switched instantly across the entire encoding bitrate spectrum. We have observed the former to improve the QoE. However, we have also observed no QoE improvement in the case of quality bitrate switches across only two encoding bitrates. Based on these observations, in the case of quality bitrate changes over more than two encoding bitrates, the ABR algorithm must change the quality bitrate gradually.

The seventh design goal specifies that the ABR algorithm should not hesitate to decrease the quality bitrate temporarily if necessary, based on the fourth key finding: temporary decrease in video quality has no impact on the Quality of Experience, except in the case of switching to the lowest quality level. We analysed quality bitrate patterns in which the quality bitrate increases or decreases temporarily, for the duration of only one segment. No QoE impairment in most cases was observed. In real-world scenarios, when the bandwidth deteriorates, it might render the current quality bitrate unsustainable, therefore, the ABR algorithm would need to reduce the quality bitrate before the client video buffer is fully depleted, to avoid a potential rebuffering event. Therefore, the ABR algorithm should reduce the quality bitrate on the first signs of bandwidth deterioration, after which, if the bandwidth instantly improves, the quality bitrate can be corrected by the ABR algorithm without a negative impact on the overall QoE.

The eighth and final design goal, based on the same key finding as the previous goal, dictates that the ABR algorithm should avoid switching down to the lowest quality bitrate if possible, as the lowest quality bitrate has a severe negative impact on the overall QoE. Results presented in the previous chapter indicated multiple instances in which switching to the lowest quality bitrate led to a significantly worse QoE. Therefore, the ABR algorithm should try to avoid switching down to the lowest quality bitrate if possible, however, it might be necessary to avoid rebuffering when the client video buffer is close to being fully depleted and the bandwidth drops below the second lowest encoding bitrate.

5.4 Llama Design

In this section, we detail and discuss the design of Llama, a new ABR algorithm created specifically for low latency live adaptive streaming. This includes the design principles as well as the adaptation logic behind the proposed ABR algorithm. Llama was designed to conform to the low latency ABR algorithm design goals presented in the previous section. Llama utilises multiple throughput measurements in its operation. Each throughput measurement is independent and optimised for a different purpose, one for decisions about reducing the quality bitrate and one for decisions about increasing the quality bitrate. This results in Llama being quick to detect and react to worsening network conditions in order to avoid rebuffering, while being careful about increasing the quality bitrate to provide stable video quality.

Designing an ABR algorithm to be used in low latency environments is a challenge, which, requires the balance between minimising the average latency and maximising the user's Quality of Experience. Achieving low latency in live adaptive streaming limits the client's potential maximum video buffer, as the buffer cannot exceed the target latency due to segments being generated in real time. The lower the client's video buffer the less time the ABR algorithm has to change the quality bitrate to counter the changes in network conditions. Failure to adapt the quality bitrate in time will lead to the client video buffer being fully depleted, causing a rebuffering event. Hence, in order for an ABR algorithm to perform well in low latency scenarios, it needs to be able to detect and react quickly to adverse network conditions, while being careful to

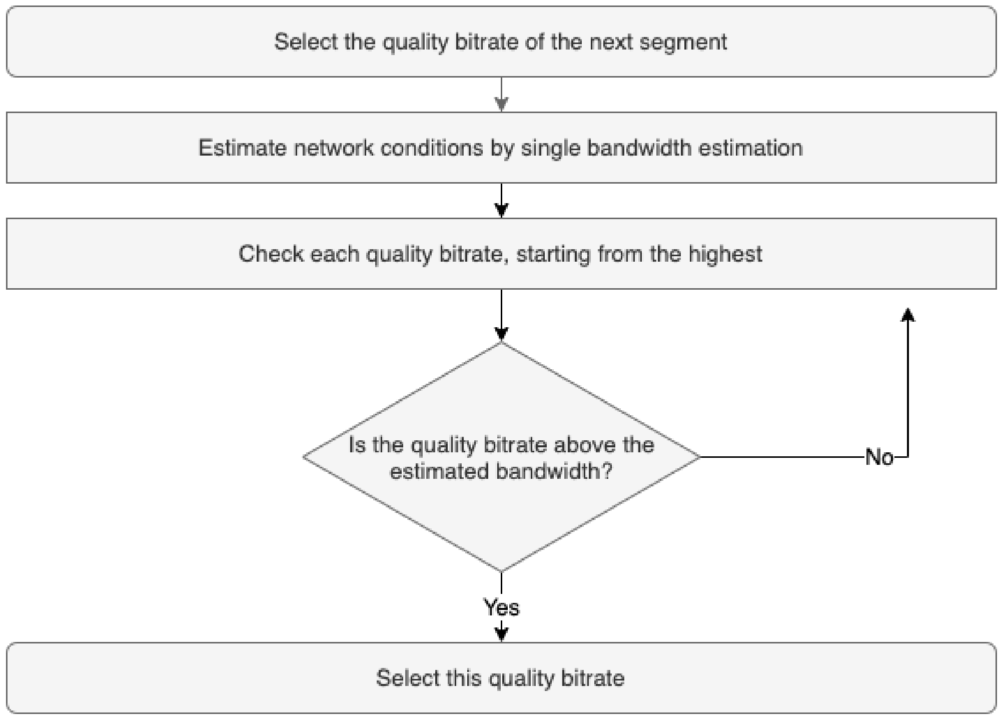
Select the quality bitrate of the next segment
Estimate network conditions by single bandwidth estimation
Check each quality bitrate, starting from the highest
Is the quality bitrate above the estimated bandwidth?
No
Yes
Select this quality bitrate

Figure 5.3: Adaptation logic of a simple bandwidth-based ABR algorithm.

not overestimate the available bandwidth when increasing the quality bitrate as the network conditions appear to improve.

Figure 5.3 demonstrates how a simple bandwidth-based ABR algorithm operates. The video player employs the ABR algorithm to select the quality bitrate of the next segment, after which the segment is fetched by the client. In order to select the quality bitrate, first, the ABR algorithm estimates the network conditions by conducting a single bandwidth estimation. Typically, the bandwidth estimate is an arithmetic mean of the throughput of a number of previous segments. The higher the number of segments the more smoothed the resulting estimate is. The throughput of a segment is calculated by utilising its corresponding HTTP response information, which can be used to deduce the download start time, download end time, as well as the size of the downloaded segment in bytes. Once the bandwidth estimate has been completed, the ABR algorithm compares the available quality bitrates, starting from the highest, against the available bandwidth. The highest sustainable quality bitrate is selected by the ABR algorithm, that is, the highest quality bitrate which is greater than the bandwidth estimate. The ABR algorithm does not employ a mechanism to allow for gradual quality bitrate switches in cases where the quality bitrate is changed by more than one encoding bitrate.

In the previous section, we have established eight design goals for a low latency ABR algorithm. In order to fulfil these design goals, Llama operates using two key design principles which govern its adaptation logic. Each design principle is independent and serves a different purpose, covering the entire eight design goals.

The first design principle focuses on avoiding rebuffering by detecting and reacting to deteriorating network conditions as soon as possible. According to the third design goal, the ABR algorithm must prioritise avoiding rebuffering as it is the most detrimental factor to the overall Quality of Experience. In low latency live adaptive streaming, to avoid rebuffering, the ABR algorithm needs to act on the first signs of deteriorating network conditions and reduce the quality bitrate at which segments are requested before the client video buffer is completely depleted. Hence, to satisfy this design goal a short-term throughput estimation method that tracks the available bandwidth closely is required, and the current quality bitrate needs to be reduced instantly if it is above this measurement. However, the quality bitrate needs to be decreased gradually, meaning, the quality bitrate can be reduced by only one encoding bitrate every segment. This is primarily to satisfy the sixth design goal. Additionally, this ensures that the lowest quality representation is only selected if necessary, and hence attempts to minimise the time spent at the lowest quality bitrate, which we have found in our subjective study to have a significant negative impact on QoE as stated in the eighth design goal. In the event of overreaction, that is when the network conditions worsen for less than one segment duration, the short decrease in quality bitrate will have no significant impact on the QoE as stated in the seventh design goal. Rebuffering events in low latency live adaptive streaming will increase the average latency by the rebuffering duration, especially when no catch-up mechanism is employed at the client. Hence, in order for the ABR algorithm to work well in low latency scenarios, and to satisfy the first design goal, this design principle needs to be of the highest priority.

The second design principle deals with achieving a stable quality bitrate in order to improve the overall Quality of Experience. According to design goals 4 and 5, the ABR algorithm must prioritise stable quality bitrate instead of maximising the average bitrate. Low quality bitrate will significantly reduce the QoE, however, a high variance in quality bitrate can also be detrimental to the overall QoE as seen in the results presented in the previous chapter. These design goals call for a long-term throughput estimation method, which will provide a smoothed bandwidth estimate, based on which, the quality bitrate will be increased only if it exceeds this measurement. It will allow the ABR algorithm to avoid temporary increases in quality bitrate when the available bandwidth improves only for a short period of time, resulting in a stable quality bitrate. Additionally, avoiding switches to the higher quality bitrates unless they are sustainable, in which case they will improve the overall QoE, will reduce the bandwidth consumption, as well as will reduce the chances of rebuffering events due to bandwidth possibly deteriorating shortly after improving. Again, the quality bitrate needs to be increased gradually, that is, by one encoding bitrate every segment, to completely satisfy the sixth design goal. This design principle is supported by the subjective study results, presented in the previous chapter, which showed that: stable quality bitrate is preferred even if it results in lower average bitrate; short temporary increases in quality bitrate do not improve the QoE; and gradual multi-level quality bitrate switches improved the overall QoE.

The flowchart in Figure 5.4 presents the adaption logic behind Llama, demonstrating the utilisation of the two key design principles, as well as, the two independent throughput estimations. In order to find the suitable quality bitrate for the next segment, Llama begins by estimating the network conditions, which, involves the use of two independent throughput estimation methods. The first method focuses on the short-term bandwidth estimate, designed to provide an accurate representation

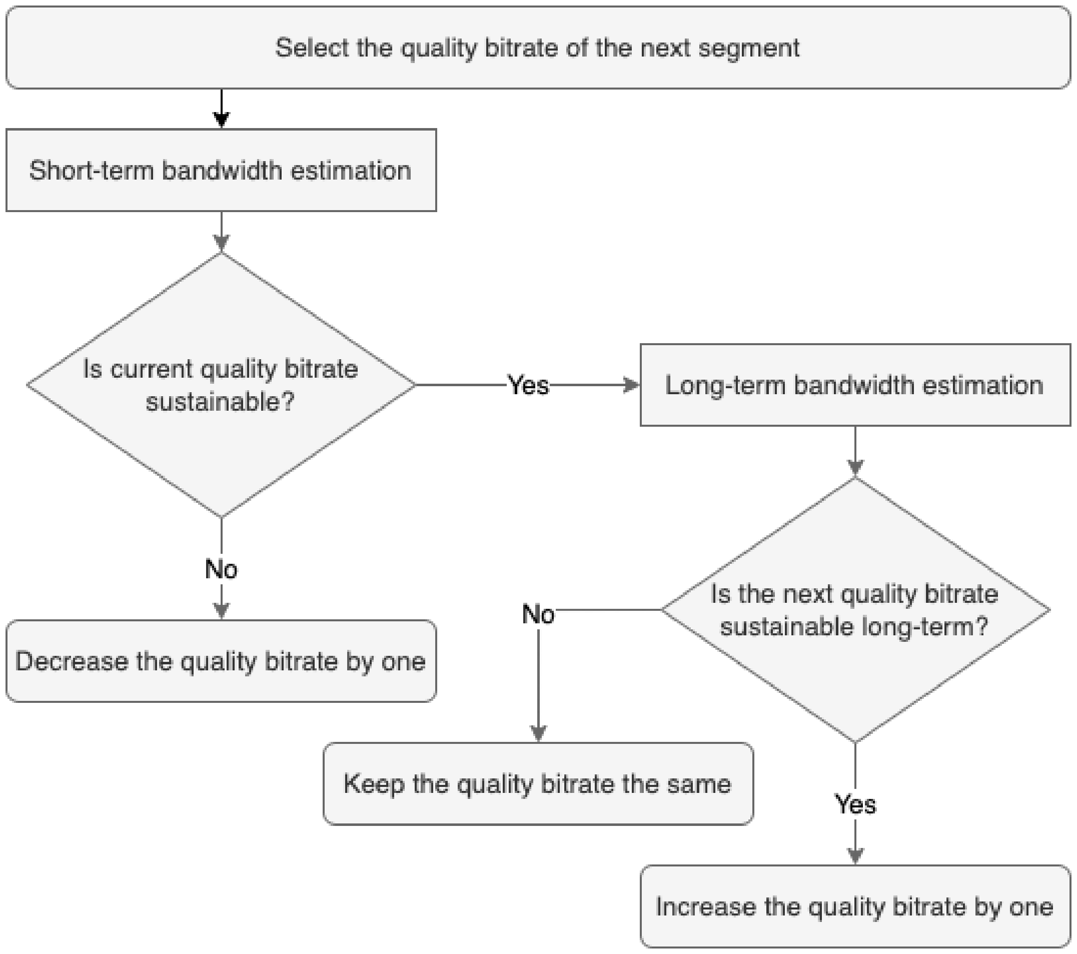
Select the quality bitrate of the next segment
Short-term bandwidth estimation
Is current quality bitrate sustainable?
Yes
Long-term bandwidth estimation
No
Decrease the quality bitrate by one
Is the next quality bitrate sustainable long-term?
No
Keep the quality bitrate the same
Yes
Increase the quality bitrate by one

Figure 5.4: Adaptation logic of the proposed ABR algorithm, Llama.

Table 5.3: Analysis of throughput traces.

	Temp. increases	Temp. decreases	Traces affected
Base measurements	7304	2572	3618
Moving arithmetic mean	108	16	113
Moving harmonic mean	37	6	43

of the current network conditions, while the second method focuses on the long-term bandwidth estimate, designed to give the ABR algorithm an accurate estimate of the bandwidth over a longer period of time. Once both bandwidth estimation methods have been completed, the ABR algorithm proceeds by enforcing the first design principle as it compares the short-term measurement against the current quality bitrate. If the short-term measurement is greater than the current quality bitrate, it indicates the current quality bitrate is unsustainable and can lead to a rebuffering event, hence the ABR algorithm immediately reduces the quality bitrate by one. If the current quality bitrate is sustainable, the ABR algorithm proceeds to enforce the second design principle by establishing if the next quality bitrate is sustainable in the long term. The next quality bitrate, just one encoding bitrate higher, is compared against the long-term bandwidth estimate. If the former is greater than the latter, the quality bitrate is increased by one, as the network conditions have improved sufficiently to allow for higher quality bitrate to become sustainable. However, if the next quality bitrate is smaller than the long-term bandwidth estimate, the quality bitrate remains unchanged.

In order to find suitable throughput estimation methods for the design principles set out above, we have analysed the throughput traces used and described in Chapter 7 along with some throughput estimation methods. Table 5.3 shows some characteristics of the 7,000 throughput traces. The table shows the number of temporary increases in available bandwidth, where the bandwidth increases by more than 500kbps and then decreases by more than 500kbps within 10 seconds, as well as, the number of temporary decreases in bandwidth, where the bandwidth decreases by more than 500kbps and then increases by more than 500kbps in less than 10 seconds. The table also shows the number of throughput traces that experienced at least one temporary decrease or

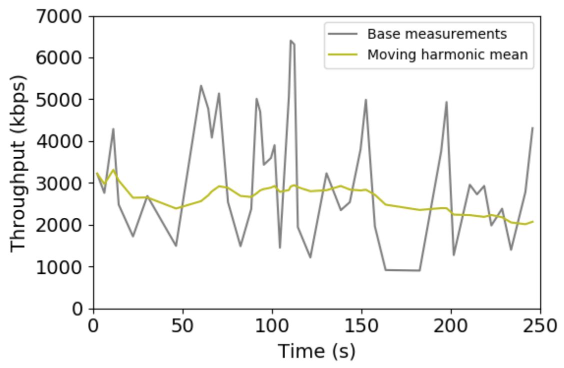

Figure 5.5: Sample trace with base measurements & moving harmonic mean plotted.

increase in bandwidth. The first row demonstrates how unstable the bandwidth is, with over half of the traces experiencing at least one temporary change in bandwidth, and with a total of 7,304 temporary increases and 2,572 temporary decreases in available bandwidth. A smoothing function is often used to reduce the effect of temporary changes in bandwidth. The table shows the impact of smoothing using a moving arithmetic mean and a moving harmonic mean, each using 20 measurements over a period of 40s, showing that the moving harmonic mean is more effective, reducing the number of temporary changes in bandwidth and the number of traces that experience at least one temporary change. Figure 5.5 shows one of the traces with base measurements and moving harmonic mean plotted. It demonstrates how using a smoothing function decreases the number of temporary increases and decreases in estimated bandwidth. At around 160s, the available bandwidth decreases for 20s, however, the moving harmonic mean smooths this out by estimating a higher value for the available bandwidth. In a low latency scenario, this would result in severe rebuffering as the client buffer would be much smaller than the duration of this temporary bandwidth drop.

Algorithm 1 demonstrates which throughput estimation methods are used by Llama to satisfy each design principle. As seen in Table 5.3 and Figure 5.5, using a smoothing function will eliminate some temporary decreases in throughput and hence prevent rapid response to worsening network conditions. Hence, Llama measures the bitrate at which the most recent segment was fetched to decide whether to switch to a lower quality

Algorithm 1 Llama

```
1: $harmonicMeanSize \leftarrow 20$
2: for $segment, n$ do
3:     $lastThroughput \leftarrow segmentThroughput(n-1)$
4:     $harmonicMean \leftarrow calculateHarmonicMean(n-1)$
5:     if $lastThroughput < currentQualityBitrate$ then
6:         Reduce quality representation by one if possible
7:     else if $harmonicMean > nextQualityBitrate$ and $lastThroughput > nextQualityBitrate$ then
8:         Increase quality representation by one if possible
9:     else
10:        Keep the same quality representation
11:    end if
12: end for
13: function SEGMENTTHROUGHPUT($s$)
14:     $throughput \leftarrow segmentSize[s]/downloadTime[s]$
15:     return $throughput$
16: end function
17: function CALCULATEHARMONICMEAN($s$)
18:     $sum \leftarrow 0$
19:     $sampleSize \leftarrow 0$
20:     for $i \leftarrow (s, s - harmonicMeanSize)$ do
21:         if $i > 0$ then
22:             $throughput \leftarrow segmentSize[i]/downloadTime[i]$
23:             $sum \leftarrow sum + (1/throughput)$
24:             $sampleSize \leftarrow sampleSize + 1$
25:         end if
26:     end for
27:     $mean \leftarrow sum/sampleSize$
28:     return $(1/mean)$
29: end function
```

bitrate, switching down to the next lower quality bitrate if the current quality bitrate is higher than the throughput of the last segment. This will allow the ABR to react to worsening network conditions quickly and avoid potential rebuffering events, satisfying the first design principle. However, if the same throughput measurement was used by the ABR to decide whether to switch to a higher quality bitrate, large variations in quality bitrate would occur as the quality bitrate would increase and decrease following temporary changes in available bandwidth. To avoid this issue the ABR employs a second throughput measurement, calculated as the harmonic mean of throughput of the past twenty segments. This smoothed estimate is used by Llama to decide whether to request the next segment at a higher quality bitrate, only choosing to do so when the harmonic mean estimate is higher than the next quality bitrate, resulting in stable quality bitrate and satisfying the second design principle.

The ABR algorithm has one parameter, *harmonicMeanSize*, which specifies the number of segments that are used for the harmonic mean calculation, by default set to twenty segments, for both DASH and CMAF - where chunks are aggregated into segments. When there are fewer than *harmonicMeanSize* segments available, all available segments are used for the calculation. The more segments used, the more smoothing is applied to the throughput measurement, which provides more conservative decisions of when to switch to a higher quality. Fewer segments in this calculation would lead to a higher average quality bitrate at the cost of more variation in quality bitrate. In future work the optimal value for this parameter needs to be determined, however, the current default of twenty segments is sufficient to test the two design principles behind Llama. The same applies to the choice of the long-term bandwidth estimation method as harmonic mean, while being adequate to test Llama's design, might not be the optimal method for detection of long term network conditions.

Llama operates differently from the other three recently proposed low latency ABR algorithms. It uses multiple throughput measurements, each to serve a different purpose, instead of just one as found in Stallion[48]. LoL+[77] calculates future QoE by using a linear equation that takes into account multiple QoE factors, making its performance dependent on the accuracy of the QoE estimation. Llama avoids this issue by focusing on the main two QoE factors, prioritising one over the other, enabling the ABR algorithm to effectively avoid rebuffering while trying to provide high and stable quality bitrate when the network conditions improve. L2A[68] optimises for latency by calculating the response time, future buffer level, and potential rebuffering for each bitrate selection, making its performance dependent on the accuracy of latency prediction. Llama mitigates this issue by prioritising avoiding rebuffering to minimise the latency. All four low latency ABR algorithms depend on accurate throughput estimation provided by the player. We evaluate Llama against the three low latency ABR algorithms in Chapter 7.

5.5 Summary

In this chapter, we have discussed the design behind Llama, a new ABR algorithm created for low latency live adaptive streaming, where the client video buffer is severely limited creating challenging conditions for bitrate adaptation. In the first section, we have set out the aims and objectives of the ABR algorithm, as, to improve the Quality of Experience while trying to minimise the average latency in low latency scenarios. In the second section, we have presented the analysis of the performance of on-demand ABR algorithms in low latency scenarios across multiple target latency settings, justifying the need for a specialised low latency ABR algorithm, as well as, establishing the benchmark ABR performance for evaluation of such algorithms. In the third section, we have outlined design goals for a low latency ABR algorithm.

In the last section, we have discussed the design principles behind the adaptation logic of Llama, a Low Latency Adaptive Media Algorithm. Llama incorporates two independent throughput measurements to achieve its two key design principles, enabling the ABR algorithm to perform well in low latency scenarios, minimising the average latency by detecting and reacting to worsening network conditions instantly, while maximising the QoE by aiming to provide stable video quality.

Chapter 6

Implementation

In this chapter, we discuss the implementation details of the simulation model used in the evaluation of the proposed ABR algorithm, presented in Chapter 7. In the first section, we detail the implementation of the simulation model which allows for faster than real-time evaluation of ABR algorithms in low latency live adaptive streaming. This includes low latency streaming using regular DASH, as well as, CMAF combined with HTTP/1.1 Chunked Transfer Encoding. In the second section, we summarise the implementation of the simulation model.

6.1 Simulation Model

In this section, we discuss the simulation model used in the evaluation experiments, presented in Chapter 7, to assess the performance of multiple ABR algorithms under a large set of network conditions as well as client configurations in low latency live adaptive streaming scenarios. The simulation model was developed in NS-3[98], a discrete-event network simulator, extending an existing implementation of the on-demand DASH model[100]. The extended model is available to the research community on GitHub [99]. In the first subsection, we discuss the existing model and detail how it operates. In the second subsection, we discuss the first extension implemented which allows for the evaluation of ABR algorithms in live adaptive streaming conditions. In

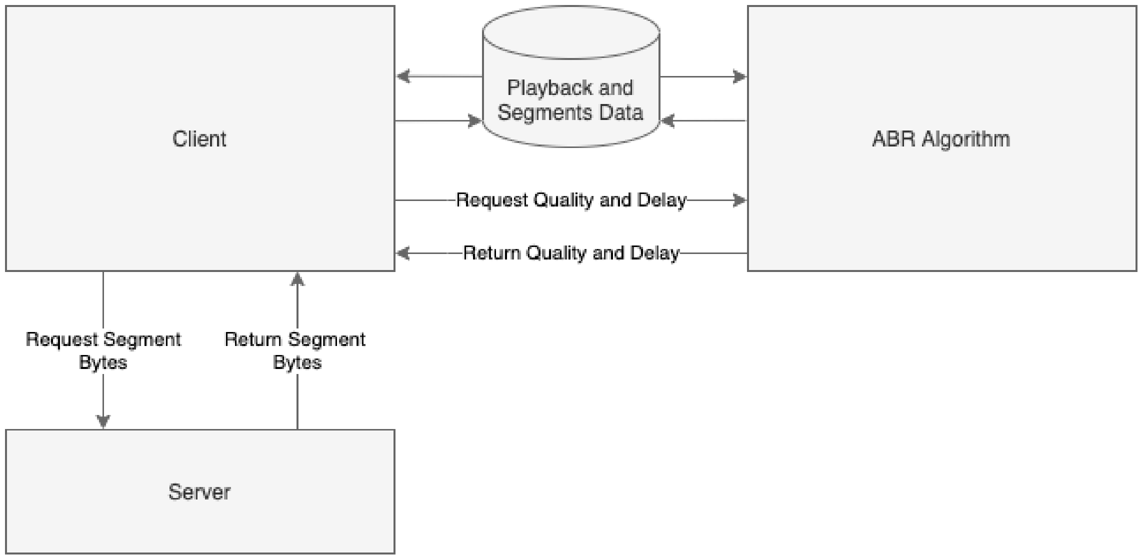

Figure 6.1: Diagram showing the overall structure of the simulation model, where the client consults the ABR algorithm prior to requesting segments from the server.

the third subsection, we discuss the changes made to enable low latency live adaptive streaming via CMAF, where segments are further divided into chunks and transmitted using HTTP/1.1 Chunked Transfer Encoding. In the fourth subsection, we detail the traffic shaping implemented to enable dynamic bandwidth in a streaming session, allowing for testing of ABR algorithms under changing network conditions. In the fifth subsection, we discuss the additional client-based ABR algorithms implemented. In the sixth subsection, we detail the extensive client configuration, including the new parameters added, which allows for the performance evaluation of ABR algorithms under a variety of client configurations. In the final subsection, we verify the accuracy of the simulation model by comparing it against a real DASH player.

6.1.1 Existing Model

The simulation model is based on an existing implementation of an on-demand DASH streaming module for NS-3, written in C++, presented in [100]. In this module the content is transmitted using TCP, between a server and any number of clients, imitating on-demand adaptive streaming where the whole multimedia content is already generated and segmented at the beginning of a streaming session. This module supports external client-based ABR algorithms with three implementations included: Panda[73], Festive[65], and Tobasco2[87]. Figure 6.1 demonstrates an overview of the module. A streaming session begins with the client establishing a connection with the server, after which the client begins to fetch the content, segment by segment. The client employs an ABR algorithm, which can be configured to any of the implementations available, to determine the quality bitrate and the request delay for each segment. Information about each request and segment is logged along with playback metrics over time.

Playback is simulated, with the client employing a video buffer which increases by a single segment each time a new segment is fetched. The client video buffer is depleted by the value of one segment every segment duration. When the client video buffer is depleted, that is when it equals to zero segments after the playback of the last segment has ended, a rebuffering event is triggered which last until another segment is fetched, increasing the client video buffer by one segment again. Each rebuffering event along with its start time, end time, and duration is logged by the simulation model.

The flowchart in Figure 6.2 demonstrates how the segments are fetched by the client. It begins with the first segment. The client first consults the ABR algorithm, which, selects an appropriate quality bitrate for the segment as well as the request delay. The ABR algorithm has access to playback metrics as well as the data of past segment downloads, based on which, it can estimate the current buffer level as well as the throughput at which the previous segments were fetched. Following the response from the ABR algorithm, the client proceeds to wait for the period of the request delay, which, can be utilised by an ABR algorithm to control the size of the client video buffer.

Once the period specified by the request delay has passed, the client requests the segment from the server by issuing a request for a number of bytes corresponding to the file size of the segment at the specified quality bitrate, contained in the segment sizes file. This module does not use actual video segments, instead, it requires the knowledge of the file size of each segment, for each quality bitrate, passed as input.

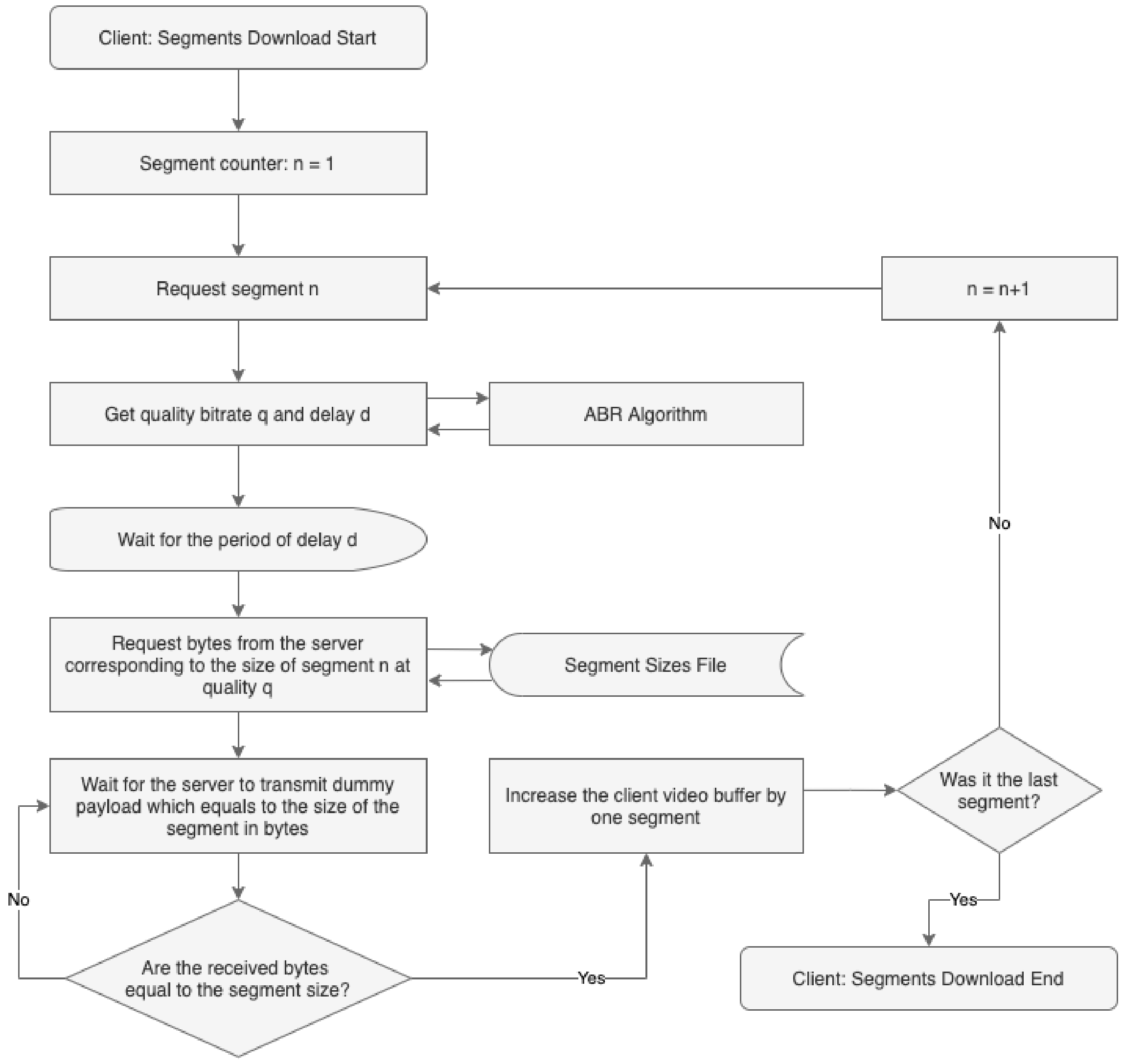
Client: Segments Download Start
Segment counter: n = 1
Request segment n
n = n+1
Get quality bitrate q and delay d
ABR Algorithm
No
Wait for the period of delay d
Request bytes from the server corresponding to the size of segment n at quality q
Segment Sizes File
Wait for the server to transmit dummy payload which equals to the size of the segment in bytes
Increase the client video buffer by one segment
Was it the last segment?
No
Yes
Are the received bytes equal to the segment size?
Yes
Client: Segments Download End

Figure 6.2: On-demand content delivery implementation in the simulation model.

The server responds by transmitting a dummy payload of the size specified in the request. Once the client receives the entire response, the number of bytes corresponding to the file size of the requested segment, the client video buffer is increased by one segment. If this was the last segment specified in the segment sizes file, the client ends the process of downloading all the segments and continues to simulate the playback of the remaining segments in the client video buffer. Otherwise, the client increases its segment counter by one a starts the process again by fetching the next segment while beginning to simulate the playback of segments in parallel.

6.1.2 Live DASH and Latency Configuration

In order to evaluate ABR performance in low latency live adaptive streaming, the existing module required a vast extension. As described in the previous subsection, the existing model only supported on-demand adaptive streaming, where the content is already fully generated at the beginning of the streaming sessions, allowing the client to fetch all of the segments one after another when no request delay was applied by the ABR algorithm.

In live adaptive streaming, this is no longer the case as the content is generated in real-time, meaning, the client can only fetch segments that have already been generated. The number of available already generated segments to the client when joining a streaming session depends on the client's target latency setting, the higher the target latency the further from the live edge the client will be when joining a streaming session. For example, at target latency of two segments, with the segment duration of 2s, the client will be able to join the streaming session only after two segments have already been generated by the server, 4s after the streaming session began. In order to test ABR performance in low latency live adaptive streaming, the simulation model must support live adaptive streaming as well as the client's target latency configuration to ensure low latency conditions.

Figure 6.3 shows how segments are fetched by the client in the extended model, which now supports live adaptive streaming as well as target latency configuration,

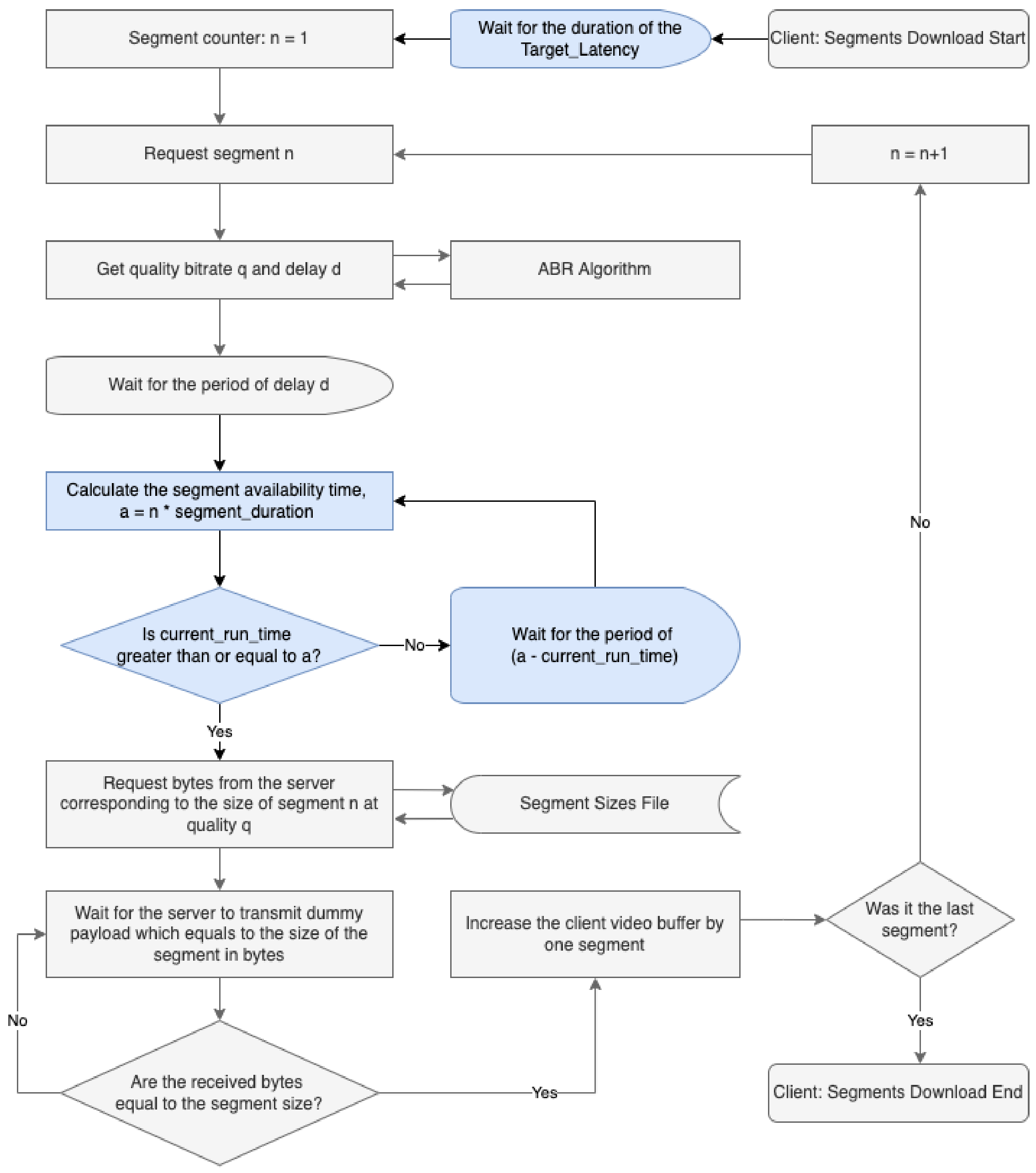
Client: Segments Download Start
Wait for the duration of the Target_Latency
Segment counter: n = 1
Request segment n
n = n+1
Get quality bitrate q and delay d
ABR Algorithm
Wait for the period of delay d
Calculate the segment availability time, a = n * segment_duration
No
Is current_run_time greater than or equal to a?
No
Wait for the period of (a - current_run_time)
Yes
Request bytes from the server corresponding to the size of segment n at quality q
Segment Sizes File
Wait for the server to transmit dummy payload which equals to the size of the segment in bytes
Increase the client video buffer by one segment
Was it the last segment?
No
Yes
Are the received bytes equal to the segment size?
Yes
Client: Segments Download End

Figure 6.3: Live content delivery implementation in the simulation model.

with new elements highlighted in blue. Prior to starting to fetch the content segment by segment, the client waits for the duration of the target latency setting ensuring the client joins the streaming session at the desired time behind the live edge. Target latency is the result of two client parameters, live delay and join offset. Live delay dictates the number of segments behind the live edge, and join offset adds an additional delay before the client joins a streaming session. The target latency equals to the sum of live delay multiplied by segment duration and the join offset.

The second major change involves the timing of segment requests made by the client. In advance of sending a request to the server for each segment, the client calculates the segment's availability time, that is, the time at which segment should become available if the content is generated in real-time, defined as the segment duration multiplied by the segment number, where segments become available periodically every segment duration - imitating live adaptive streaming. Upon calculating the segment's availability time, the client compares it against the current time and only issues a segment request if the latter is greater than or equal to the former. Otherwise, the client waits until the run time reaches the segment's availability time and then issues the segment request. This ensures that the client only requests segments once they have become available on the server as if they were generated in real-time.

6.1.3 CMAF Chunks and HTTP/1.1 CTE

Recently a new method for low latency live adaptive streaming has been proposed. It consists of utilising CMAF chunks transmitted via HTTP/1.1 Chunked Transfer Encoding (CTE). Common Media Application Format allows for segments to be further divided into smaller chunks of equal duration, which can be played out as soon as received by the client. However, to significantly reduce encoding overhead the quality bitrate can be changed at segment level only, just as with regular DASH. When combined with HTTP/1.1 Chunked Transfer Encoding, to reduce the network overhead using partial HTTP responses, the client requests each segment once, after which the first chunk is fetched, and the remaining chunks are transmitted by the server as they

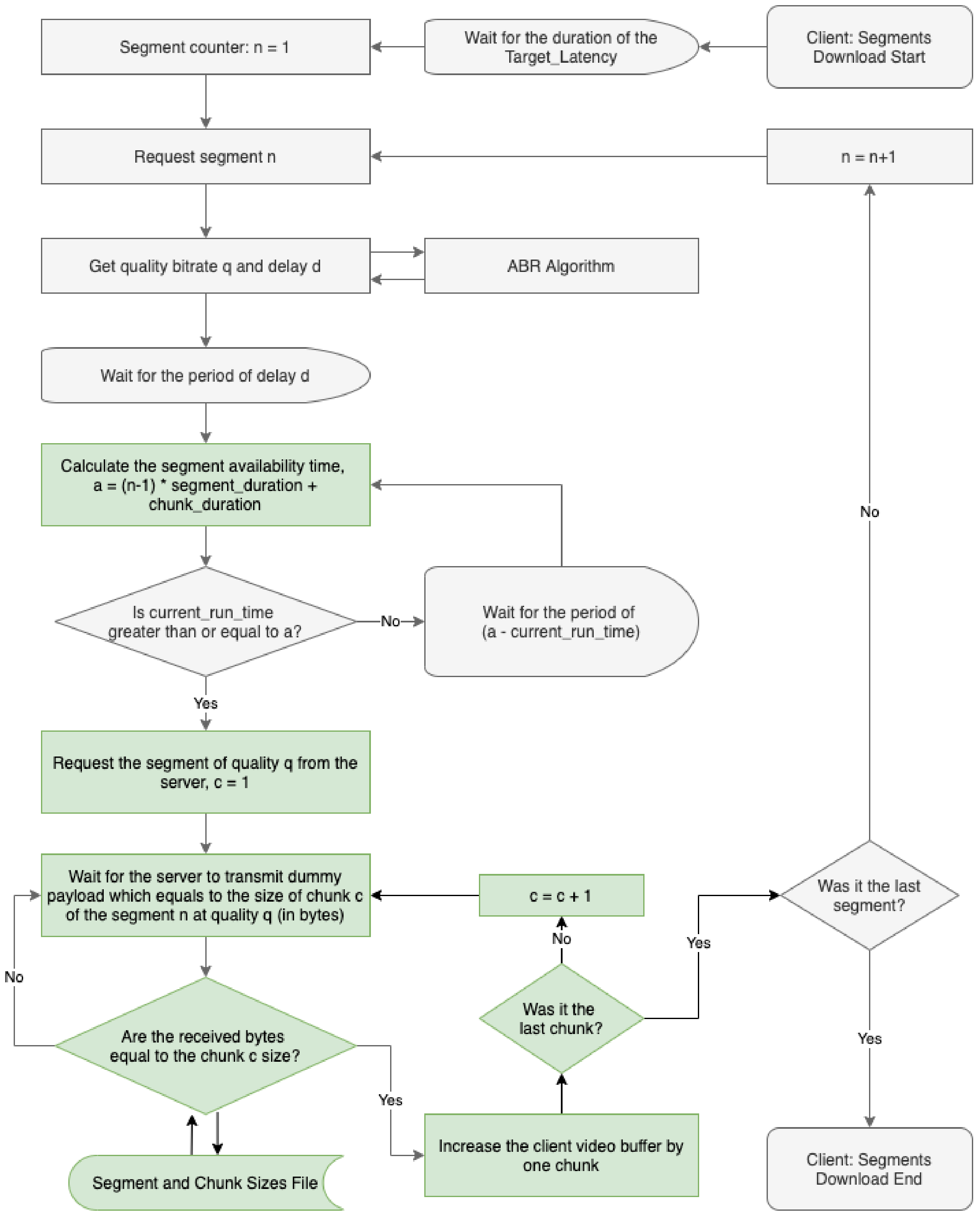
Client: Segments Download Start
Wait for the duration of the Target_Latency
Segment counter: n = 1
Request segment n
n = n+1
Get quality bitrate q and delay d
ABR Algorithm
Wait for the period of delay d
Calculate the segment availability time, a = (n-1) * segment_duration + chunk_duration
Is current_run_time greater than or equal to a?
No
Wait for the period of (a - current_run_time)
Yes
Request the segment of quality q from the server, c = 1
Wait for the server to transmit dummy payload which equals to the size of chunk c of the segment n at quality q (in bytes)
c = c + 1
No
Are the received bytes equal to the chunk c size?
No
Yes
Segment and Chunk Sizes File
Increase the client video buffer by one chunk
Was it the last chunk?
No
Yes
Was it the last segment?
No
Yes
Client: Segments Download End

Figure 6.4: Implementation of CMAF chunks and HTTP/1.1 CTE delivery.

become available, without additional client requests. A segment can be requested as soon as its first chunk becomes available, allowing the client to start fetching each segment earlier than when using regular DASH segments.

The flowchart in Figure 6.4 demonstrates the changes made to the model, highlighted in green, in order to facilitate low latency adaptive streaming using CMAF chunks transmitted using HTTP/1.1 Chunked Transfer Encoding. Since segments are further divided into chunks, when the content is generated in real-time the segments become available as soon as their first chunks are encoded. Hence, the client now calculates the segment availability time as the sum of single chunk duration and the regular segment availability time of the previous segment. Just as in regular live adaptive streaming, the client waits until the run time is greater than or equal to the segment availability time before sending a segment request to the server.

The segment request now consists of the selected quality bitrate instead of the byte size of the segment, as the server has a segment counter which keeps track of the transmitted segments to deliver consecutive segments in the correct order, as well as, access to the segments and chunks sizes file allowing it to deliver chunks of the correct size in bytes. Upon issuing a segment request, the client waits for the response and once the number of bytes corresponding to the size of the first chunk has been transmitted, the client video buffer is increased by one chunk. If there is more than one chunk per segment, the client waits for the remaining chunks to be transmitted in a similar fashion, without additional segment requests.

Once all of the segment's chunks have been fetched, the client moves on to requesting the next segment, if available, by repeating the entire process. Once all segments, and their corresponding chunks, have been fetched, the client ends the fetching process and proceeds to continue with the playback of the remaining chunks in the client video buffer.

6.1.4 Traffic Shaping

With the intention of evaluating the performance of ABR algorithms in low latency live adaptive streaming, the model was extended to support traffic shaping on the link between the clients and the server. This is crucial as it allows for testing of ABR algorithms under different network conditions which can have a significant impact on a streaming session, including the ABR algorithm's behaviour. In order to replicate real-world conditions where the bandwidth is not static, the model needs to support dynamic bandwidth which can change over time.

The model supports any number of clients, which are connected together using the NS-3 CSMA NetDevice class, creating a simple Ethernet-like connection capped at 100mbps. The connection between the clients and the server is created using the Point-To-Point class in NS-3, which allows for the creation of simple point-to-point networks. The bandwidth and the delay of the link are set to 20mbps and 0ms respectively at the beginning of a streaming session. Both parameters can be modified in the client configuration. After the initial link is created with the specified link parameters, the bandwidth of the point-to-point link is changed over time, based on input from a trace file. The trace file contains pairs of timestamps (s) and bandwidth settings (kbps), sorted chronologically, where the timestamp indicates the run-time at which the bandwidth will change to the accompanying bandwidth value.

In the case of multiple clients, the bandwidth of the link between the server and the clients is controlled, allowing for the evaluation of ABR algorithm behaviour in scenarios where the clients need to compete for the bandwidth. In the evaluation experiments presented in Chapter 7, there is always only one client connected to the server. In this case, the bandwidth of the link between the client and the server is controlled, allowing for assessment of ABR algorithm behaviour in different network conditions which can evolve over time.

6.1.5 Additional ABR Algorithms

The existing model allowed for the implementation of additional client-based ABR algorithms. It included three ABR algorithms: Panda[73], Festive[65], and Tobasco2[87]. It was extended to include two more popular on-demand ABR algorithms, RobustMPC[146] and Bola[120], as well as the most recent three low latency ABR algorithms, Stallion[48], L2A[68], and LoL+[77]. The proposed ABR algorithm, Llama, was also implemented.

Panda uses a probe-and-adapt approach similar to TCP's congestion control. It determines a target average data rate, based on which an appropriate quality bitrate is selected. It monitors throughput and adjusts the target average data rate accordingly, as well as calculating the inter-request time for each segment, allowing the buffer level to move towards the configured minimum buffer level.

Festive employs a random scheduler and estimates bandwidth using a harmonic mean which is robust to outliers, this helps improve bandwidth estimation; additionally, the random scheduler requests segments independently of the player's start time, which improves the fairness between players operating in parallel.

RobustMPC solves an optimization problem for a number of segments ahead and uses throughput prediction. It requires both throughput estimation and buffer level to solve this, optimizing towards a set of defined QoE metrics. The implementation is based on the ABR algorithm's implementation found here[1].

Bola employs Lyapunov optimization techniques to minimize rebuffering and maximize video quality. It primarily uses buffer level to determine the appropriate quality bitrate for future segments. The implementation is based on the DASH.JS implementation of Bola[2].

Stallion is a bandwidth-based ABR that measures the moving arithmetic mean and standard deviation of throughput, as well as, latency. The implementation is based on the authors' extension for DASH.JS[3].

[1]https://github.com/hongzimao/pensieve/blob/master/test/mpc.py

[2]https://github.com/Dash-Industry-Forum/dash.js/tree/development/src/streaming/rules/abr

[3]https://github.com/Wimnet/twitch_challenge/blob/master/low-latency/StallionRule.js

L2A (Learn2Adapt) is based on Online Convex Optimisation and aims to minimise the latency. The implementation is based on the implementation included in DASH.JS[2].

LoL+ offers both heuristic and learning-based approaches, optimising for best QoE. In this model, the learning-based variant of LoL+ was included. The implementation is based on the implementation included in DASH.JS[2].

Llama, the proposed low latency ABR algorithm which we describe in more detail in Chapter 5, operates by focusing on two design principles, each accompanied by a different throughput estimation method - optimised to fulfil the specific design principle. The ABR algorithm was implemented according to the pseudo-code provided in Chapter 5.

6.1.6 Client Configuration

In this subsection, we discuss the extensive configuration of streaming sessions available in the simulation model. In order to evaluate the ABR performance in a variety of conditions, the simulation model needs to support configurable streaming sessions. Table 6.1 presents the available parameters for each streaming session.

Each streaming session can be given a unique identifier using the simulationId parameter. Once a simulation run of a streaming session is complete, the performance metrics of the session are saved using the unique identifier. Parameter logLevel dictates how much detail about the streaming session is recorded, by default set to 1 which records detailed information about each segment including quality bitrate, byte size, request time, download start time, download end time, playback time, latency at the start of playback, as well as player metrics including the buffer level throughout the streaming sessions and encountered rebuffering events, and finally each ABR algorithm's decision. When set to 3, aggregated data is recorded, with some of the average metrics calculated.

The streaming session can be configured to standard DASH, with segmentDuration set to the duration of segments, and chunk parameter set to 0. SegmentSizeFile parameter is set to reference a file containing the resulting segment sizes (in bytes) of

Table 6.1: Streaming session and client parameters in the simulation model.

Parameter	**Description**
simulationId	Unique identifier for the streaming sessions, mainly for the logging purpose
numberOfClients	Number of clients in the streaming sessions
segmentDuration	Segment duration or chunk duration when in CMAF mode
adaptationAlgo	ABR algorithm to be employed by each client
segmentSizeFile	Reference to the file which contains the file sizes of all segments for each encoding bitrates, or file sizes of all cmaf chunks in the CMAF mode
linkRate	The initial bandwidth of the link between the server and the clients
delay	The delay of the link between the server and the clients
playbackStart	Playback start delay, how many segments need to be fetched before the playback begins, by default set to 1
trace	Throughput trace compromising of pairs of timestamps and bandwidth settings, to be used as input for the traffic shaping function
chunk	When in CMAF mode, the number of chunks per segment. If 0, the content is transmitted in DASH mode
liveDelay	Number of full DASH/CMAF segments behind the live edge, when the client joins the streaming session
joinOffset	Offset time to DASH/CMAF segment generation (s), after which the client joins the streaming session
logLevel	Logging level: 0: Full, 1: Only playback and stalls, 2: Only QoE metrics: Avg Quality Lvl, Quality S.D., Rebuffer Ratio and Rebuffer Frequency

encoded content at different quality bitrates. The simulation run can also be configured to utilise CMAF chunks and HTTP/1.1 Chunked Transfer Encoding by setting the chunk parameter to the number of chunks per segment. segmentDuration parameter needs to be set to the chunk duration. SegmentSizeFile parameter needs to reference a file containing chunk sizes (in bytes) of the encoded content in multiple quality bitrates.

Target latency of the client in a streaming session can be configured using the liveDelay and joinOffset parameters. liveDelay indicates how far behind the live edge the client will be when it joins the streaming session (in segments) while joinOffset adds an additional delay before the client joins the session (in seconds). The resulting target latency equals to the sum of live delay multiplied by segment duration and the join offset.

Traffic shaping can be applied using the following parameters. LinkRate and delay set the initial bandwidth and delay of the link between the server and the clients. Trace parameter references a throughput trace file compromising of pairs of timestamps and bandwidth settings, to be used as input for the traffic shaping function. The enclosed bandwidth settings within the trace file will be applied at the times specified throughout the streaming session.

6.1.7 Verification

In order to confirm the accuracy of our simulation model, we compared its performance with that of a real-time implementation based on the DASH.JS [25] player which was modified to allow for the testing of different target latency settings. The following parameters have been enabled: Live Delay, which determines how far the player is behind the live edge, and Stable Buffer Time, which is the target buffer for the player. By default, these parameters were overridden when the player is in low latency mode, which is required to deliver CMAF chunks using HTTP/1.1 Chunked Transfer Encoding. The player was employed in the Chromium browser [20].

A simple HTTP server was created in Node.js, to serve pre-encoded DASH segments and CMAF chunks using HTTP/1.1 Chunked Transfer Encoding. Segments and chunks

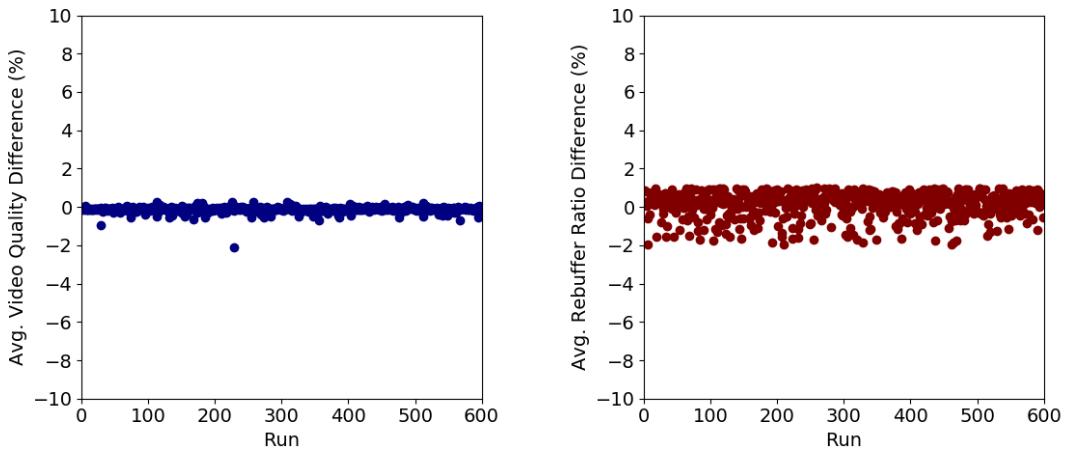

Figure 6.5: Difference in the performance of the NS-3 and Dash.JS frameworks, measured using the QoE metrics, Video Quality (left), and Rebuffer Ratio (right) across 600 runs consisting of 100 throughput traces and 6 latency settings.

are served when they become available, as if the content were being encoded in real-time. The server, which is available on GitHub [23], was placed inside a virtual mininet [88] network, with the client located outside on a separate local network. These networks were connected by a link with bandwidth controlled using the linux tc module [125].

We compared the performance of the simulation model with that of the DASH.JS player using 100 out of 7000 throughput traces presented in Chapter 7, with Live Delay set to 1 to 6 segments, and with the player configured to low latency mode, enabling the CMAF mode. In total, this resulted in 600 real-time runs with the four-minute clip giving a total run time of around 42 hours.

Figure 6.5 shows the average percentage difference in the performance of the modified simulation model and DASH.JS player, when measured using the two QoE metrics: Video Quality, and Rebuffer Ratio. The left plot shows that the average Video Quality achieved by the simulation model and by DASH.JS on each of the 600 runs differ by less than +/-2%, with most differences being within +/-1%. The right plot shows that the average Rebuffer Ratio values achieved by the simulation model and by

DASH.JS are also within +/-2% of each other, and most differences are between 0% and 1%.

Traffic shaping in real networks is difficult, leading to less faithful reproductions of the network conditions contained in the throughput traces than in a simulation model, resulting in the differences seen in the metrics above. Considering this, these results indicate that the simulation model is an accurate reflection of a real-world system.

6.2 Summary

In this chapter, we have discussed the implementation of the simulation model used in the evaluation presented in Chapter 7, including the proposed ABR algorithm, Llama. The simulation model offers faster than real-time evaluation of highly configurable streaming sessions in low latency live adaptive streaming. It supports regular DASH as well as CMAF chunks combined with HTTP/1.1 Chunked Transfer Encoding. Each streaming session can be configured to satisfy a variety of client configurations, including the target latency of the client which can have a significant impact on the performance of ABR algorithms. Traffic shaping between the clients and the server is supported by the simulation model, enabling the assessment of ABR algorithms in realistic network conditions based on real throughput traces. The simulation model was verified against the DASH.JS player, a real DASH implementation, and achieved high accuracy rendering it a reliable tool for evaluation of ABR algorithms in low latency live adaptive streaming.

Chapter 7

Evaluation

In this chapter we present the evaluation of Llama, a new ABR algorithm designed specifically for low latency live adaptive streaming. In the first section, we discuss the aims and objectives of this evaluation. In the second section, we establish the common methodology behind each evaluation experiment presented in this chapter. In the third section, we evaluate Llama against the four on-demand ABR algorithms presented in Chapter 5. In the fourth section, we evaluate Llama against the three most recent low latency ABR algorithms, published the same year as Llama. In the final section, we conclude by summarising this chapter.

7.1 Aims and Objectives

The main objective of this chapter is to evaluate Llama's suitability for low latency live adaptive streaming. In order to determine this, the performance of the proposed ABR algorithm needs to be compared against the benchmark performance established in the previous chapter, where the performance of four prominent on-demand ABR algorithms has been analysed, using two methods of low latency delivery, regular DASH as well as CMAF. These results demonstrated how CMAF, where segments are further divided into chunks, can significantly improve the ABR algorithm's performance at low target latency settings, however, the improvement was not universal across all four

ABR algorithms tested. Therefore, the proposed ABR algorithm needs to be evaluated using both low latency delivery methods, DASH and CMAF.

In Chapter 5 performance of four prominent on-demand ABR algorithms was analysed. These ABR algorithms have not been designed for live or low latency adaptive streaming and hence might perform worse than the ABR algorithms specialised to operate in low latency conditions. Recently, three new low latency ABR algorithms have been published – proposed the same year as Llama. Hence, to establish the proposed ABR algorithm's suitability for low latency, its performance needs to be compared against the performance of these three new low latency adaption solutions. In total, the performance of the proposed ABR algorithm will be assessed against seven different ABR algorithms in this chapter.

Low latency live adaptive streaming requires the ABR algorithm to not only try and maximise the Quality of Experience, but also to try and minimise the average latency of a streaming session – all this while operating with a small client video buffer due to the low target latency. Therefore, to judge the ABR algorithm's performance and suitability for low latency scenarios, both the Quality of Experience as well as the average latency needs to be measured and studied. Additionally, the ABR performance is significantly impacted by the target latency configuration of the client, as demonstrated in Chapter 5, requiring the evaluation to include a broad range of combinations of the client settings and parameters.

7.2 Methodology

In this section, we discuss the methodology behind the evaluation of the proposed low latency ABR algorithm. In order to test the proposed ABR under a large number of configurations, faster than real-time evaluation was required, hence, a simulation model was developed which we describe in the first subsection. Next, we detail the test content used as well as the encoding process and parameters. Later, we discuss the throughput traces used in our experiments to create realistic network conditions. Lastly, we outline the metrics used to assess the performance of each ABR algorithm.

7.2.1 Simulation Model

In this subsection, we briefly discuss the simulation model, which we have presented in full detail in Chapter 6. The simulation model drastically reduces the run-time of an individual streaming session, from 4 minutes to 2-3 seconds for the content of 4-minute duration, allowing for an extensive ABR algorithm evaluation in a reasonable time. The simulation model was developed in a discrete-event network simulator NS-3[98]. It supports live DASH, CMAF chunks, latency configuration, a variety of ABR algorithms, and accurate traffic shaping between the client and the server. It has the following parameters.

The Chunk (chunk) parameter sets the client into DASH or CMAF mode. When in DASH mode, the client requests the content segment by segment, requesting each segment only once the entire segment has become available on the server, simulating regular live DASH. In CMAF mode, the client requests each segment as soon as its first chunk becomes available on the server and fetches the first chunk, after which the remaining chunks belonging to the segment are transmitted by the server when they become available, without additional client request, simulating the delivery of CMAF chunks using HTTP/1.1 Chunked Transfer Encoding.

The ABR algorithm (adaptationAlgo) parameter determines the adaptation logic to be employed by the client, selecting one of the following ABR algorithms: Llama, Panda[73], Festive[65], RobustMPC[146], Bola[120], Stallion[48], L2A[68], and LoL+[77]. Llama operates as described in Chapter 5, while the remaining ABR algorithms are tuned according to the parameters presented in their respective papers. Prior to fetching each segment, the client consults the ABR algorithm requesting the appropriate quality bitrate for the segment to be selected. ABR algorithm's decisions take the highest priority as the client cannot override the selected quality bitrate.

The Throughput Trace (trace) parameter specifies what traffic shaping should be applied during the streaming sessions. It references a file, which contains timestamps and bandwidth values in kbps, which are used to adapt the bandwidth of the link between the client and server over time.

The initial bandwidth and delay of the link between the client and server were specified by the Link Rate (linkRate) and Delay (delay) parameters. In our evaluations, the link rate parameter was set to 20mbps, however, it was always overridden instantly as each throughput trace started at 0s. The delay was set to 0ms and always remained constant as the employed throughput traces did not include delay information.

The Live Delay (liveDelay) parameter specifies which segment the client requests first when joining a session, effectively, determining how many segments behind the live edge the client will be at the beginning of the streaming session. For example, when set to one, the client will request the most recent segment available on the server, and when set to two, the client will request the second most recent segment available.

The Join Offset (joinOffset) parameter determines an additional delay experienced by the client before the first segment, specified by the live delay parameter, is fetched. When set to 0s, the client will request the first segment as soon as the segment becomes available on the server. When the segment duration is 2s and the Join Offset is set to 1s, the client requests a first segment mid-way between consecutive segments being made available on the server.

The Live Delay and Join Offset parameters combined indicate the target latency, where Live Delay affects the potential maximum buffer level of the client, as does Join Offset, but only with CMAF and a chunk duration less than Join Offset. The latency is defined as the time between a segment being generated at the server and the segment being presented at the client.

7.2.2 Test Content and Encoding

One piece of content was selected which was used for every single evaluation run to ensure a fair comparison between achieved QoE metrics in different conditions. The first four minutes of the BigBuckBunny movie [11] were used as the test content. It was encoded using x264 [138] at bitrates and resolutions shown in Table 7.1. The encoded test content was then segmented using MP4Box [94] into 2s segments for DASH mode, and into 2s segments with 0.5s chunks for CMAF mode. The file sizes of the resulting

Table 7.1: Video quality representations used for the encoding of content.

Quality Index	**Resolution**	**Avg. Bitrate (Kbps)**
0	426x240	400
1	640x360	800
2	854x480	1200
3	1280x720	2400
4	1920x1080	4800

segments/chunks were used in the simulation model as actual segments/chunks are not required.

7.2.3 Throughput Traces

ABR performance evaluation requires testing under a variety of dynamic network conditions as the changes in bandwidth will require the ABR algorithm to make decisions regarding the appropriate quality bitrate. To simulate realistic network conditions, throughput traces derived from CDN logs of a commercial live TV service, BT Sport 1, were used. Each throughput trace contains pairs of timestamps and values (kbps) sorted in chronological order. The bandwidth is changed at the time belonging to the pair, setting the bandwidth on the link between the client and server until the next bandwidth pair in the throughput trace.

The CDN logs, which were provided by BT, contained the request time, segment size and download time of each segment for every streaming session over a whole day. From this information, the bandwidth and the time can be deduced, with the bandwidth calculated at the segment size divided by segment download time and the time specified as the segment request time. Each throughput trace corresponds to one streaming session from the CDN logs.

Throughput traces shorter than the duration of the test content (four minutes) were disregarded, and longer traces were trimmed to match the duration of the test

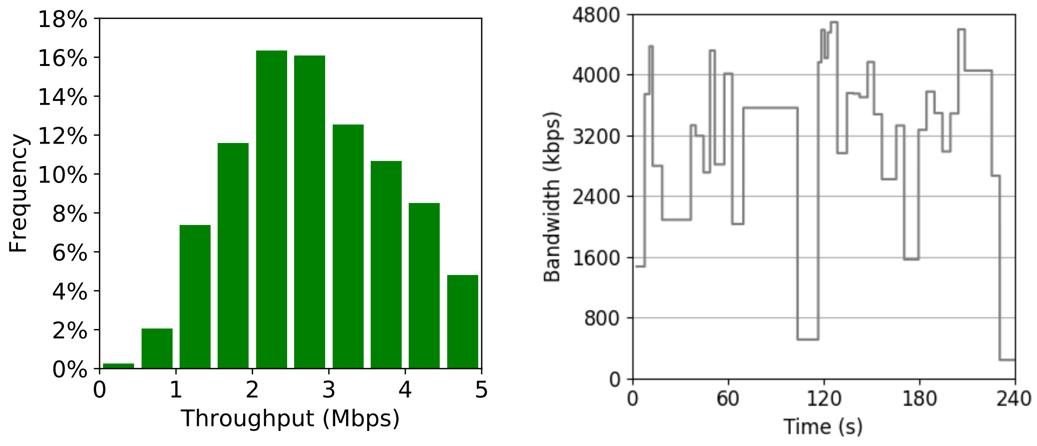

Figure 7.1: Distribution of bandwidth measurements below 5 Mbps in the 7,000 throughput traces, along with an example showing one of the traces.

content. Throughput traces which contained bandwidth measurements resulting in the arithmetic mean greater than the highest quality bitrate in our experiments (4800kbps), as well as traces where all bandwidth measurements corresponded to only one quality bitrate were disregarded, as in both cases the ABR algorithm would not need to make any decisions - rendering these traces unsuitable for evaluation of ABR performance.

This resulted in 7,000 throughput traces to use in the evaluation experiments. The distribution of bandwidth measurements in these throughput traces is shown as a histogram in Figure 7.1, which shows the relative frequencies of bandwidth measurements below 5 Mbps, along with a plot of one of the traces. Approximately 89% of the bandwidth measurements are below the highest quality bitrate of 4800 kbps. All 7,000 throughput traces were published on GitHub[128].

7.2.4 Performance Metrics

With the intention of assessing ABR algorithm performance in low latency live adaptive streaming, the following performance metrics were recorded. Average Live Latency which is calculated as the arithmetic mean of the latency recorded every segment, and indicates the latency of the streaming session. In low latency live streaming, the

ABR algorithm must aim to minimise the latency to achieve the desired target latency. Since there is no catch-up mechanism employed at the client in the simulation model, during a streaming session the latency will increase if a rebuffering event occurs, by the rebuffering duration, and never decrease.

Another important metric to study when trying to assess the performance of an ABR algorithm is the overall Quality of Experience, which the ABR algorithm must aim to maximise. In order to measure the overall Quality of Experience, the predicted mean opinion score was calculated using the ITU-T Rec. P.1203 model which combines the bitrate, resolution, framerate at which each segment is delivered, as well as the frequency and duration of all rebuffering events into a single value between 1 and 5 [105, 108]. A higher value indicates better overall Quality of Experience. The standalone implementation of the model was used, which can be found here [63].

Additionally, some of the contributing QoE factors were also recorded. Quality Variability, a standard deviation of the quality bitrates of the requested segments, indicates how stable the quality bitrate is. Rebuffer Ratio, calculated as the ratio of the total rebuffering time to playback time, reveals the total playback interruption time. Video Quality, which is calculated as the arithmetic mean of quality bitrate selected by the ABR algorithm. Finally, the number of sessions which experienced rebuffering was also recorded and presented as a percentage of the entire set of sessions.

7.3 Comparison with prominent on-demand ABRs

In this section, we present the results of the evaluation which compares the performance of the proposed ABR algorithm against four prominent on-demand ABR algorithms: Panda, Festive, RobustMPC and Bola. Each of these ABR algorithms was not designed for live or low latency adaptive streaming. The results are presented in the first subsection using the performance metrics described in the previous section. We discuss and summarise the results in the final subsection.

7.3.1 Results

In this subsection, we present the results of our evaluation. We compare the performance of Llama against Panda, Festive, RobustMPC and Bola firstly when the client is in DASH mode, and then again when the client is in CMAF mode.

We have used the following parameters for both evaluations. The Live Delay was set to values ranging from 1 to 3 segments with the Join Offset set to 0s, 0.5s, 1s, and 1.5s for each Live Delay setting, resulting in 12 Target Latency configurations, from 2s to 7.5s with increments of 0.5s. All 7,000 throughput traces were used for each combination of Live Delay and Join Offset resulting in 84,000 runs per ABR algorithm for one mode of client. In total, the content was delivered to the client 840,000 times, including for all five ABR algorithms and both DASH and CMAF clients.

The performance of all ABR algorithms is reported using the performance metrics described in the previous section, averaged over the 7,000 runs with different bandwidth profiles. We combined Live Delay and Join Offset into a single value of Target Latency, which indicates the effective latency in seconds, calculated as Live Delay multiplied by the segment duration of two seconds and added to the Join Offset value.

7.3.1.1 DASH

For this evaluation, the client was configured in DASH mode and used with segments of duration of two seconds. Figure 7.2 shows the performance of all five ABR algorithms in terms of average Video Quality, Quality Variability, Rebuffer Ratio, P.1203 MOS, and Live Latency, as well as, the percentage of sessions experiencing rebuffering for each Target Latency setting.

Performance in terms of rebuffering. Llama achieved the second lowest average Rebuffer Ratio out of all five ABRs between Target Latency settings of 2s and 4s, with RobustMPC achieving the lowest. However, RobustMPC also achieved the lowest average Video Quality for these settings, which is discussed later. Above Target Latency of 4s, Llama outperformed all other ABRs. At Target Latency of 2s, the average Rebuffer Ratio achieved by Panda, Festive, RobustMPC, Bola, and Llama was 1.49%,

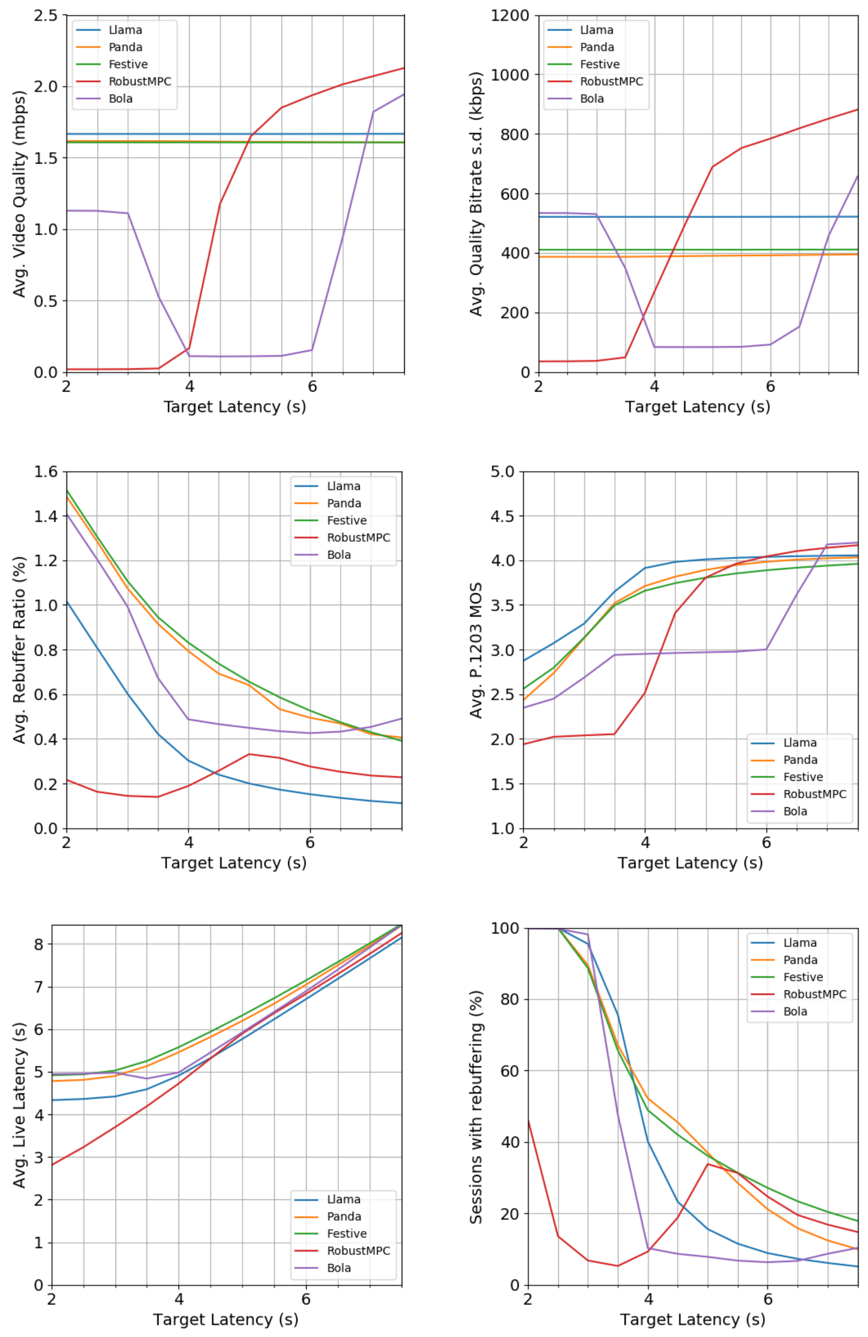

Figure 7.2: Performance of four on-demand ABRs and Llama, when used in a DASH client, in terms of average Video Quality, Quality Variability, Rebuffer Ratio, P.1203 MOS, and Live Latency, as well as, the percentage of sessions experiencing rebuffering.

1.51%, 0.22%, 1.41%, and 1.02% respectively. At Target Latency of 4s, the average Rebuffer Ratio decreased to 0.79%, 0.83%, 0.19%, 0.49%, and 0.3% for Panda, Festive, RobustMPC, Bola and Llama respectively. At Target Latency of 6s, it decreased further for Panda, Festive, Bola and Llama to 0.49%, 0.53%, 0.43%, and 0.15% respectively. However, in the case of RobustMPC it increased to 0.28%.

Most ABRs had a high percentage of sessions with rebuffering events at the lowest value of Target Latency, and a decreasing percentage as the value of Target Latency increased. At Target Latency of 2s, the percentage of sessions with rebuffering events for Panda, Festive, RobustMPC, Bola, and Llama equalled to 100%, 100%, 40.16%, 99.74%, and 100% respectively. At Target Latency of 4s, it decreased to 52.19%, 48.89%, 9.3%, 10.26%, and 40.13% for Panda, Festive, RobustMPC, Bola, and Llama respectively. At Target Latency of 6s, it decreased further for Panda, Festive, Bola and Llama to 21.16%, 27.14%, 6.33%, and 8.89% respectively, while for RobustMPC the percentage of sessions with rebuffering events increased to 24.7%.

As Target Latency increased, the average Rebuffer Ratio and the percentage of sessions with rebuffering events for Llama always decreased, indicating that Llama can increase its performance by not only taking advantage of higher Live Delay settings, but also higher values of Join Offset.

Performance in terms of video quality. Llama achieved the highest average Video Quality for Target Latency settings between 2s and 5s, and the second highest for Target Latency between 5.5s and 6.5s, where RobustMPC achieved the highest. Above Target Latency of 6.5s, Llama achieved lower average Video Quality than RobustMPC and Bola, the former one achieving the highest. At Target Latency of 2s, the average Video Quality achieved by Panda, Festive, RobustMPC, Bola, and Llama was equal to 1.62, 1.61, 0.02, 1.13, and 1.67 respectively. At higher values of Target Latency, Festive and Llama maintained the same average Video Quality, and Panda slightly decreased its average Video Quality to 1.61 at the highest Target Latency. RobustMPC's average Video Quality stayed low for Target Latency settings up to 4s where it was equal to 0.17, and started to increase significantly at Target Latency of 4.5s where it increased to 1.18. RobustMPC achieved the highest average Video Quality for Target Latency of

5.5s and higher, peaking at 2.13 when at the highest Target Latency. Bola's average Video Quality started to decrease at Target Latency of 3s and stayed low at less than 0.16 for Target Latency settings between 4s and 6s. At Target Latency of 6.5s it started to increase again, peaking at 1.94 when at the highest Target Latency setting.

Llama achieved near constant average Quality Variability across all values of Target Latency, with differences within 0.51 and peaking at 521.7. Panda and Festive also achieved constant average Quality Variability, with differences within 7.88 and 0.65 along with peaks at 394.82 and 411.28 respectively. RobustMPC's average Quality Variability was below 49.02 up to the Target Latency of 3.5s, and significantly increased as Target Latency increased beyond 3.5s with a peak value of 881.38 at the highest Target Latency. Bola had average Quality Variability of 534.08-530.62 for Target Latency of 2-3s, which decreased between Target Latency of 3s and 4s to 83.9 and stayed within 0.6 up to Target Latency of 6s. Its average Quality Variability increased beyond Target Latency of 6s and peaked at 656.61 when at the highest Target Latency setting.

Llama achieved the highest average Video Quality for values of Target Latency up to 5s, and remained consistently high for higher values of Target Latency. It also achieved near constant Quality Variability for all values of Target Latency resulting in a much more stable quality bitrate than RobustMPC at Target Latency of 5s and higher.

Overall performance. For Target Latency between 2s and 5.5s, Llama outperformed all other ABRs in terms of P.1203 MOS. At Target Latency of 6s, it was still the highest, along with RobustMPC which achieved the same P.1203 MOS. Beyond Target Latency of 6s, Llama was outperformed by RobustMPC and Bola which achieved slightly higher P.1203 MOS. At Target Latency of 2s, Panda, Festive, RobustMPC, Bola and Llama had P.1203 MOS of 2.43, 2.56, 1.94, 2.35, and 2.87 respectively. At Target Latency of 4s, P.1203 MOS increased to 3.71, 3.66, 2.51, 2.95, and 3.91 for Panda, Festive, RobustMPC, Bola, and Llama respectively. At Target Latency of 6s it further increased to 3.98, 3.89, 4.04, 3.0, and 4.04 with peaks of 4.03, 3.96, 4.17, 4.2, and 4.05 at the highest Target Latency for Panda, Festive, RobustMPC, Bola, and Llama respectively.

Llama achieved the second lowest average Live Latency for Target Latency settings up to 4.5s, and the lowest for Target Latency of 5-7.5s. RobustMPC achieved the lowest average Live Latency for Target Latency settings between 2s and 4.5s, on average 0.7s lower than Llama, however, at the cost of significantly worse P.1203 MOS, on average 1.1 worse when compared to Llama. For Target Latency settings between 4.5-6s Llama achieved the highest P.1203 MOS as well as the lowest average live latency when compared to the remaining four ABR algorithms. At Target Latency of 6.5-7.5s, Llama achieved the lowest average Live Latency, at the cost of on average 0.09 and 0.14 worse P.1203 MOS when compared to RobustMPC and Bola respectively, an inconsequential deterioration in Quality of Experience. Overall, out of all five ABR algorithms, Llama achieved the best balance between minimising the average Live Latency and maximising the P.1203 MOS.

7.3.1.2 CMAF

For this evaluation, the client was configured in CMAF mode and used with segments of duration of two seconds and chunk duration of 0.5 seconds. Figure 7.3 shows the performance of all five ABRs in terms of average Video Quality, Quality Variability, Rebuffer Ratio, P.1203 MOS, and Live Latency, as well as, the percentage of sessions experiencing rebuffering for each value of Target Latency.

Performance in terms of rebuffering. For Target Latency between 2s and 3s, Llama achieved the second lowest average Rebuffer Ratio and RobustMPC achieved the lowest. For higher values of Target Latency, Llama achieved the lowest average Rebuffer Ratio. At Target Latency of 2s, the average Rebuffer Ratio for Panda, Festive, RobustMPC, Bola, and Llama equalled to 1.01%, 0.89%, 0.11%, 0.77%, and 0.32% respectively. At Target Latency of 4s it changed to 0.79%, 0.58%, 0.23%, 0.41%, and 0.17% for Panda, Festive, RobustMPC, Bola, and Llama respectively. At Target Latency of 6s it changed further to 0.61%, 0.39%, 0.17%, 0.43%, and 0.11% for Panda, Festive, RobustMPC, Bola, and Llama respectively.

Llama had fewer sessions with rebuffering events than Panda and Festive at all Target Latency settings, and the lowest percentage of sessions with rebuffering events at Target Latency of 4.5-7.5s. At Target Latency of 2s, the percentage of sessions

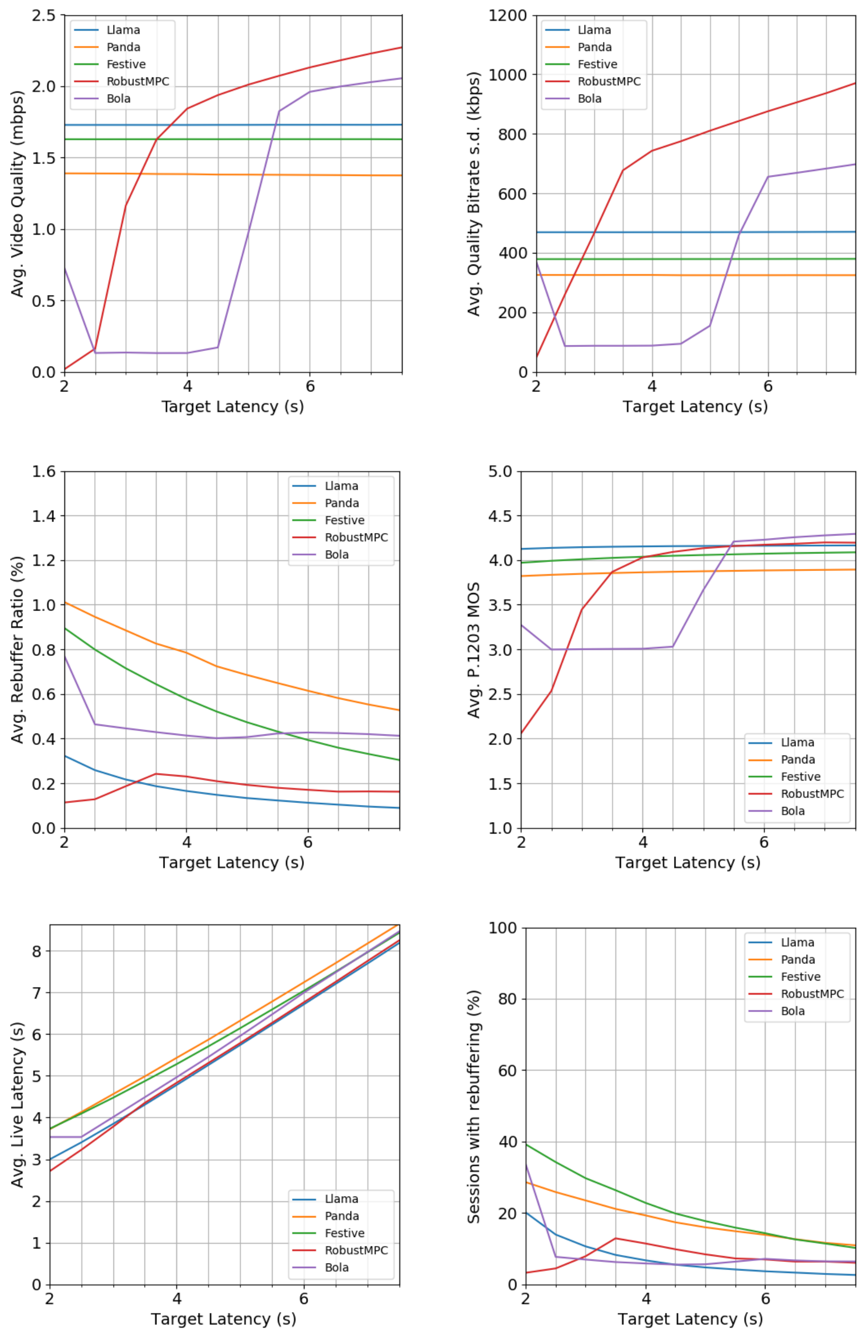

Figure 7.3: Performance of four on-demand ABRs and Llama, when used in a CMAF client, in terms of average Video Quality, Quality Variability, Rebuffer Ratio, P.1203 MOS, and Live Latency, as well as, the percentage of sessions experiencing rebuffering.

with rebuffering events for Panda, Festive, RobustMPC, Bola, and Llama was 28.61%, 39.2%, 3.24%, 33.64%, and 20.13% respectively. At Target Latency of 4s, it increased to 11.44% for RobustMPC, and decreased to 19.31%, 22.83%, 5.86%, and 6.73% for Panda, Festive, Bola, and Llama respectively. At Target Latency of 6s, it increased to 7.14% for Bola, and decreased to 13.87%, 14.31%, 6.97%, and 3.66% for Panda, Festive, RobustMPC, and Llama respectively.

With Llama, both the average Rebuffer Ratio and the percentage of sessions with rebuffering decreased as the Target Latency increased, indicating that Llama can take advantage of higher Live Delay and Join Offset values to increase its performance with CMAF, just as it did with DASH.

Performance in terms of video quality. Llama achieved the highest average Video Quality at Target Latency between 2s and 3.5s, and remained consistently high for higher values of Target Latency. It outperformed Panda and Festive at all values of Target Latency, but was outperformed by RobustMPC for Target Latency of 4s-7.5s, as well as by Bola for Target Latency of 5.5-7.5s. At Target Latency of 2s, the average Video Quality was equal to 1.39, 1.63, 0.02, 0.73, and 1.73 for Panda, Festive, RobustMPC, Bola, and Llama respectively. It remained constant for Festive and Llama across all Target Latency settings and near constant for Panda as it decreased by only 0.01 at the highest Target Latency. RobustMPC's average Video Quality was low for the first two Target Latency settings of 2-2.5s, measuring 0.02-0.16, but started to significantly increase at the third Target Latency setting of 3s, outperforming all other ABRs at Target Latency settings of 4-7.5s. Bola's average Video Quality at the lowest Target Latency was 0.73, then decreased to less than 0.18 for Target Latency between 2.5s and 4.5s; however, it then started to significantly increase at Target Latency of 5-7.5s - peaking at 2.06 when at the highest Target Latency.

Average Quality Variability achieved by Llama remained near constant across all values of Target Latency with differences within 1.24. This was also the case for Panda and Festive, with differences within 0.74 and 0.79 respectively. RobustMPC's average Quality Variability was low at the lowest Target Latency, and sharply increased as the Target Latency increased, peaking at 969.65 when at the highest Target Latency

- approximately twice as high as Llama's average Quality Variability. Bola's average Quality Variability was lower than Llama's up to Target Latency of 5.5s, and higher for Target Latency of 6-7.5s - peaking at 697.36 when at highest Target Latency. Llama was much more stable than RobustMPC for most values of Target Latency, and Bola for Target Latency values of 6-7.5s. However, it was also slightly less stable than Panda and Festive.

Overall performance. Llama achieved the highest average P.1203 MOS for Target Latency of 2-5s, with only up to 0.04 lower average P.1203 MOS than RobustMPC for Target Latency of 5.5-7.5s, where both of the ABRs were slightly outperformed by Bola. At a Target Latency of 2s, average P.1203 MOS achieved by Panda, Festive, RobustMPC, Bola, and Llama was 3.82, 3.97, 2.06, 3.27 and 4.12 respectively. At Target Latency of 4s, it decreased to 3.01 for Bola, and increased to 3.86, 4.04, 4.03, and 4.15 for Panda, Festive, Bola, and Llama respectively. At Target Latency of 6s, it increased to 3.88, 4.07, 4.17, 4.23, and 4.16 for Panda, Festive, RobustMPC, Bola, and Llama respectively.

Out of all five ABR algorithms, Llama achieved the second lowest average Live Latency at Target Latency settings between 2s and 3s, and the lowest average Live Latency for the remaining Target Latency settings of 3.5-7.5s. RobustMPC achieved the lowest average Live Latency for the Target Latency settings of 2-3s, however, it also achieved on average 1.5 worse P.1203 MOS than Llama, indicating a significant deterioration in Quality of Experience. At Target Latency of 3.5-5s Llama achieved the best P.1203 MOS while having the lowest average Live Latency. At Target Latency of 5.5s and above, Llama continued to achieve the lowest average Live Latency, as the cost of slightly worse P.1203 MOS when compared to RobustMPC and Bola, which had on average 0.02 and 0.09 respectively better P.1203 MOS at these settings. Overall, just as in the DASH client, when compared to the four on-demand ABR algorithms, Llama achieved the best balance between minimising the Live Latency and maximising the P.1203 MOS.

Table 7.2: Percentage of sessions for which Llama achieved better P.1203 MOS than the tested on-demand ABR algorithms.

DASH					
		Panda	Festive	RobustMPC	Bola
Target Latency	2s	90%	85%	98%	88%
	4s	57%	63%	98%	97%
	6s	42%	55%	40%	95%
CMAF					
		Panda	Festive	RobustMPC	Bola
Target Latency	2s	92%	65%	99%	84%
	4s	92%	60%	66%	96%
	6s	90%	55%	37%	23%

7.3.2 Discussion

In DASH, Llama achieved 0.31-0.93 better average P.1203 MOS than the on-demand ABR algorithms when the Target Latency was equal to 2s, that is, when the latency is just one segment duration, as well as achieving up to 33% less rebuffering. The only ABR algorithm which had a lower average Rebuffer Ratio at this configuration was RobustMPC, which achieved a value 78% lower than Llama. However, this low average Rebuffer Ratio is a result of RobustMPC having significantly lower average Video Quality, equal to 0.02 and indicative of mostly choosing the lowest level of quality, compared to Llama's average Video Quality of 1.67. This is reflected in the average P.1203 MOS, with Llama achieving 0.93 higher than RobustMPC. At Target Latency of 4s, Llama achieved 0.2-1.4 better P.1203 MOS than other ABR algorithms, and up to 64% less rebuffering. Table 7.2 shows the percentage of sessions for which Llama achieved better P.1203 MOS than other ABRs at different Target Latency settings. Llama achieved better P.1203 MOS than the other ABR algorithms for 85-98%, and 57-98% of sessions at Target Latency of 2s and 4s respectively.

CMAF has been developed to enable reduced latency in live streaming, where the use of CMAF chunks can reduce the minimum latency from one segment duration to a single chunk duration. CMAF chunks combined with HTTP/1.1 Chunked Transfer Encoding allow for the use of small chunks without the additional network overhead. We have tested our new ABR algorithm when used in a client which supports CMAF chunks. At Target Latency of 2s and 4s, Llama achieved 0.15-2.06 and 0.11-1.14 better average P.1203 MOS than the other ABR algorithms, while also reducing rebuffering by up to 68% and 79% respectively. Table 7.2 shows Llama achieved a higher P.1203 MOS than the other ABR algorithms for 65-99% and 60-96% of sessions at Target Latency of 2s and 4s respectively. These results clearly demonstrate that Llama can be used with CMAF and is capable of utilizing CMAF chunks to boost its performance.

At higher values of Target Latency, Llama was outperformed by Bola and RobustMPC in terms of average P.1203 MOS and Video Quality in both DASH and CMAF modes. In DASH mode at the highest Target Latency, when compared to Llama, RobustMPC achieved 28% higher average Video Quality and 0.12 better average P.1203 MOS, while Bola achieved 16% higher average Video Quality and 0.15 better average P.1203 MOS. However, Bola and RobustMPC had 109% and 346% respectively higher average Rebuffer Ratio than Llama. As seen in Table 7.2, at Target Latency of 6s, Llama achieved better P.1203 MOS for 40% and 95% of sessions when compared to RobustMPC and Bola respectively in DASH, and for 37% and 23% of sessions when compared to RobustMPC and Bola respectively in CMAF.

The first design principle behind Llama aims to improve the ABR algorithm's ability to detect and react to deterioration in network conditions quickly. In both DASH and CMAF, for most Target Latency settings, Llama achieved a lower Rebuffer Ratio while producing better P.1203 MOS than the remaining four ABR algorithms. These results indicate that Llama was successful at quickly reacting to negative changes in network conditions, rendering the first design principle effective.

Llama's second design principle aims to produce stable quality bitrate, in order to improve the overall Quality of Experience as well as to improve consumed bandwidth efficiency. This requires the ABR algorithm to minimise the number of unnecessary quality switches. Across all Target Latency settings, Llama's average Video Quality and Quality Variability remained near-constant, indicating that the ABR algorithm prioritises stable quality bitrate even with a significantly increased client video buffer, which, prompted Bola and RobustMPC to adjust the quality bitrate more frequently, resulting in significantly greater Quality Variability when compared to Llama at higher Target Latency settings. Additionally, this principle resulted in Llama having fewer unnecessary quality increases which allowed it to avoid rebuffering in cases where an increase in available bandwidth was shortly followed by a significant decrease.

In the presented results the impact of video quality stability on the overall QoE can be observed. At the highest value of Target Latency, RobustMPC achieved the highest average Video Quality and the second lowest average Rebuffer Ratio, but it did not achieve the highest average P.1203 MOS as this was negatively impacted by its frequent quality switches - which can be seen on the average Quality Variability charts where RobustMPC was the highest by a significant margin. This was the case for both DASH and CMAF. In CMAF mode, at the highest Target Latency setting, Llama achieved 52% lower average Quality Variability and 24% lower average Video Quality than RobustMPC, but its P.1203 MOS was only 0.03 lower.

In summary, Llama outperformed the four prominent on-demand ABR algorithms in terms of P.1203 Mean Opinion Score as well as Rebuffer Ratio, for both DASH and CMAF, at the two lowest Live Delay settings, which are crucial to achieving low latency in live adaptive streaming. In DASH mode, the P.1203 MOS improved by up to 1.4 respectively, and rebuffering reduced by up to 64%, while in CMAF mode, Llama again outperformed the other ABR algorithms, achieving up to 2.06 better P.1203 MOS and reducing rebuffering by up to 79%. Llama improved the P.1203 MOS in 57%-98% and 60%-99% of sessions in DASH and CMAF respectively at the two lowest Live Delay settings.

7.4 Comparison with state-of-the-art low latency ABRs

In this section, the proposed ABR algorithm is evaluated against three low latency ABR algorithms which were published the same year as Llama. These ABR algorithms are Stallion, L2A and LoL+, each designed specifically for low latency live adaptive streaming. The performance of each ABR algorithm is presented using the performance metrics described previously.

7.4.1 Results

In this subsection, we present the results of our evaluation. We compare the performance of Llama against Stallion, L2A, and LoL+ in both DASH and CMAF clients.

We have used the following parameters. The Live Delay was set to values ranging from 1 to 3 segments with the Join Offset set to 0s, 0.5s, 1s, and 1.5s for each Live Delay setting, resulting in 12 Target Latency configurations, from 2s to 7.5s with increments of 0.5s. All 7,000 throughput traces were used for each combination of Live Delay and Join Offset resulting in 84,000 runs per ABR algorithm for one mode of client. In total, the content was delivered to the client 672,000 times, including for all four low latency ABR algorithms and both DASH and CMAF clients.

The performance of all ABR algorithms is reported using the performance metrics described in the previous section, averaged over the 7,000 runs with different bandwidth profiles. We combined Live Delay and Join Offset into a single value of Target Latency, which indicates the effective latency in seconds, calculated as Live Delay multiplied by the segment duration of two seconds and added to the Join Offset value.

7.4.1.1 DASH

For this evaluation, the client was configured in DASH mode and used with segments of duration of two seconds. Figure 7.4 shows the performance of all four low latency

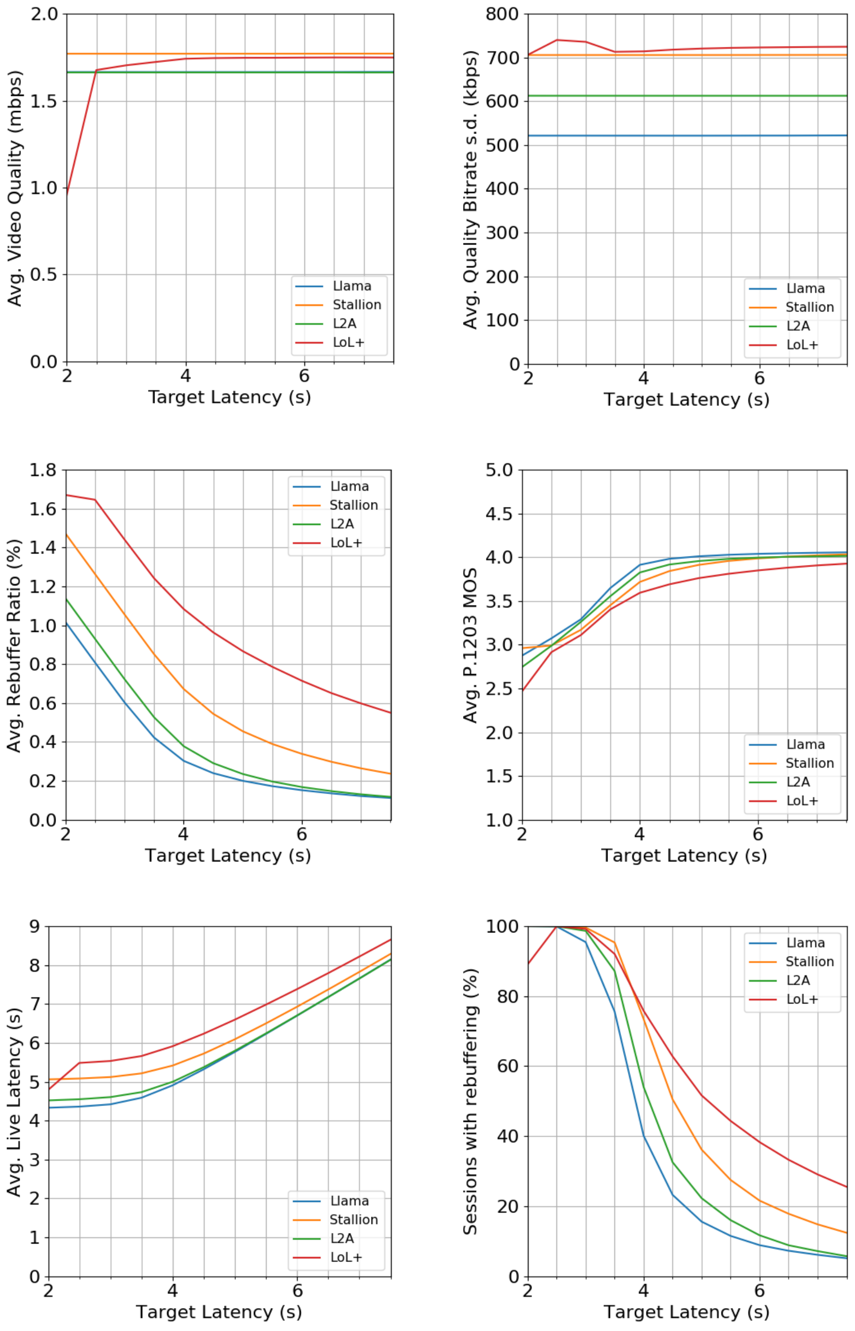

Figure 7.4: Performance of all four low latency ABRs, when used in a DASH client, in terms of average Video Quality, Quality Variability, Rebuffer Ratio, P.1203 MOS, and Live Latency, as well as, the percentage of sessions experiencing rebuffering.

ABR algorithms in terms of average Video Quality, Quality Variability, Rebuffer Ratio, P.1203 MOS, and Live Latency, as well as, the percentage of sessions experiencing rebuffering for each Target Latency setting.

Performance in terms of rebuffering. Llama achieved the lowest average Rebuffer Ratio out of all four low latency ABRs for all values of Target Latency. At the lowest Target Latency, Llama reduced the average Rebuffer Ratio by 0.45, 0.12, and 0.65 when compared to Stallion, L2A, and LoL+ respectively. As Target Latency increased, Llama's improvement in terms of average Rebuffer Ratio over the other ABRs decreased. At the highest Target Latency, Llama reduced the average Rebuffer Ratio by 0.13, 0.01, and 0.44 when compared to Stallion, L2A, and LoL+ respectively.

Llama had the lowest percentage of sessions experiencing rebuffering for Target Latency values of 2.5s and higher. At the lowest Target Latency, LoL+ had the lowest number of sessions experiencing rebuffering at 89%, compared to 100% for the other three ABRs. However, the average Rebuffer Ratio achieved by LoL+ at this setting was significantly higher than for the other three ABRs, suggesting that even though fewer sessions were affected by rebuffering, it was significantly more severe. At Target Latency of 4s, the percentage of sessions experiencing rebuffering for Llama was lower by 33.33, 13.93, and 35.70 than for Stallion, L2A, and LoL+ respectively. As Target Latency increased, these differences decreased. At the highest Target Latency, the percentage of sessions experiencing rebuffering for Llama was lower by 7.27, 0.60, and 20.42 than for Stallion, L2A, and LoL+ respectively.

Performance in terms of video quality. Llama, Stallion, and L2A achieved average Video Quality of 1.67, 1.77, and 1.66 respectively across all values of Target Latency. LoL+ achieved average Video Quality of 0.96 at the lowest Target Latency, and 1.68-1.75 for the remaining values of Target Latency. LoL+ was the only ABR for which the average Video Quality varied across different values of Target Latency. Llama and L2A achieved 0.10 and 0.11 respectively lower average Video Quality than Stallion for all values of Target Latency, as well as, on average 0.07 and 0.08 respectively lower average Video Quality than LoL+ for Target Latency values of 2.5s and higher.

Llama, Stallion, L2A, and LoL+ achieved average Quality Variability of 521-522, 705-706, 612, and 706-740kbps respectively for all values of Target Latency. Llama achieved on average 184, 91, and 201 lower Quality Variability than Stallion, L2A, and LoL+ respectively across all values of Target Latency. Llama did not achieve the highest average Video Quality across all values of Target Latency, however, it did achieve the lowest Quality Variability across all values of Target Latency. This suggests that Llama offered the most stable Video Quality out of all three low latency ABRs at the cost of the reduced average Video Quality.

Overall performance. Llama achieved the best average P.1203 MOS out of all four low latency ABRs for Target Latency values of 2.5s and higher, where on average it improved the P.1203 MOS by 0.09, 0.06, and 0.21 over Stallion, L2A, and LoL+ respectively. At the lowest Target Latency, Stallion achieved 0.09, 0.22, and 0.5 better P.1203 MOS than Llama, L2A, and LoL+ respectively. However, it also had 0.73s, 0.54s, and 0.27s higher average Live Latency than Llama, L2A, and LoL+ respectively.

Llama achieved the lowest average Live Latency for Target Latency values of 2-5.5s, the joint lowest with L2A for Target Latency values of 6-6.5s, and the second lowest average Live Latency, 0.01s higher than L2A, for Target Latency values of 7s and 7.5s. Overall, when used in a DASH client, Llama achieved the best balance between minimising the Live Latency, which is crucial in low latency live streaming, and maximising the P.1203 MOS, which reflects the users' Quality of Experience.

7.4.1.2 CMAF

For this evaluation, the client was configured in CMAF mode and used with segments of duration of two seconds and chunk duration of 0.5 seconds. Figure 7.5 shows the performance of all four low latency ABR algorithms in terms of average Video Quality, Quality Variability, Rebuffer Ratio, P.1203 MOS, and Live Latency, as well as, the percentage of sessions experiencing rebuffering for each value of Target Latency.

Performance in terms of rebuffering. Llama achieved the lowest average Rebuffer Ratio for Target Latency values of 2-5.5s and 6.5s, and the joint lowest with L2A for Target Latency values of 6s and 7-7.5s. At the lowest Target Latency, Llama achieved an average Rebuffer Ratio 0.34, 0.08, and 0.36 lower than Stallion, L2A, and LoL+

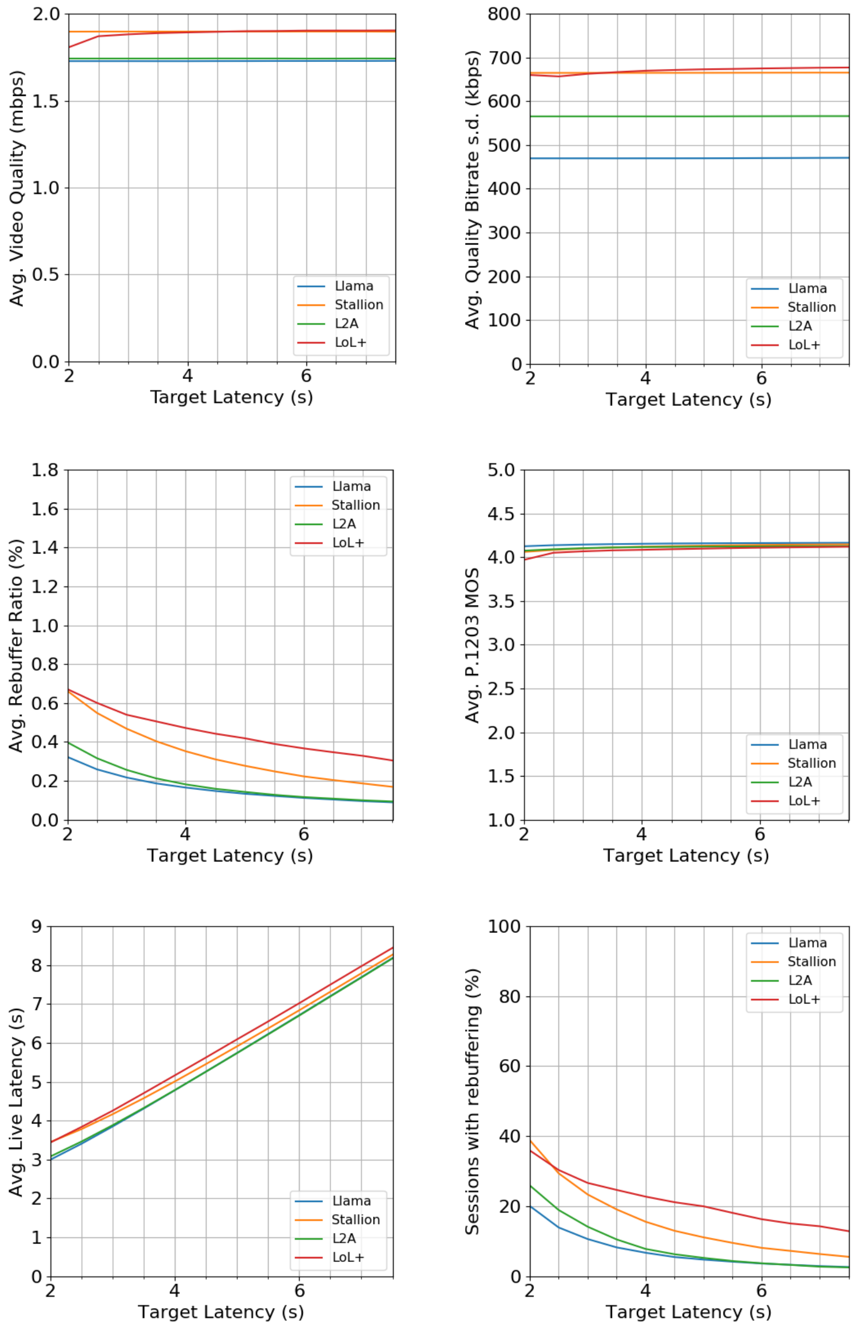

Figure 7.5: Performance of all four low latency ABRs, when used in a CMAF client, in terms of average Video Quality, Quality Variability, Rebuffer Ratio, P.1203 MOS, and Live Latency, as well as, the percentage of sessions experiencing rebuffering.

respectively. As the Target Latency increased, these differences decreased and L2A achieved the same average Rebuffer Ratio as Llama. At the highest Target Latency, Llama and L2A both achieved 0.08 and 0.22 lower Rebuffer Ratio than Stallion and LoL+ respectively.

In terms of the percentage of sessions experiencing rebuffering, Llama outperformed the other three low latency ABRs at Target Latency values of 2-6s, where on average it achieved 10.03, 2.14, and 15.35 lower percentage of sessions experiencing rebuffering than Stallion, L2A, and LoL+ respectively. At Target Latency values of 6.5-7.5s, Llama had the second lowest percentage of sessions experiencing rebuffering, only 0.05-0.17 higher than L2A.

Performance in terms of video quality. Llama, Stallion, and L2A achieved average Video Quality of 1.73, 1.9, and 1.74 respectively across all values of Target Latency. LoL+ achieved an average Video Quality of 1.81-1.91 across all values of Target Latency, and was the only low latency ABR with average Video Quality that varied across different values of Target Latency. Llama, on average achieved 0.17, 0.01, and 0.16 lower average Video Quality than Stallion, L2A, and LoL+ respectively across all values of Target Latency.

Llama, Stallion, L2A, and LoL+ achieved average Quality Variability of 469-470, 665, 565-566, and 657-677kbps respectively for all values of Target Latency. On average, Llama achieved 195, 96, and 200 lower Quality Variability than Stallion, L2A, and LoL+ respectively for all values of Target Latency. Llama did not achieve the highest average Video Quality for all values of Target Latency, however, it did achieve the lowest Quality Variability across all values of Target Latency. Again, just as in the DASH client, Llama offered the most stable Video Quality out of all four low latency ABRs at the cost of the reduced average Video Quality.

Overall performance. Llama achieved the best average P.1203 MOS out of all four low latency ABRs across all values of Target Latency. At the lowest Target Latency, the average P.1203 MOS for Llama, Stallion, L2A, and LoL+ was 4.12, 4.06, 4.07, and 3.97 respectively. As Target Latency increased, Llama's improvement over other ABRs in terms of average P.1203 MOS decreased. At the highest Target Latency, the average

P.1203 MOS achieved by Llama, Stallion, L2A, and LoL+ equalled to 4.16, 4.15, 4.13, and 4.12 respectively.

Llama achieved the lowest average Live Latency for Target Latency values of 2-3.5s, the joint lowest with L2A for Target Latency values of 4-4.5s, and the second lowest average Live Latency, only 0.01s higher than L2A, for Target Latency values of 5-7.5s. Across all values of Target Latency, Llama achieved on average 0.21s, 0.01s, and 0.35s lower average Live Latency, while achieving 0.03, 0.04, and 0.07 better average P.1203 MOS than Stallion, L2A, and LoL+ respectively. Overall, when used in a CMAF client, Llama outperformed the other low latency ABRs in terms of balancing the need to minimise the Live Latency and to maximise the P.1203 MOS.

7.4.1.3 Single Scenario

Figure 7.6 shows the performance of the four ABR algorithms in a single scenario consisting of a CMAF client configured to Live Delay of 1 segment and Join Offset of 0s, resulting in Target Latency of 2s. The top chart shows the available bandwidth and the throughput estimated by each ABR. There are two lines for Llama, the dashed line shows the throughput of the most recent segment and the solid line shows the harmonic mean of 20. The next two charts show the quality and live latency. Since no catch-up mechanism is employed at the client, an increase in live latency indicates a rebuffering event of the same duration. The P.1203 MOS equalled to 3.8, 3.6, 3.5, and 3.4, while the average Live Latency was 2.6s, 3.8s, 3.5s, and 4.5s for Llama, Stallion, L2A, and LoL+ respectively.

In this throughput trace, at 33s the bandwidth increased from 2.3 to 6.2mbps, then 3s later further increased to 8.8mbps and 2s later to 9.3mbps, and finally after only 2s it dropped to 3.3mbps. Stallion overestimated the available bandwidth after this temporary increase occurred and selected the highest quality, causing rebuffering of 1.6s. L2A increased the quality to the highest, but only briefly, and quickly switched down by two quality indexes to avoid rebuffering. LoL+ chose the highest quality briefly twice, and gradually switched down by two quality indexes to avoid rebuffering. Llama did not increase the quality when the bandwidth improved briefly and hence did not

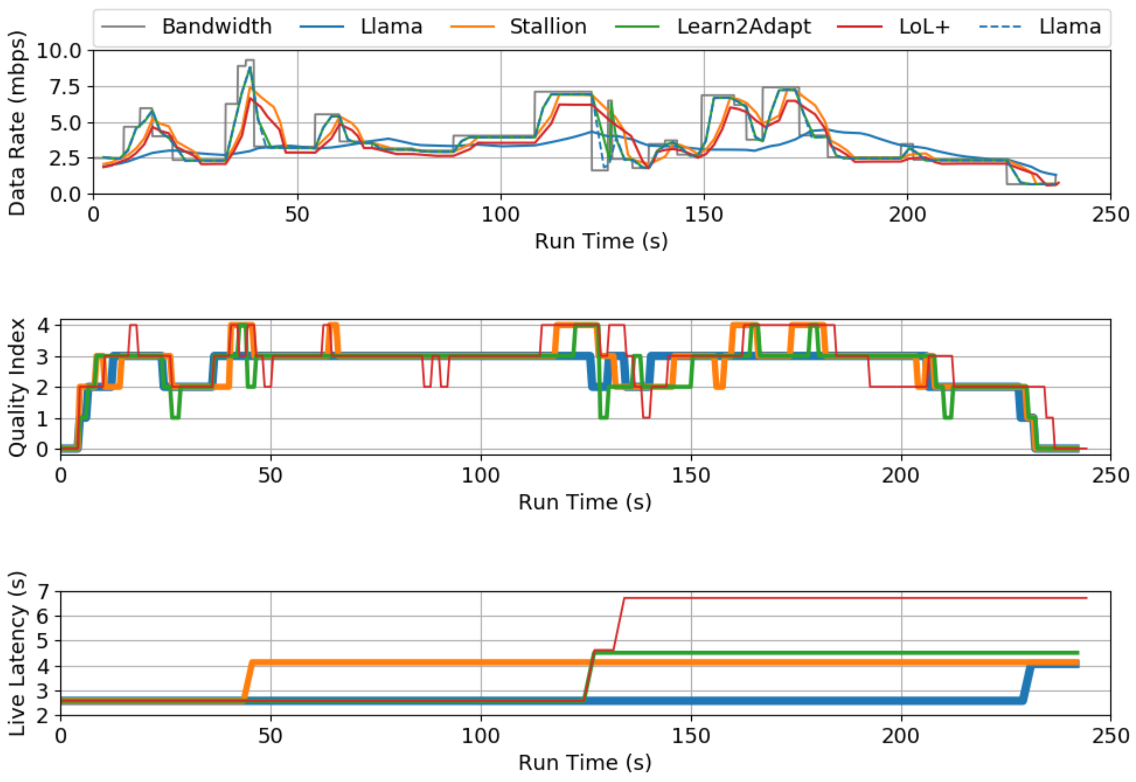

Figure 7.6: Performance of all four low latency ABRs in a single scenario, presented in terms of throughput estimation, video quality and live latency.

need to switch down to avoid rebuffering, resulting in constant video quality.

At 123s, the bandwidth dropped from 7.1 to 1.6mbps, after 4s it briefly increased to 6.5mbps for 1s and finally dropped to 2.4mbps. Stallion, operating at a higher buffer level due to increased live latency by the previous rebuffering event, gradually reduced quality by two indexes to avoid rebuffering. L2A was too slow to switch down from the highest quality resulting in rebuffering of 1.9s, after which it reduced the quality by 3 indexes instantly, and slowly increased the quality ignoring the spike in bandwidth. LoL+ was also too slow to reduce the quality resulting in rebuffering of 2s, after which, it reduced the quality by only one index and when the bandwidth spiked it again increased the quality to the highest leading to further rebuffering of 2.1s. Llama avoided rebuffering by quickly reducing the quality by one index, then when the bandwidth spiked, it increased the quality by one index back, and to avoid the second potential rebuffering event it decreased the quality by one index again.

At 225s, the bandwidth dropped from 2.4 to 0.7mbps. Llama decreased the quality by one index but suffered rebuffering anyway. Stallion and L2A did exactly the same, however, avoided rebuffering by already operating at a higher buffer level due to the previous rebuffering which increased their live latency. LoL+ increased the quality and avoided rebuffering as it was operating at a much higher buffer level, with 2.2-4.1s higher live latency than other ABRs.

Throughout the entire trace, Llama never switched to the highest quality while the three other ABRs frequently tried to reach the highest quality leading to frequent quality switches. In some cases, such as at around 40s, the other three ABRs had to over-correct their decision by reducing the quality to below the pre-increase level to avoid rebuffering, leading to even higher video quality instability. Llama achieved 0.2, 0.3, and 0.4 better P.1203 MOS while having 1.2s, 0.9s, and 1.9s lower average Live Latency than Stallion, L2A, and LoL+ respectively.

Table 7.3: Percentage of sessions for which Llama achieved better P.1203 MOS than other low latency ABR algorithms.

DASH				
		Stallion	L2A	LoL+
Target Latency	2s	41%	72%	75%
	4s	69%	81%	79%
	6s	41%	81%	59%
CMAF				
		Stallion	L2A	LoL+
Target Latency	2s	50%	79%	67%
	4s	45%	78%	48%
	6s	43%	78%	44%

7.4.2 Discussion

In our evaluation, we have compared the performance of Llama against three low latency ABR algorithms: Stallion, L2A, and LoL+. There is no catch-up mechanism employed at the client, meaning, any rebuffering will increase the client's Live Latency for the remainder of the streaming session. In some cases, it can lead to better ABR performance as the client is now operating with a larger potential buffer due to the increased Live Latency. This means that an ABR algorithm which achieves comparable P.1203 at a lower average Live Latency is the better performer due to operating in potentially more challenging conditions.

In DASH, at Target Latency of 2s, Llama achieved 0.19-0.73s lower average Live Latency than other ABR algorithms. Llama achieved 0.13 and 0.41 better P.1203 MOS than L2A and LoL+, while reducing rebuffering by 11% and 39% respectively. Stallion achieved 0.09 better average P.1203 MOS than Llama, at the cost of 0.73s higher average Live Latency and 31% more rebuffering. At Target Latency of 4s, Llama reduced the average Live Latency by 0.09-1s, while having 0.08-0.32 better average P.1203 MOS and 21-72% less rebuffering than other ABR algorithms. At Target Latency of 6s, Llama reduced the average Live Latency by 0-0.68s and improved the average P.1203 MOS by 0.04-0.19 while reducing rebuffering by 12-79% when compared to other ABR algorithms. Table 7.3 shows that Llama improved 41-75%, 69-81% and 41-59% of sessions for Target Latency settings of 2s, 4s, and 6s respectively.

In CMAF, at the Target Latency of 2s, Llama achieved 0.09-0.46s lower average Live Latency, while having 0.05-0.15 better average P.1203 MOS and 20-52% less rebuffering than other low latency ABR algorithms. At Target Latency of 4s, Llama reduced the average Live Latency by 0-0.38s, while improving the average P.1203 MOS by 0.03-0.07 and reducing rebuffering by 6-64% when compared to other ABR algorithms. At Target Latency of 6s, Llama achieved 0.13s and 0.31s lower average Live Latency, while also improving the average P.1203 MOS by 0.02 and 0.05 and reducing rebuffering by 50% and 70% when compared to Stallion and LoL+ respectively. L2A achieved 0.01s lower average Live Latency than Llama, at the cost of 0.04 lower average P.1203 MOS and 8% more rebuffering. Table 7.3 shows Llama improved 50-67%, 45-78%, and 43-78% of sessions in terms of P.1203 MOS for Target Latency settings of 2s, 4s, and 6s respectively.

Llama's adaption logic is based on two design principles. The first design principle is quick and effective rebuffering avoidance. For most values of Target Latency, Llama achieved the lowest average Rebuffer Ratio while having the highest average P.1203 MOS, when compared to the three other ABR algorithms. At the lowest Target Latency setting, Llama reduced rebuffering by up to 39% and 52%, while improving the average P.1203 MOS by up to 0.41 and 0.15 in DASH and CMAF clients respectively. Across all values of Target Latency on average, Llama reduced rebuffering by 44% and 42% in DASH and CMAF respectively when compared to all three other ABR algorithms. These results indicate that Llama is able to react to worsening network conditions in a timely manner and avoid potential rebuffering events effectively.

The second design principle behind Llama's adaptation logic aims to produce a stable quality bitrate. This requires the ABR algorithm to minimise the number of unnecessary quality bitrate switches. When compared to the other low latency ABR algorithms, Llama achieved the lowest average Quality Variability while having the highest average P.1203 MOS for most Target Latency settings. These results indicate that the proposed ABR algorithm can offer a stable quality bitrate to improve the Quality of Experience.

In summary, across all Target Latency settings, Llama outperformed the three other low latency ABR algorithms, in terms of balancing the requirements to maximise P.1203 MOS and to minimise the Live Latency, in both DASH and CMAF clients. On average across all values of Target Latency, Llama achieved 0.08 better P.1203 MOS and 43% less rebuffering than the three other low latency ABR algorithms.

7.5 Summary

In this chapter, we have evaluated the suitability of Llama, a Low Latency Adaptive Media Algorithm, for low latency live adaptive streaming. The performance of the proposed ABR algorithm was compared against four prominent on-demand ABRs and three low latency ABRs - published the same year as Llama.

Both evaluation experiments were carried out using a common methodology, compromising a simulation model which allowed for faster than real-time evaluation under an extensive set of client configurations. With the intention of applying realistic network conditions, throughput traces were derived from CDN logs of a real commercial online TV service. The performance of each ABR algorithm was reported using a variety of performance metrics, including the average Live Latency, indicating the latency experienced during a streaming session, as well as, the P.1203 MOS, indicating the overall Quality of Experience of the streaming session.

In total, Llama was evaluated against seven different ABR algorithms, across multiple client configurations and in a variety of realistic network conditions. Llama showed the best performance as in most cases it achieved the lowest average Live Latency, crucial to ensuring a low latency streaming session, while achieving the best P.1203 MOS, representative of overall Quality of Experience perceived by the user.

Chapter 8

Conclusions

This thesis aimed to investigate and improve low latency live adaptive streaming over the Internet. Online multimedia streaming has evolved radically over the past three decades, driven by the ever-increasing demand for such services. A relatively recent change has seen the advancement of multimedia streaming towards highly scalable solutions such as HTTP Adaptive Streaming, where the content is encoded into multiple quality bitrates, each with different bandwidth requirements, and divided into segments of equal duration. The resulting segments, of various quality bitrates which are interchangeable, are hosted on standard HTTP servers. In this stateless architecture, the streaming session is driven by the client, which requests the content by issuing standard HTTP requests, segment by segment.

In adaptive streaming, an ABR algorithm is employed, most of the time at the client side, which is tasked with the monitoring of network conditions and adjusting the quality bitrate of each segment to be requested by the client. Its main task is to maximise the Quality of Experience (QoE) of the streaming session, primarily by adjusting the quality bitrate to match the available bandwidth over time. The quality bitrate can be changed by any number of available encoding bitrates, every segment duration. This thesis aimed to investigate the adaption logic, a crucial aspect of adaptive streaming, when employed in low latency conditions which consist of unique and challenging constraints.

This thesis aimed to propose an ABR algorithm designed specifically for low latency live adaptive streaming, where the task of the ABR algorithm is further expanded as it now needs to attempt to minimise the average latency while trying to maximise the Quality of Experience (QoE) of the streaming session. This process requires the design and implementation of the proposed solution, as well as rigorous evaluation which compares the performance of the proposed solution against state-of-the-art ABR algorithms in a variety of network conditions and latency settings.

Additionally, this thesis aimed to investigate the Common Media Application Format (CMAF), a recent development in the area of HTTP Adaptive Streaming which can enhance the Quality of Experience (QoE) in low latency live adaptive streaming. CMAF allows for further division of content from segments to smaller chunks of equal duration. Each chunk can be played out as soon as received by the client, bringing the availability of segments forward as they can be fetched by the client as soon as the first chunk is generated - as opposed to regular DASH where the client needs to wait for the entire segment to be generated. The quality bitrate can still be changed at the segment level only in an effort to significantly reduce the encoding overhead of CMAF chunks.

8.1 Thesis Contributions

In this thesis, low latency live adaptive streaming has been investigated, leading to the creation of a new ABR algorithm, designed specifically for low latency conditions. This required the analysis of existing prominent on-demand ABR algorithms, in the two main methods of low latency live adaptive streaming, to investigate the impact of reduced latency on ABR performance. Additionally, this required extensive subjective testing to be carried out, to further investigate the impact of changes in video quality bitrate on the overall Quality of Experience. Based on the results of these investigations, design goals were derived for a new low latency ABR algorithm. This thesis makes the following contributions in relation to low latency live adaptive streaming.

8.1.1 Investigation of low latency live adaptive streaming and the impact of latency on ABR algorithm performance

In Chapter 5 of this thesis, two main low latency live adaptive streaming methods and their impact on the performance of prominent on-demand ABR algorithms, and in turn on the overall Quality of Experience, have been analysed. The first method utilised regular DASH streaming, where content is divided into segments of equal duration. The second method combined CMAF, where segments are further divided into chunks of equal duration, with HTTP/1.1 Chunked Transfer Encoding. Each CMAF chunk can be played out as soon as received by the client, however, the quality bitrate can still be changed at the segment level only, in order to significantly reduce the encoding overhead. HTTP/1.1 Chunked Transfer Encoding allows for partial HTTP responses, which, are utilised to reduce the number of HTTP requests needed. The client requests each segment once, just as in regular DASH, and fetches the first chunk, after which the server will transmit each of the remaining chunks when they will become available, without additional client requests, effectively reducing the network overhead.

CMAF chunks can be beneficial in low latency conditions, as the client no longer needs to wait for an entire segment to be generated, instead, the client can request each segment as soon as the first chunk becomes available, which once downloaded can be added to the client video buffer. In low latency conditions, with latency typically below 6s, the client video buffer is severely limited as its maximum potential level cannot exceed the client's latency. In such conditions, chunks can be beneficial as the buffer can be expanded more frequently, periodically every chunk duration as opposed to every segment duration in regular DASH.

Both methods of low latency live adaptive streaming have been tested using four prominent on-demand ABR algorithms across four latency settings. The results showed that the target latency setting has a significant impact on ABR algorithm performance. At the two lowest target latency settings, 2s and 4s, the tested ABR algorithms performed poorly when compared to their performance at the two highest latency settings of 6s and 8s. Additionally, it can be observed that as the target latency setting increased, the performance of each ABR algorithm improved. However, this improvement was not universal across the four ABR algorithms tested.

Furthermore, the results showed that CMAF chunks can improve ABR performance, as the measured Quality of Experience improved in most of the cases. However, in some cases the improvement was insignificant. Additionally, in some cases the measured Quality of Experience was worse when an ABR algorithm was deployed in a CMAF client. This thesis confirmed that CMAF can significantly improve the ABR performance in low latency live adaptive streaming, however, it also highlighted that the ABR algorithm must be compatible with CMAF chunks to maximise the potential improvement.

8.1.2 Investigation of quality switches and their impact on the overall Quality of Experience in adaptive streaming

Chapter 4 of this thesis presented the results of an extensive subjective study, carried out to investigate how users perceive different forms of changes in video quality bitrate. It consisted of 28 video quality patterns, divided into seven experiment groups, each testing a different type or degree of video quality impairment. The experiments were carried out remotely, utilising an online survey, developed to support the playback of test sequences without the addition of unintended quality impairments. A standard video quality assessment method was used, Absolute Category Rating (ACR), where the participant is presented with one test sequence at a time, after which they are asked to rate it on a scale of 1-5 before proceeding to the next test sequence. The participant responses were processed according to the standard methodology, with Mean Opinion Score (MOS) for each test sequence calculated, reflecting the Quality of Experience (QoE) measured. The experiment encompassed 14 pieces of content and 63 participants.

The results of this extensive subjective study highlight the importance of stable video quality. The findings were as follows. Stable quality bitrate, where fewer quality bitrate switches occurred, was preferred even if it resulted in a 42% lower average bitrate. Short temporary increase or decrease in quality bitrate has no significant impact on Quality of Experience. Delivery of content at the lowest quality bitrate, even if temporary, had a significant negative impact on the Quality of Experience. Gradual multi-level quality bitrate changes, that is where the quality bitrate changes at most by one encoding bitrate every segment, were preferred to instant multi-level quality bitrate changes, where the quality bitrate can change by any number of encoding bitrates between two segments. However, in the case of multi-level quality bitrate switches over only two encoding bitrates, there was no significant difference in MOS between gradual and instant switches. Based on the key findings of this subjective study, this thesis outlined design goals for a new low latency ABR algorithm in Chapter 5.

8.1.3 Llama, a Low Latency Adaptive Media Algorithm

Chapter 5 of this thesis proposed a new ABR algorithm, Llama, designed specifically for low latency live adaptive streaming. Design analysis was performed and presented, compromising of assessing the impact of latency on the performance of prominent on-demand ABR algorithms in low latency live adaptive streaming. Building on the results of the analysis as well as the key findings of the subjective study presented in Chapter 4, this thesis constructed design goals for an ABR algorithm designed to operate in low latency conditions, highlighting the need for the ABR algorithm to be capable of operating in environments where the client video buffer is significantly restricted. Additionally, a low latency ABR algorithm must be compatible with CMAF, as it can improve the ABR performance significantly at low latency settings.

This thesis presented Llama, a Low Latency Adaptive Media Algorithm, with its adaption logic and pseudo-code detailed and explained. The proposed ABR algorithm operates on two design principles, each constructed to satisfy a different subset of the design goals established earlier. The first design principle focuses on quick and robust detection of worsening network conditions with the intention of instant reaction to adjust the quality bitrate, which is crucial in low latency live adaptive streaming where the client video buffer is limited, giving the ABR algorithm limited time to react to degradation in network conditions before it is depleted. The second design principle is tasked with achieving a stable quality bitrate in order to maximise the Quality of Experience.

Each design principle required a different method for estimating the network conditions. Hence, the proposed ABR algorithm employs two independent throughput estimation methods, each optimised for a different purpose. Traditional bandwidth-based ABR algorithms rely on a single throughput estimation for their quality bitrate decisions. The utilisation of multiple throughput estimation methods resulted in an ABR algorithm best suited for low latency live adaptive streaming as it is capable of detecting and reacting to deterioration in network conditions quickly to minimise the average latency while aiming to provide stable quality bitrate based on long-term throughput estimates to maximise the Quality of Experience.

Chapter 7 of this thesis presented an extensive evaluation of the proposed ABR algorithm. It was evaluated against four prominent on-demand ABR algorithms, as well as, three state-of-the-art low latency ABR algorithms, published the same year as Llama. The evaluation was conducted using a simulation model, described in Chapter 6, which supports two modes of low latency live adaptive streaming, DASH and CMAF, and allows for faster than real-time evaluation of streaming sessions. All eight ABR algorithms were evaluated using 12 latency settings, in realistic network conditions based on 7000 throughput traces derived from throughput data of a real commercial live TV service. The performance of ABR algorithms was presented using individual QoE metrics, the average latency, and the P.1203 QoE model, a standardised model which predicts the MOS of each streaming session, representing the overall Quality of Experience, without the need for subjective testing.

This thesis found the proposed ABR algorithm to outperform the prominent on-demand ABR algorithms at low latency settings, as well as, the state-of-the-art low latency ABR algorithms at all tested latency settings, in both DASH and CMAF modes. Llama achieved the best balance between minimising the average latency, a crucial aspect in low latency live adaptive streaming, and maximising the P.1203 MOS, representing the overall Quality of Experience. The proposed ABR algorithm improved the Quality of Experience in 41-99% of streaming sessions when compared to the seven other ABR algorithms at the lowest latency setting while achieving the lowest average latency at most settings. The utilisation of multiple throughput measurements to fulfil the two design principles of Llama has been effective, as the proposed ABR algorithm has achieved the most stable quality bitrate while significantly reducing rebuffering when compared to the remaining seven ABR algorithms at most settings. Additionally, the proposed ABR algorithm was found to successfully leverage the use of chunks when in CMAF mode, as its performance improved significantly when compared to its performance in the DASH mode.

8.1.4 Commercial and Wider Research Impacts

This thesis presented a simulation model, detailed in Chapter 6, which imitates two modes of low latency live adaptive streaming while allowing for faster than real-time evaluation of streaming sessions. It supports low latency live adaptive streaming using regular DASH as well as CMAF chunks combined with HTTP/1.1 Chunked Transfer Encoding. The model supports extensive client configuration, including the target latency and traffic shaping settings. Additionally, the model is extensible, allowing for the implementation of new ABR algorithms. This framework can be used to further investigate live adaptive streaming, as well as test and prototype future ABR algorithms. The source code of the simulation model, along with basic deployment instructions, have been published on GitHub[99], allowing the research community to utilise this resource freely. The simulation model has already been used by other researchers in [18].

The extensive evaluation of ABR algorithms, presented in Chapter 7, depended on the recreation of realistic network conditions. The utilised simulation model allowed for traffic shaping to be employed between the server and client, however, it required an appropriate input. In order to source suitable traffic profiles, which would create realistic network conditions during the streaming sessions, this thesis utilised data from CDN logs of a real commercial live online TV service, BT Sport 1. Based on this data, 7000 throughput traces have been derived. Each throughput trace was four minutes long, with bandwidth measurements below the highest encoding bitrate of the ABR ladder tested, 4800kbps. This dataset has been published on GitHub[128], making it available to the research community.

This thesis has proposed a new ABR algorithm, Llama, designed specifically for low latency live adaptive streaming, where the client video buffer is severely restricted due to the desired low latency posing challenging conditions. The proposed ABR algorithm operates using two design principles, making it quick to detect and react to deterioration in network conditions, while being careful about increasing the quality bitrate in order to provide stable quality bitrate throughout the streaming session. Llama utilised two independent throughput measurement methods, each deployed to satisfy a different design principle. Typically, bandwidth-based ABR algorithms utilise only a single throughput estimation method. In September 2020, a patent application for the innovative design behind Llama, titled ”Using Network Throughput Measurements for Adaptive Bit Rate Streaming”, has been filed by British Telecommunications.

8.2 Future Work

This thesis has investigated low latency live adaptive streaming, focusing on the impact of low latency on the performance of ABR algorithms, and in turn the overall Quality of Experience. There are several areas of future work which can expand on the outcomes presented in this thesis.

The Common Media Application Format (CMAF) has been investigated in this thesis. Design analysis, presented in Chapter 5, investigated the impact of latency on the performance of prominent on-demand ABR algorithms in two methods of delivery of low latency live adaptive streaming. The first method was the use of regular DASH, with short segments of 2s. The second method employed CMAF, combined with HTTP/1.1 Chunked Transfer Encoding, with four chunks, 0.5s each, per segment. The results showed the negative impact of low latency on ABR algorithm performance, as well as the positive impact of CMAF chunks on ABR performance, especially at the two lowest latency settings. This investigation could be further expanded to improve our understanding of CMAF's impact on ABR performance. This would consist of testing more encoding settings to find out how many chunks per segment result in the best trade-off between ABR performance improvement and the encoding as well as network overhead of CMAF chunks delivered by HTTP partial responses. In the evaluation work presented in this thesis, only one setting has been tested which consisted of four 0.5s chunks per segment, however, a segment can be divided into any number of chunks. Additionally, alternative methods of chunks delivery need to be investigated, such as HTTP/2 Server Push.

The impact of video quality bitrate changes on Quality of Experience has been further investigated in this thesis. In adaptive streaming, the quality bitrate can be changed at the segment level, every segment duration, by any number of encoding bitrates. This can lead to scenarios in which the quality bitrate is abruptly changed across the entire quality bitrate spectrum, which, can have a significant impact on the Quality of Experience of a streaming session. In Chapter 4, an extensive subjective study has been presented. It encompassed 28 video quality patterns, divided into seven experiment groups, each testing a different type or magnitude of video quality impairment. Test sequences for the selected video quality patterns were produced using a total of 14 video clips, consisting of sports content, a type of content mainly used in live streaming. This study could be further expanded in the future to include more types of content, to make the results applicable to scenarios outside low latency live streaming. Additionally, more video quality patterns need to be tested as, while 28 patterns were a considerably large number, they were only a small subset of all possible video quality variations. Furthermore, the number of quality bitrates available, and different encoding settings and bitrates, can have a significant impact on the perceived quality of changes in video quality in adaptive streaming, requiring further investigation in this area.

A new ABR algorithm, designed specifically for low latency live adaptive streaming, has been presented in this thesis. The proposed ABR algorithm utilises two independent throughput estimation methods to satisfy two distinct design principles. The first throughput estimation method follows the available bandwidth closely by measuring the throughput of the most recent single segment, enabling the ABR algorithm to detect degradation in available bandwidth quickly. The second throughput estimation method estimates the long-term available bandwidth by calculating the harmonic mean of throughput of the past twenty segments, enabling the ABR algorithm to increase the quality bitrate only when the bandwidth appears to be improving in the long term. Recent advancements in the area of ABR research have seen the employment of machine-learning algorithms to improve the Quality of Experience in on-demand streaming. Future work would investigate the suitability of machine-learning algorithms for the adaptation logic in low latency live adaptive streaming. The proposed ABR algorithm, Llama, could be improved by employing machine-learning algorithms to optimise the two individual throughput estimation methods used.

8.3 Summary

This thesis has aimed to investigate low latency live adaptive streaming, primarily focusing on the adaptation logic necessary to provide better Quality of Experience while trying to minimise the latency. The impact of latency on ABR performance has been investigated in this thesis, encompassing two methods of low latency live adaptive streaming, DASH and CMAF. The results have shown low latency conditions to be challenging as they led to a significant deterioration in the performance of prominent on-demand ABR algorithms. However, this deterioration was of considerably smaller magnitude when the ABR algorithms were used in CMAF mode, where the segments are further divided into chunks.

This thesis has investigated the impact of various video quality impairments on the Quality of Experience. An extensive subjective study has been conducted, testing 28 video quality patterns using 14 video clips and standard methodology for video quality assessment. The results have shown a strong preference for stable quality bitrate, where fewer changes in quality bitrate are present, even if it resulted in a significantly lower average bitrate, rendering the average bitrate an unreliable predictor of Quality of Experience in adaptive streaming.

Llama, a Low Latency Adaptive Media Algorithm, has been presented in this thesis. It is one of the first four ABR algorithms designed specifically for low latency live adaptive streaming - all published the same year. The proposed ABR algorithm was designed based on the results of the investigation of low latency live adaptive streaming as well as the results of the subjective study. It utilises two independent throughput measurement methods, each designed to satisfy a different design principle. This results in an ABR algorithm that is quick to react to deterioration in network conditions while being capable of providing stable quality bitrate.

The proposed ABR algorithm has been rigorously evaluated in this thesis, using a purposely developed simulation model which allows for faster than real-time evaluation of streaming sessions. It has been evaluated in two methods of low latency delivery, DASH and CMAF, under a variety of latency settings in realistic network conditions, based on CDN data of a real commercial live TV service. The proposed ABR algorithm has been compared against four prominent on-demand ABR algorithms as well as three state-of-the-art low latency adaption solutions. Llama achieved the best balance between maximising the Quality of Experience and minimising the average latency when compared to the seven other ABR algorithms in most settings.

Appendix A

Subjective Study: Participant Information Sheet

Lancaster University

School of Computing and Communications

Participant information sheet

For further information about how Lancaster University processes personal data for research purposes and your data rights please visit our webpage: www.lancaster.ac.uk/research/data-protection

I am a PhD student at Lancaster University, and I would like to invite you to take part in a research study about quality impairments in online video streaming.

Please take time to read the following information carefully before you decide whether or not you wish to take part.

What is the study about?
This study aims to improve our understanding of how users perceive video quality in online video streaming scenarios.

Why have I been invited?
I have approached you because I am interested in people who are familiar with online video streaming to find out how they perceive video quality. I would be very grateful if you would agree to take part in this study.

What will I be asked to do if I take part?
If you decided to take part, this would involve the filling out an online survey. It will take around 15 minutes and will consist of short video clips which you will be asked to rate.

What are the possible benefits from taking part?
You will contribute to improving the video quality in online video streaming by giving the research community a better understanding of users' perception of video quality.

Do I have to take part?
No. It's completely up to you to decide whether or not you take part. You are free to withdraw without giving any reason up to the time you have completed the survey.

What if I change my mind?
After you complete the survey, your response will be anonymised, and we will not able to remove it.

What are the possible disadvantages and risks of taking part?

There no risks involved in this study, but it will take around 15 minutes of your time to complete the study.

How will my data be stored?
Your response will be stored in encrypted files (that is no-one other than me, the researcher will be able to access them) and on password-protected computers. In accordance with University guidelines, I will keep the data securely for a minimum of ten years.

How will we use the information you have shared with us and what will happen to the results of the research study?
I will use the data you have shared for academic purposes only. This will include a journal article and my PhD thesis. I may also present the results of my study at academic conferences.

When writing up the findings from this study, I would like to reproduce some of the responses you shared with me. When doing so, I will only use aggregated data from all participants, so you cannot be identified in our publications.

Who has reviewed the project?
This study has been reviewed and approved by the Faculty of Science and Technology Research Ethics Committee.

What if I have a question or concern?
If you have any queries or if you are unhappy with anything that happens concerning your participation in the study, please contact myself at {t.lyko@lancaster.ac.uk}, or my PhD supervisor, Matthew Broadbent, at {m.broadbent@lancaster.ac.uk}.

If you have any concerns or complaints that you wish to discuss with a person who is not directly involved in the research, you can also contact Adrian Friday at {a.friday@lancaster.ac.uk}.

Appendix B

Subjective Study: Consent Form

Consent Form

----- Survey page start -----

Consent

1. I confirm that I have read and understand the information sheet for the above study. I have had the opportunity to consider the information, ask questions and have had these answered satisfactorily.
2. I understand that my participation is voluntary and that I am free to withdraw at any time during the survey, without giving any reason. Once I complete the survey, I will not be able to withdraw my response as it will impossible to identify it.
3. I understand that any information given by me may be used in future reports, academic articles, publications or presentations by the researcher/s, but my personal information will not be included, and I will not be identifiable.
4. I understand that data will be kept according to University guidelines for a minimum of 10 years after the end of the study.
5. I agree to take part in this study.

By pressing ‘Next’ you agree with statements above.

[Next button]

---- Survey page end ------

Appendix C

Subjective Study: Invitation Email

Email Invitation

Hello,

As part of my PhD studies, I would like to invite you to take part in a research study about quality impairments in online video streaming.

This study aims to improve our understanding of how users perceive video quality in online video streaming scenarios.

If you decided to take part, this would involve filling out an online survey. It will take around 15 minutes and will consist of short video clips which you will be asked to rate.

The survey can be found at: [link to the online survey]

By taking part in this study, you will contribute to improving the video quality in online video streaming by giving the research community a better understanding of users' perception of video quality.

Please find attached the participant information sheet for more detail. If you have any questions, feel free to contact me.

Thank you for considering your participation in this project.

Best regards,
Tomasz Lyko

References

[1] RFC 8216. "HTTP Live Streaming". In: (2017). URL: `https://tools.ietf.org/html/rfc8216`.

[2] Saamer Akhshabi et al. "Server-Based Traffic Shaping for Stabilizing Oscillating Adaptive Streaming Players". In: *Proceeding of the 23rd ACM Workshop on Network and Operating Systems Support for Digital Audio and Video*. NOSSDAV '13. Oslo, Norway: Association for Computing Machinery, 2013, pp. 19–24. ISBN: 9781450318921. DOI: `10.1145/2460782.2460786`.

[3] Brahim Allan, Mike Nilsson, and Ian Kegel. "A Subjective Comparison of Broadcast and Unicast Transmission Impairments". In: *SMPTE Motion Imaging Journal* 128.6 (2019), pp. 1–15.

[4] Siddhartha Annapureddy et al. "Is high-quality VoD feasible using P2P swarming?" In: *Proceedings of the 16th international conference on World Wide Web*. 2007, pp. 903–912.

[5] Christos G Bampis and Alan C Bovik. "Learning to predict streaming video QoE: Distortions, rebuffering and memory". In: *arXiv preprint arXiv:1703.00633* (2017).

[6] Christos G. Bampis, Zhi Li, and Alan C. Bovik. "Continuous Prediction of Streaming Video QoE Using Dynamic Networks". In: *IEEE Signal Processing Letters* 24.7 (2017), pp. 1083–1087. DOI: `10.1109/LSP.2017.2705423`.

[7] N. Barman and M. G. Martini. "QoE Modeling for HTTP Adaptive Video Streaming–A Survey and Open Challenges". In: *IEEE Access* 7 (2019), pp. 30831–30859.

[8] A. Bentaleb et al. "A Survey on Bitrate Adaptation Schemes for Streaming Media Over HTTP". In: *IEEE Communications Surveys Tutorials* 21.1 (2019), pp. 562–585.

[9] Abdelhak Bentaleb et al. "Bandwidth Prediction in Low-latency Chunked Streaming". In: *Proceedings of the 29th ACM Workshop on Network and Operating Systems Support for Digital Audio and Video.* NOSSDAV '19. Amherst, Massachusetts: ACM, 2019, pp. 7–13. ISBN: 978-1-4503-6298-6. DOI: `10.1145/3304112.3325611`.

[10] Jan A Bergstra and CA Middelburg. "ITU-T Recommendation G. 107: The E-Model, a computational model for use in transmission planning". In: (2003).

[11] *BigBuckBunny*. URL: `https://peach.blender.org/`.

[12] Niels Bouten et al. "QoE-driven in-network optimization for adaptive video streaming based on packet sampling measurements". In: *Computer networks* 81 (2015), pp. 96–115.

[13] N. Bouzakaria, C. Concolato, and J. Le Feuvre. "Overhead and performance of low latency live streaming using MPEG-DASH". In: *IISA 2014, The 5th International Conference on Information, Intelligence, Systems and Applications.* 2014, pp. 92–97.

[14] Joachim Bruneau-Queyreix et al. "MS-Stream: A multiple-source adaptive streaming solution enhancing consumer's perceived quality". In: *2017 14th IEEE Annual Consumer Communications Networking Conference (CCNC).* 2017, pp. 427–434. DOI: `10.1109/CCNC.2017.7983147`.

[15] Brad Cain et al. *RFC 3376: Internet group management protocol, version 3.* 2002.

[16] Mathieu Carnec, Patrick Le Callet, and Dominique Barba. "Objective quality assessment of color images based on a generic perceptual reduced reference". In: *Signal Processing: Image Communication* 23.4 (2008), pp. 239–256.

[17] Miguel Castro et al. "Splitstream: High-bandwidth multicast in cooperative environments". In: *ACM SIGOPS operating systems review* 37.5 (2003), pp. 298–313.

[18] Miguel Catalan-Cid et al. "FALCON: joint fair airtime allocation and rate control for DASH video streaming in software defined wireless networks". In: *Proceedings of the 30th ACM Workshop on Network and Operating Systems Support for Digital Audio and Video*. 2020, pp. 14–20.

[19] Chao Chen et al. "Modeling the Time—Varying Subjective Quality of HTTP Video Streams With Rate Adaptations". In: *IEEE Transactions on Image Processing* 23.5 (2014), pp. 2206–2221. DOI: 10.1109/TIP.2014.2312613.

[20] *Chromium Web Browser*. URL: https://www.chromium.org/.

[21] VNI Cisco. "Cisco visual networking index: Forecast and trends, 2017–2022". In: *White Paper* 1 (2018).

[22] T Cloonan and J Allen. "Competitive analysis of adaptive video streaming implementations". In: *SCTE Cable-Tec Expo Technical Workshop*. 2011.

[23] *CMAF Server*. URL: https://github.com/tomlyko/cmaf-server.

[24] Stefano D'Aronco, Laura Toni, and Pascal Frossard. "Price-Based Controller for Utility-Aware HTTP Adaptive Streaming". In: *IEEE MultiMedia* 24.2 (2017), pp. 20–29. DOI: 10.1109/MMUL.2017.41.

[25] *DASH.JS*. URL: https://github.com/Dash-Industry-Forum/dash.js/wiki.

[26] Luca De Cicco, Saverio Mascolo, and Vittorio Palmisano. "Feedback Control for Adaptive Live Video Streaming". In: *Proceedings of the Second Annual ACM Conference on Multimedia Systems*. MMSys '11. San Jose, CA, USA: Association for Computing Machinery, 2011, pp. 145–156. ISBN: 9781450305181. DOI: 10.1145/1943552.1943573.

[27] Luca De Cicco et al. “ELASTIC: A Client-Side Controller for Dynamic Adaptive Streaming over HTTP (DASH)”. In: *2013 20th International Packet Video Workshop*. 2013, pp. 1–8. DOI: 10.1109/PV.2013.6691442.

[28] Johan De Vriendt, Danny De Vleeschauwer, and David Robinson. “Model for estimating QoE of video delivered using HTTP adaptive streaming”. In: *2013 IFIP/IEEE International Symposium on Integrated Network Management (IM 2013)*. 2013, pp. 1288–1293.

[29] Steve E Deering. *RFC1112: Host Extensions for IP multicasting*. 1989.

[30] Andrea Detti, Bruno Ricci, and Nicola Blefari-Melazzi. “Tracker-assisted rate adaptation for MPEG DASH live streaming”. In: *IEEE INFOCOM 2016 - The 35th Annual IEEE International Conference on Computer Communications*. 2016, pp. 1–9. DOI: 10.1109/INFOCOM.2016.7524620.

[31] Florin Dobrian et al. “Understanding the impact of video quality on user engagement”. In: *ACM SIGCOMM computer communication review* 41.4 (2011), pp. 362–373.

[32] Z. Duanmu et al. “A Quality-of-Experience Index for Streaming Video”. In: *IEEE Journal of Selected Topics in Signal Processing* 11.1 (2017), pp. 154–166. DOI: 10.1109/JSTSP.2016.2608329.

[33] Zhengfang Duanmu, Kede Ma, and Zhou Wang. “Quality-of-Experience for Adaptive Streaming Videos: An Expectation Confirmation Theory Motivated Approach”. In: *IEEE Transactions on Image Processing* 27.12 (2018), pp. 6135–6146. DOI: 10.1109/TIP.2018.2855403.

[34] Hans Eriksson. “Mbone: The multicast backbone”. In: *Communications of the ACM* 37.8 (1994), pp. 54–60.

[35] A. E. Essaili, T. Lohmar, and M. Ibrahim. “Realization and Evaluation of an End-to-End Low Latency Live DASH System”. In: *2018 IEEE International Symposium on Broadband Multimedia Systems and Broadcasting (BMSB)*. 2018, pp. 1–5.

[36] Nagabhushan Eswara et al. “A Continuous QoE Evaluation Framework for Video Streaming Over HTTP”. In: *IEEE Transactions on Circuits and Systems for Video Technology* 28.11 (2018), pp. 3236–3250. DOI: 10.1109/TCSVT.2017.2742601.

[37] Arsham Farshad et al. “Leveraging SDN to provide an in-network QoE measurement framework”. In: *2015 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*. 2015, pp. 239–244. DOI: 10.1109/INFCOMW.2015.7179391.

[38] W Fenner. *RFC2236: Internet Group Management Protocol, Version 2*. 1997.

[39] R Fielding, Yves Lafon, and Julian Reschke. “Hypertext transfer protocol (HTTP/1.1): range requests”. In: *IETF RFC7233, June* (2014).

[40] Matteo Gadaleta et al. “D-DASH: A Deep Q-Learning Framework for DASH Video Streaming”. In: *IEEE Transactions on Cognitive Communications and Networking* 3.4 (2017), pp. 703–718. DOI: 10.1109/TCCN.2017.2755007.

[41] M. N. Garcia, W. Robitza, and A. Raake. “On the accuracy of short-term quality models for long-term quality prediction”. In: *2015 Seventh International Workshop on Quality of Multimedia Experience (QoMEX)*. 2015, pp. 1–6. DOI: 10.1109/QoMEX.2015.7148123.

[42] Panagiotis Georgopoulos et al. “Towards Network-Wide QoE Fairness Using Openflow-Assisted Adaptive Video Streaming”. In: *Proceedings of the 2013 ACM SIGCOMM Workshop on Future Human-Centric Multimedia Networking*. FhMN ’13. Hong Kong, China: Association for Computing Machinery, 2013, pp. 15–20. ISBN: 9781450321839. DOI: 10.1145/2491172.2491181.

[43] D. Ghadiyaram, J. Pan, and A. C. Bovik. “A Subjective and Objective Study of Stalling Events in Mobile Streaming Videos”. In: *IEEE Transactions on Circuits and Systems for Video Technology* 29.1 (2019), pp. 183–197.

[44] Audio-Video Transport Working Group et al. *RFC1889: RTP: A transport protocol for real-time applications*. 1996.

[45] Yang Guo et al. "P2Cast: peer-to-peer patching scheme for VoD service". In: *Proceedings of the 12th international conference on World Wide Web*. 2003, pp. 301–309.

[46] Yang Guo et al. "Ponder: providing commercial-quality video-on-demand service using peer-to-peer network". In: *Technical repot, corporate research, Thomson Inc* (2006).

[47] Zhili Guo, Yao Wang, and Xiaoqing Zhu. "Assessing the visual effect of non-periodic temporal variation of quantization stepsize in compressed video". In: *2015 IEEE International Conference on Image Processing (ICIP)*. 2015, pp. 3121–3125. DOI: 10.1109/ICIP.2015.7351378.

[48] Craig Gutterman et al. "Stallion: Video Adaptation Algorithm for Low-Latency Video Streaming". In: *Proceedings of the 11th ACM Multimedia Systems Conference*. MMSys '20. Istanbul, Turkey: Association for Computing Machinery, 2020, pp. 327–332. ISBN: 9781450368452. DOI: 10.1145/3339825.3397044.

[49] Bo Han et al. "MP-DASH: Adaptive Video Streaming Over Preference-Aware Multipath". In: *Proceedings of the 12th International on Conference on Emerging Networking EXperiments and Technologies*. CoNEXT '16. Irvine, California, USA: Association for Computing Machinery, 2016, pp. 129–143. ISBN: 9781450342926. DOI: 10.1145/2999572.2999606.

[50] Alain Horé and Djemel Ziou. "Image Quality Metrics: PSNR vs. SSIM". In: *2010 20th International Conference on Pattern Recognition*. 2010, pp. 2366–2369. DOI: 10.1109/ICPR.2010.579.

[51] Tobias Hoßfeld et al. "Assessing effect sizes of influence factors towards a QoE model for HTTP adaptive streaming". In: *2014 sixth international workshop on quality of multimedia experience (qomex)*. IEEE. 2014, pp. 111–116.

[52] Rémi Houdaille and Stéphane Gouache. "Shaping HTTP Adaptive Streams for a Better User Experience". In: *Proceedings of the 3rd Multimedia Systems Conference*. MMSys '12. Chapel Hill, North Carolina: Association for Computing Machinery, 2012, pp. 1–9. ISBN: 9781450311311. DOI: 10.1145/2155555.2155557.

[53] Te-Yuan Huang et al. "A Buffer-Based Approach to Rate Adaptation: Evidence from a Large Video Streaming Service". In: *Proceedings of the 2014 ACM Conference on SIGCOMM*. SIGCOMM '14. Chicago, Illinois, USA: Association for Computing Machinery, 2014, pp. 187–198. ISBN: 9781450328364. DOI: 10.1145/2619239.2626296.

[54] K Hughes and D Singer. "Information technology–Multimedia application format (MPEG-A)–Part 19: Common media application format (CMAF) for segmented media". In: *ISO/IEC* (2017), pp. 23000–19.

[55] Rafael Huysegems et al. "HTTP/2-Based Methods to Improve the Live Experience of Adaptive Streaming". In: *Proceedings of the 23rd ACM International Conference on Multimedia*. MM '15. Brisbane, Australia: Association for Computing Machinery, 2015, pp. 541–550. ISBN: 9781450334594. DOI: 10.1145/2733373.2806264.

[56] ITU-R. *BT.500-13: Methodology for the subjective assessment of the quality of television pictures*. 2012.

[57] ITU-T. *P.910: Subjective video quality assessment methods for multimedia applications*. 1999.

[58] ITU-T. *P.1070: Opinion model for video-telephony applications*. 2007.

[59] ITU-T. *P.564: Conformance testing for voice over IP transmission quality assessment models*. 2007.

[60] ITU-T. *P.1201: Parametric Non-intrusive Assessment of Audiovisual Media Streaming Quality*. 2012.

[61] ITU-T. *P.1202: Parametric non-intrusive bitstream assessment of video media streaming quality*. 2013.

[62] ITU-T. *1203.3: Parametric bitstream-based quality assessment of progressive download and adaptive audiovisual streaming services over reliable transport-Quality integration module*. 2017.

[63] *ITU-T Rec. P.1203 Standalone Implementation*. URL: `https://github.com/itu-p1203/itu-p1203`.

[64] V. Jacobson. "Congestion Avoidance and Control". In: *Symposium Proceedings on Communications Architectures and Protocols*. SIGCOMM '88. Stanford, California, USA: Association for Computing Machinery, 1988, pp. 314–329. ISBN: 0897912799. DOI: `10.1145/52324.52356`.

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

[66] Vinay Joseph and Gustavo de Veciana. "NOVA: QoE-driven optimization of DASH-based video delivery in networks". In: *IEEE INFOCOM 2014 - IEEE Conference on Computer Communications*. 2014, pp. 82–90. DOI: `10.1109/INFOCOM.2014.6847927`.

[67] Parikshit Juluri, Venkatesh Tamarapalli, and Deep Medhi. "Measurement of Quality of Experience of Video-on-Demand Services: A Survey". In: *IEEE Communications Surveys Tutorials* 18.1 (2016), pp. 401–418. DOI: `10.1109/COMST.2015.2401424`.

[68] Theo Karagkioules et al. "Online Learning for Low-Latency Adaptive Streaming". In: *Proceedings of the 11th ACM Multimedia Systems Conference*. MMSys '20. Istanbul, Turkey: Association for Computing Machinery, 2020, pp. 315–320. ISBN: 9781450368452. DOI: `10.1145/3339825.3397042`.

[69] T. Kohonen, M. R. Schroeder, and T. S. Huang. *Self-Organizing Maps*. 3rd. Berlin, Heidelberg: Springer-Verlag, 2001. ISBN: 3540679219.

[70] Dejan Kostić et al. "Bullet: High bandwidth data dissemination using an overlay mesh". In: *Proceedings of the nineteenth ACM symposium on Operating systems principles*. 2003, pp. 282–297.

[71] Vengatanathan Krishnamoorthi et al. "BUFFEST: Predicting Buffer Conditions and Real-Time Requirements of HTTP(S) Adaptive Streaming Clients". In: *Proceedings of the 8th ACM on Multimedia Systems Conference*. MMSys'17. Taipei, Taiwan: Association for Computing Machinery, 2017, pp. 76–87. ISBN: 9781450350020. DOI: 10.1145/3083187.3083193.

[72] Maxim Levkov. "Video encoding and transcoding recommendations for HTTP Dynamic Streaming on the Adobe® Flash® Platform". In: *White Paper, Adobe Systems Inc* (2010).

[73] Z. Li et al. "Probe and Adapt: Rate Adaptation for HTTP Video Streaming At Scale". In: *IEEE Journal on Selected Areas in Communications* 32.4 (Apr. 2014), pp. 719–733. ISSN: 0733-8716. DOI: 10.1109/JSAC.2014.140405.

[74] Zhi Li et al. "Streaming Video over HTTP with Consistent Quality". In: *Proceedings of the 5th ACM Multimedia Systems Conference*. MMSys '14. Singapore, Singapore: Association for Computing Machinery, 2014, pp. 248–258. ISBN: 9781450327053. DOI: 10.1145/2557642.2557658.

[75] Zhi Li et al. "VMAF: The journey continues". In: *Netflix Technology Blog* 25 (2018).

[76] Jan Lievens et al. "Perceptual video quality assessment in HTTP adaptive streaming". In: *2015 IEEE International Conference on Consumer Electronics (ICCE)*. 2015, pp. 72–73. DOI: 10.1109/ICCE.2015.7066323.

[77] May Lim et al. "When They Go High, We Go Low: Low-Latency Live Streaming in Dash.Js with LoL". In: *Proceedings of the 11th ACM Multimedia Systems Conference*. MMSys '20. Istanbul, Turkey: Association for Computing

Machinery, 2020, pp. 321–326. ISBN: 9781450368452. DOI: 10.1145/3339825.3397043.

[78] Chenghao Liu, Imed Bouazizi, and Moncef Gabbouj. “Rate Adaptation for Adaptive HTTP Streaming”. In: *Proceedings of the Second Annual ACM Conference on Multimedia Systems*. MMSys ’11. San Jose, CA, USA: Association for Computing Machinery, 2011, pp. 169–174. ISBN: 9781450305181. DOI: 10.1145/1943552.1943575.

[79] Chenghao Liu et al. “Rate adaptation for dynamic adaptive streaming over HTTP in content distribution network”. In: *Signal Processing: Image Communication* 27.4 (2012), pp. 288–311.

[80] T. Lohmar et al. “Dynamic adaptive HTTP streaming of live content”. In: *2011 IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks*. June 2011, pp. 1–8. DOI: 10.1109/WoWMoM.2011.5986186.

[81] T. Lyko et al. “Llama - Low Latency Adaptive Media Algorithm”. In: *2020 IEEE International Symposium on Multimedia (ISM)*. 2020, pp. 113–121. DOI: 10.1109/ISM.2020.00027.

[82] Tomasz Lyko et al. “Evaluation of CMAF in Live Streaming Scenarios”. In: *Proceedings of the 30th ACM Workshop on Network and Operating Systems Support for Digital Audio and Video*. NOSSDAV ’20. Istanbul, Turkey: Association for Computing Machinery, 2020, pp. 21–26. ISBN: 9781450379458. DOI: 10.1145/3386290.3396932.

[83] Tomasz Lyko et al. “Improving quality of experience in adaptive low latency live streaming”. In: *Multimedia Tools and Applications* (July 2023). ISSN: 1380-7501. DOI: 10.1007/s11042-023-15895-9.

[84] Nazanin Magharei and Reza Rejaie. “Prime: Peer-to-peer receiver-driven mesh-based streaming”. In: *IEEE/ACM transactions on networking* 17.4 (2009), pp. 1052–1065.

[85] Hongzi Mao, Ravi Netravali, and Mohammad Alizadeh. "Neural Adaptive Video Streaming with Pensieve". In: *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*. SIGCOMM '17. Los Angeles, CA, USA: Association for Computing Machinery, 2017, pp. 197–210. ISBN: 9781450346535. DOI: `10.1145/3098822.3098843`.

[86] Konstantin Miller, Abdel-Karim Al-Tamimi, and Adam Wolisz. "QoE-Based Low-Delay Live Streaming Using Throughput Predictions". In: *ACM Trans. Multimedia Comput. Commun. Appl.* 13.1 (Oct. 2016). ISSN: 1551-6857. DOI: `10.1145/2990505`.

[87] Konstantin Miller et al. "Adaptation algorithm for adaptive streaming over HTTP". In: *2012 19th International Packet Video Workshop (PV)*. 2012, pp. 173–178. DOI: `10.1109/PV.2012.6229732`.

[88] *Mininet*. URL: `http://www.mininet.org`.

[89] Volodymyr Mnih et al. "Asynchronous Methods for Deep Reinforcement Learning". In: *Proceedings of The 33rd International Conference on Machine Learning*. Ed. by Maria Florina Balcan and Kilian Q. Weinberger. Vol. 48. Proceedings of Machine Learning Research. New York, New York, USA: PMLR, 20–22 Jun 2016, pp. 1928–1937.

[90] R. K. P. Mok, E. W. W. Chan, and R. K. C. Chang. "Measuring the quality of experience of HTTP video streaming". In: *12th IFIP/IEEE International Symposium on Integrated Network Management (IM 2011) and Workshops*. 2011, pp. 485–492. DOI: `10.1109/INM.2011.5990550`.

[91] Ricky K. P. Mok et al. "QDASH: A QoE-Aware DASH System". In: *Proceedings of the 3rd Multimedia Systems Conference*. MMSys '12. Chapel Hill, North Carolina: Association for Computing Machinery, 2012, pp. 11–22. ISBN: 9781450311311. DOI: `10.1145/2155555.2155558`.

[92] Ricky K.P. Mok et al. "Inferring the QoE of HTTP Video Streaming from User-Viewing Activities". In: *Proceedings of the First ACM SIGCOMM Workshop on Measurements up the Stack*. W-MUST '11. Toronto, Ontario, Canada: Association for Computing Machinery, 2011, pp. 31–36. ISBN: 9781450308007. DOI: `10.1145/2018602.2018611`.

[93] Anush Krishna Moorthy and Alan Conrad Bovik. "A Two-Step Framework for Constructing Blind Image Quality Indices". In: *IEEE Signal Processing Letters* 17.5 (2010), pp. 513–516. DOI: `10.1109/LSP.2010.2043888`.

[94] *MP4Box*. URL: `https://gpac.wp.imt.fr/mp4box/`.

[95] Christopher Mueller et al. "Oscillation compensating Dynamic Adaptive Streaming over HTTP". In: *2015 IEEE International Conference on Multimedia and Expo (ICME)*. 2015, pp. 1–6. DOI: `10.1109/ICME.2015.7177435`.

[96] Christopher Müller, Stefan Lederer, and Christian Timmerer. "An evaluation of dynamic adaptive streaming over HTTP in vehicular environments". In: *Proceedings of the 4th Workshop on Mobile Video*. 2012, pp. 37–42.

[97] Hyunwoo Nam et al. "Towards QoE-aware video streaming using SDN". In: *2014 IEEE Global Communications Conference*. 2014, pp. 1317–1322. DOI: `10.1109/GLOCOM.2014.7036990`.

[98] *NS-3*. URL: `https://www.nsnam.org/`.

[99] *NS3-DASH-CMAF-Model*. URL: `https://github.com/tomlyko/ns3-dash-cmaf-model`.

[100] Harald Ott, Konstantin Miller, and Adam Wolisz. "Simulation Framework for HTTP-Based Adaptive Streaming Applications". In: *Proceedings of the Workshop on Ns-3*. WNS3 '17. Porto, Portugal: ACM, 2017, pp. 95–102. ISBN: 978-1-4503-5219-2. DOI: `10.1145/3067665.3067675`.

[101] Ihsan Mert Ozcelik and Cem Ersoy. "Low-Latency Live Streaming Over HTTP in Bandwidth-Limited Networks". In: *IEEE Communications Letters* 25.2 (2021), pp. 450–454. DOI: `10.1109/LCOMM.2020.3030887`.

[102] Stefano Petrangeli et al. "A multi-agent Q-Learning-based framework for achieving fairness in HTTP Adaptive Streaming". In: *2014 IEEE Network Operations and Management Symposium (NOMS)*. 2014, pp. 1–9. DOI: `10.1109/NOMS.2014.6838245`.

[103] Stefano Petrangeli et al. "QoE-Driven Rate Adaptation Heuristic for Fair Adaptive Video Streaming". In: *ACM Trans. Multimedia Comput. Commun. Appl.* 12.2 (Oct. 2015). ISSN: 1551-6857. DOI: `10.1145/2818361`.

[104] Stefano Petrangeli et al. "Software-defined network-based prioritization to avoid video freezes in HTTP adaptive streaming". In: *International Journal of Network Management* 26.4 (2016), pp. 248–268.

[105] Alexander Raake et al. "A bitstream-based, scalable video-quality model for HTTP adaptive streaming: ITU-T P.1203.1". In: *Ninth International Conference on Quality of Multimedia Experience (QoMEX)*. Erfurt: IEEE, May 2017. ISBN: 978-1-5386-4024-1. DOI: `10.1109/QoMEX.2017.7965631`.

[106] Benjamin Rainer et al. "A seamless Web integration of adaptive HTTP streaming". In: *2012 Proceedings of the 20th European Signal Processing Conference (EUSIPCO)*. IEEE. 2012, pp. 1519–1523.

[107] Werner Robitza, Marie-Neige Garcia, and Alexander Raake. "A modular HTTP adaptive streaming QoE model — Candidate for ITU-T P.1203 ("P.NATS")". In: *2017 Ninth International Conference on Quality of Multimedia Experience (QoMEX)*. 2017, pp. 1–6. DOI: `10.1109/QoMEX.2017.7965689`.

[108] Werner Robitza et al. "HTTP Adaptive Streaming QoE Estimation with ITU-T Rec. P.1203 – Open Databases and Software". In: *9th ACM Multimedia Systems Conference*. Amsterdam, 2018. ISBN: 9781450351928. DOI: `10.1145/3204949.3208124`.

[109] Demostenes Z. Rodriguez et al. "Quality metric to assess video streaming service over TCP considering temporal location of pauses". In: *IEEE Transactions on*

Consumer Electronics 58.3 (2012), pp. 985–992. DOI: 10.1109/TCE.2012.6311346.

[110] Michele A. Saad, Alan C. Bovik, and Christophe Charrier. “Blind Image Quality Assessment: A Natural Scene Statistics Approach in the DCT Domain”. In: *IEEE Transactions on Image Processing* 21.8 (2012), pp. 3339–3352. DOI: 10.1109/TIP.2012.2191563.

[111] H Schulzrinne et al. “Real-time streaming protocol version 2.0”. In: *IETF, December* (2016).

[112] Henning Schulzrinne, Anup Rao, and Robert Lanphier. *RFC 2326: Real time streaming protocol (RTSP)*. 1998.

[113] Henning Schulzrinne et al. *RFC3550: RTP: A transport protocol for real-time applications*. 2003.

[114] Kalpana Seshadrinathan and Alan Bovik. “Temporal hysteresis model of time varying subjective video quality”. In: June 2011, pp. 1153–1156. DOI: 10.1109/ICASSP.2011.5946613.

[115] Hemanth P Sethuram et al. “Streaming Video QoE Modeling and Prediction: A Long Short-Term Memory Approach”. In: *arXiv preprint arXiv:1807.07126* (2018).

[116] Yun Shen et al. “A method of QoE evaluation for adaptive streaming based on bitrate distribution”. In: *2014 IEEE International Conference on Communications Workshops (ICC)*. 2014, pp. 551–556. DOI: 10.1109/ICCW.2014.6881256.

[117] Kamal Deep Singh, Yassine Hadjadj-Aoul, and Gerardo Rubino. “Quality of experience estimation for adaptive HTTP/TCP video streaming using H.264/AVC”. In: *2012 IEEE Consumer Communications and Networking Conference (CCNC)*. 2012, pp. 127–131. DOI: 10.1109/CCNC.2012.6181070.

[118] Ashkan Sobhani, Abdulsalam Yassine, and Shervin Shirmohammadi. "A Video Bitrate Adaptation and Prediction Mechanism for HTTP Adaptive Streaming". In: *ACM Trans. Multimedia Comput. Commun. Appl.* 13.2 (Mar. 2017). ISSN: 1551-6857. DOI: 10.1145/3052822.

[119] Iraj Sodagar. "The MPEG-DASH Standard for Multimedia Streaming Over the Internet". In: *IEEE MultiMedia* 18.4 (2011), pp. 62–67. DOI: 10.1109/MMUL.2011.71.

[120] K. Spiteri, R. Urgaonkar, and R. K. Sitaraman. "BOLA: Near-optimal bitrate adaptation for online videos". In: *IEEE INFOCOM 2016 - The 35th Annual IEEE International Conference on Computer Communications*. Apr. 2016, pp. 1–9. DOI: 10.1109/INFOCOM.2016.7524428.

[121] Nicolas Staelens et al. "Subjective Quality Assessment of Longer Duration Video Sequences Delivered Over HTTP Adaptive Streaming to Tablet Devices". In: *IEEE Transactions on Broadcasting* 60.4 (2014), pp. 707–714. DOI: 10.1109/TBC.2014.2359255.

[122] V. Swaminathan and S. Wei. "Low latency live video streaming using HTTP chunked encoding". In: *2011 IEEE 13th International Workshop on Multimedia Signal Processing*. 2011, pp. 1–6.

[123] Akira Takahashi, David Hands, and Vincent Barriac. "Standardization activities in the ITU for a QoE assessment of IPTV". In: *IEEE Communications Magazine* 46.2 (2008), pp. 78–84. DOI: 10.1109/MCOM.2008.4473087.

[124] S. Tavakoli et al. "Perceptual Quality of HTTP Adaptive Streaming Strategies: Cross-Experimental Analysis of Multi-Laboratory and Crowdsourced Subjective Studies". In: *IEEE Journal on Selected Areas in Communications* 34.8 (2016), pp. 2141–2153. DOI: 10.1109/JSAC.2016.2577361.

[125] *TC Module*. URL: http://manpages.ubuntu.com/manpages/xenial/man8/tc.8.html.

[126] Emmanuel Thomas et al. “Applications and deployments of server and network assisted DASH (SAND)”. English. In: *IET Conference Proceedings* (Jan. 2016), 22 (8 .)–22 (8 .)(1). URL: https://digital-library.theiet.org/content/conferences/10.1049/ibc.2016.0022.

[127] Emmanuel Thomas et al. “Enhancing MPEG DASH Performance via Server and Network Assistance”. In: *SMPTE Motion Imaging Journal* 126.1 (2017), pp. 22–27. DOI: 10.5594/JMI.2016.2632338.

[128] *Throughput Traces derived from CDN logs of BT Sport 1 service*. URL: https://github.com/lancs-net/ABR-Throughput-Traces.

[129] H. T. T. Tran et al. “A Multi-Factor QoE Model for Adaptive Streaming over Mobile Networks”. In: *2016 IEEE Globecom Workshops (GC Wkshps)*. 2016, pp. 1–6. DOI: 10.1109/GLOCOMW.2016.7848818.

[130] H. T. T. Tran et al. “A novel quality model for HTTP adaptive streaming”. In: *2016 IEEE Sixth International Conference on Communications and Electronics (ICCE)*. 2016, pp. 423–428. DOI: 10.1109/CCE.2016.7562674.

[131] Jeroen Van Der Hooft et al. “An HTTP/2 push-based approach for low-latency live streaming with super-short segments”. In: *Journal of Network and Systems Management* 26.1 (2018), pp. 51–78.

[132] R. Viola et al. “QoE-based enhancements of Chunked CMAF over low latency video streams”. In: *2019 IEEE International Symposium on Broadband Multimedia Systems and Broadcasting (BMSB)*. 2019, pp. 1–6.

[133] Aggelos Vlavianos, Marios Iliofotou, and Michalis Faloutsos. “BiToS: Enhancing BitTorrent for supporting streaming applications”. In: *Proceedings IEEE INFOCOM 2006. 25TH IEEE International Conference on Computer Communications*. IEEE. 2006, pp. 1–6.

[134] Fei Wang et al. “HAS QoE prediction based on dynamic video features with data mining in LTE network”. In: *Science China Information Sciences* 60.4 (2017), p. 042404.

[135] Qing Wang et al. “GENI Cinema: An SDN-Assisted Scalable Live Video Streaming Service”. In: *2014 IEEE 22nd International Conference on Network Protocols*. 2014, pp. 529–532. DOI: 10.1109/ICNP.2014.84.

[136] Sheng Wei and Viswanathan Swaminathan. “Low Latency Live Video Streaming over HTTP 2.0”. In: *Proceedings of Network and Operating System Support on Digital Audio and Video Workshop*. NOSSDAV ’14. Singapore, Singapore: Association for Computing Machinery, 2014, pp. 37–42. ISBN: 9781450327060. DOI: 10.1145/2597176.2578277.

[137] Dapeng Wu et al. “Streaming video over the Internet: approaches and directions”. In: *IEEE Transactions on circuits and systems for video technology* 11.3 (2001), pp. 282–300.

[138] *x264*. URL: https://www.videolan.org/developers/x264.html.

[139] Mengbai Xiao et al. “Evaluating and Improving Push Based Video Streaming with HTTP/2”. In: *Proceedings of the 26th International Workshop on Network and Operating Systems Support for Digital Audio and Video*. NOSSDAV ’16. Klagenfurt, Austria: Association for Computing Machinery, 2016. ISBN: 9781450343565. DOI: 10.1145/2910642.2910652.

[140] Zhimin Xu, Xinggong Zhang, and Zongming Guo. “QoE-Driven Adaptive K-Push for HTTP/2 Live Streaming”. In: *IEEE Transactions on Circuits and Systems for Video Technology* 29.6 (2019), pp. 1781–1794. DOI: 10.1109/TCSVT.2018.2849015.

[141] Jingteng Xue et al. “Assessing quality of experience for adaptive HTTP video streaming”. In: *2014 IEEE International Conference on Multimedia and Expo Workshops (ICMEW)*. 2014, pp. 1–6. DOI: 10.1109/ICMEW.2014.6890604.

[142] Praveen Kumar Yadav et al. “Playing Chunk-Transferred DASH Segments at Low Latency with QLive”. In: *Proceedings of the 12th ACM Multimedia Systems Conference*. MMSys ’21. Istanbul, Turkey: Association for Computing

Machinery, 2021, pp. 51–64. ISBN: 9781450384346. DOI: 10.1145/3458305.3463376.

[143] K. Yamagishi and T. Hayashi. “Parametric Quality-Estimation Model for Adaptive-Bitrate-Streaming Services”. In: *IEEE Transactions on Multimedia* 19.7 (2017), pp. 1545–1557. DOI: 10.1109/TMM.2017.2669859.

[144] Zhisheng Yan, Jingteng Xue, and Chang Wen Chen. “Prius: Hybrid Edge Cloud and Client Adaptation for HTTP Adaptive Streaming in Cellular Networks”. In: *IEEE Transactions on Circuits and Systems for Video Technology* 27.1 (2017), pp. 209–222. DOI: 10.1109/TCSVT.2016.2539827.

[145] Fuzheng Yang and Shuai Wan. “Bitstream-based quality assessment for networked video: a review”. In: *IEEE Communications Magazine* 50.11 (2012), pp. 203–209. DOI: 10.1109/MCOM.2012.6353702.

[146] Xiaoqi Yin et al. “A Control-Theoretic Approach for Dynamic Adaptive Video Streaming over HTTP”. In: *SIGCOMM Comput. Commun. Rev.* 45.4 (Aug. 2015), pp. 325–338. ISSN: 0146-4833. DOI: 10.1145/2829988.2787486.

[147] Ahmed H. Zahran et al. “SAP: Stall-Aware Pacing for Improved DASH Video Experience in Cellular Networks”. In: *Proceedings of the 8th ACM on Multimedia Systems Conference*. MMSys’17. Taipei, Taiwan: Association for Computing Machinery, 2017, pp. 13–26. ISBN: 9781450350020. DOI: 10.1145/3083187.3083199.

[148] A Zambelli. “MS-SSTR: Microsoft Smooth Streaming protocol technical report”. In: *Microsoft Corp., Redmond, WA, USA, Jun* (2010).

[149] D. Zegarra Rodríguez et al. “Video Quality Metric for Streaming Service Using DASH Standard”. In: *IEEE Transactions on Broadcasting* 62.3 (2016), pp. 628–639. DOI: 10.1109/TBC.2016.2570012.

[150] X Zhang et al. “BDONet/coolStreaming: A data-driven overlay network for live media streaming,[in Proc”. In: *IEEE Infocom, Miami, FL*. 2005.