

Quality Improvement for HTTP Adaptive Streaming over Mobile Networks

HUNG THAI LE

A DISSERTATION

SUBMITTED IN FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF DOCTOR OF PHILOSOPHY

IN COMPUTER SCIENCE AND ENGINEERING

Graduate Department of Computer and Information Systems

The University of Aizu

2017



© Copyright by Hung Thai Le
All Rights Reserved.

The thesis titled

Quality Improvement for HTTP Adaptive Streaming over Mobile Networks



by

Hung Thai Le


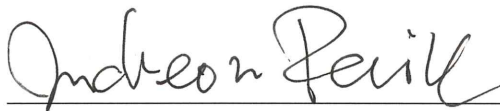
is reviewed and approved by:

Chief referee

Senior Associate Professor
Truong Cong Thang




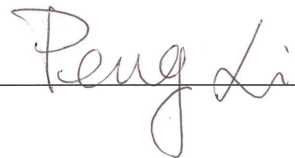
Professor
Incheon Paik



Professor
Anh T. Pham



Associate Professor
Peng Li



The University of Aizu

2017

TO MY PARENTS

Contents

LIST OF FIGURES	vi
LIST OF TABLES	vii
LIST OF ABBREVIATION	ix
ABSTRACT	1
1 INTRODUCTION	3
1.1 Motivations	5
1.2 Limitations	8
1.3 Main contributions	9
1.4 Thesis outline	11
2 BACKGROUND	13
2.1 Internet-based video streaming	13
2.1.1 History of Internet-based video streaming	14
2.1.2 Internet-based video streaming landscape	17
2.1.3 Evolution from datagram streaming to HAS	21
2.2 HAS principles	24
2.2.1 Video segments and the metadata	24
2.2.2 Video delivery	26
2.2.3 HAS client	27
2.3 Quality influence factors for HAS	30
2.3.1 Initial delay	30
2.3.2 Interruptions	31
2.3.3 Perceptual quality	31
2.3.4 Live latency	32
2.3.5 Maximizing video quality	33
2.4 Summary	34
3 ADAPTIVE STREAMING OF VARIABLE BITRATE VIDEOS	35
3.1 Related work	36
3.1.1 Adaptation method overview	36
3.1.2 Adaptation methods for VBR video streaming	38
3.2 Adaptation solution	40
3.2.1 Handling throughput and bitrate fluctuations	40

3.2.2	Adaptation algorithm	42
3.3	Experiments and Discussions	45
3.3.1	Experiment setup	45
3.3.2	Simple bandwidth scenario	47
3.3.3	Complex bandwidth scenario	48
3.3.4	Discussions	53
3.4	Summary	55
4	ADAPTIVE STREAMING OVER HTTP/2	57
4.1	Related work	58
4.1.1	HTTP/2 and Server push feature	59
4.1.2	HTTP/2-based adaptation methods	59
4.2	Problem description	60
4.2.1	Adaptation space	60
4.2.2	Buffer level estimation and Adaptation problem	62
4.3	Adaptation solution	63
4.3.1	General adaptation process	63
4.3.2	Rate decrease case ($R_i > T_i$)	65
4.3.3	Rate increase case ($R_i \leq T_i$)	66
4.4	Experiments and Discussions	67
4.4.1	Experiment settings	67
4.4.2	Simple bandwidth scenario	69
4.4.3	Complex bandwidth scenario	71
4.4.4	Discussions	74
4.5	Summary	75
5	HTTP LIVE STREAMING WITH PERCEPTUAL QUALITY CONTROL	77
5.1	Related work	78
5.1.1	Adaptation methods for HTTP live streaming	78
5.1.2	Future buffer based approach	79
5.1.3	JND metric for Adaptive Streaming	81
5.2	Adaptation solution	83
5.2.1	Overview of proposed algorithm	83
5.2.2	JND-driven criteria for Path finding	85
5.3	Experiments and Discussions	87
5.3.1	Experiment settings	87
5.3.2	Obtained results	88
5.4	Summary	91
6	LOW-DELAY LIVE STREAMING OVER HTTP	93
6.1	Problem description	94
6.2	Adaptation solution	95
6.3	Experiments and Discussions	99
6.3.1	Experiment settings	99
6.3.2	Obtained results	100
6.4	Summary	103

7	CONCLUSIONS AND FUTURE WORK	105
	ACKNOWLEDGMENT	108
	REFERENCES	122

This page intentionally left blank.

Listing of figures

1.1	Peak period traffic composition on North America fixed access networks. . . .	4
1.2	Different kinds of devices with different connections connect to a streaming server.	5
1.3	The general HAS architecture considered in the thesis.	7
1.4	The scope of our studies.	8
2.1	The instant bitrate and instant quality of a video encoded in (a) a variable rate and (b) a constant rate. Two encoded videos have the same average video bitrate. Here, the instant quality is measured in Mean Opinion Score (MOS). .	19
2.2	The evolution from datagram streaming to HTTP adaptive streaming. . . .	21
2.3	Principle of HTTP adaptive streaming in a typical system. Here, MPD refers to Media Presentation Description (or metadata).	25
2.4	Media delivery in HTTP adaptive streaming.	27
2.5	Block diagram of an HAS client.	28
2.6	Taxonomy of quality influence factors for HAS.	30
3.1	Bitrates of the versions of two test videos.	39
3.2	Test-bed organization for experiments.	46
3.3	Adaptation results of the three methods in simple bandwidth scenario. . . .	48
3.4	The bandwidth trace used in the complex bandwidth scenario.	49
3.5	Adaptation results of the three methods in the complex bandwidth scenario with “Sony Demo” video.	50
3.6	Cumulative distribution functions (CDF) of buffer level in the complex bandwidth scenario.	51
3.7	Adaptation results of the three methods in the complex bandwidth scenario with “Terminator 2” video.	53
4.1	Adaptation results of the three methods in simple bandwidth scenario. . . .	70
4.2	Adaptation results of the three methods in complex bandwidth scenario. . . .	72
5.1	Illustration of arrival curve, playout curve and trellis representation.	80
5.2	The relationship between perceptual distortion and normalized bitrate. . . .	82
5.3	Adaptation results of three adaptation methods in the complex scenario. . . .	89
5.4	Results of perceptual distortion and distortion change of three adaptation methods in the complex scenario.	90
6.1	Illustration of the request times and buffer behavior of the client at segments $n - 1$, n , and $n + 1$	97

6.2	Illustration of (a) the bandwidth trace and (b) the CDF of variable X , obtained by the observation process.	98
6.3	Test-bed organization for experiments.	99
6.4	The bandwidth trace and adaptation results of (a) the CTB method, (b) the ITB method, and (c) the PB method.	101
6.5	The bandwidth trace of a mobile network.	101
6.6	Instant value of the margin for $\varepsilon = 0.35, 0.25, 0.15$	102

List of Tables

2.1	Video streaming technologies.	15
2.2	Internet-based video streaming landscape. Control location refers to the location of adaptation engine (if any) that decides which and when media parts are transferred.	17
2.3	Representation rates for 2-second dataset.	26
3.1	Notations and definitions.	41
3.2	Version information of the two test videos.	46
3.3	Statistics of the reference methods and the proposed method with three different options for “Sony Demo”. Here STD, is the standard deviation.	51
3.4	Statistics of the reference methods and the proposed method with three different options for “Terminator 2”. Here, STD is the standard deviation.	52
4.1	Notations and definitions used in this chapter.	61
4.2	Statistics of adaptation results in complex bandwidth scenario.	73
4.3	Statistics of adaptation results in complex bandwidth scenario with a segment duration of 500ms.	74
5.1	Notations and definitions.	83
5.2	Statistics of different adaptation methods. Except the number of switches and statistics of buffer level, the unit of other parameters is JND unit.	91
6.1	Average statistics of the adaptation methods.	102

This page intentionally left blank.

List of abbreviation

ACK	Acknowledgment
CBR	Constant Bit Rate
CDN	Content Delivery Networks
CDF	Cumulative Distribution Function
CPU	Central Procession Unit
DASH	Dynamic Adaptive Streaming over HTTP
HAS	HTTP Adaptive Streaming
HD	High Definition
HDS	HTTP Dynamic Streaming developed by Adobe Systems
HLS	HTTP Live Streaming developed by Apple
HTTP	HyperText Transfer Protocol
IETF	Internet Engineer Task Force
IP	Internet Protocol
ITU	International Telecommunication Union
JND	Just Noticeable Difference
LTE	Long-Term Evolution
MOS	Mean Opinion Score
MSS	Microsoft Silverlight Smooth Streaming developed by Microsoft
MPD	Media Presentation Description
MPEG	Moving Picture Expert Group
NAT	Network Address Translation
OTT	Over The Top
PSNR	Peak Signal-to-Noise Ratio
P2P	Point-to-Point
QoE	Quality of Experience
QoS	Quality of Service
TCP	Transmission Control Protocol
RTCP	Real-Time Control Protocol
RTP	Real-time Transport Protocol
RTSP	Real-Time Streaming Protocol
RTT	Round-Trip Time
SAND	Server and Network Assisted DASH
VBR	Variable Bit Rate
VoD	Video on Demand
VR	Virtual Reality
XML	Extensible Markup Language
UDP	User Datagram Protocol

This page intentionally left blank.

Quality Improvement for HTTP Adaptive Streaming over Mobile Networks

ABSTRACT

Ever since the early years of video streaming, there has been a growing demand for high-quality video content. Thanks to today's wireless broadband connections and the pervasiveness of high-performance mobile devices, video streaming has become a major Internet service. However, due to the best-effort nature of the Internet, providing users with high video quality is challenging. As a result, all new streaming standards developed since 2008 are based on Adaptive Streaming technology which allows the client/server to adapt the video bitrate to network fluctuations in order to enable smoother viewing experience with no playback interruption.

Over the past few years, HTTP adaptive streaming (HAS) has emerged as a *de facto* standard for streaming videos over the Internet. To adapt to network fluctuations, a streaming provider should generate multiple versions (with different video bitrates, for example) of an original video, each of which is chopped into short segments. During a streaming session, an adaptation method located at the client is responsible for deciding which video bitrate should be requested for each segment to maintain a good quality of service.

Despite the ongoing efforts to guarantee high video quality, recent HAS-related studies claim that the adaptation challenge has not yet been successfully resolved, especially for HAS over mobile networks where the connection throughput is naturally time-varying. Motivated by the untapped potential of HAS technology and the fast growth of mobile video streaming in recent years, I aim to develop adaptation methods that effectively cope with throughput variations of a mobile connection to improve user experience. The contributions to this area of research are outlined in the following.

First, we observed that existing adaptation methods in HAS mostly focus on CBR (constant bitrate) video streaming. However, compared to the CBR encoding, the VBR (variable

bitrate) one has advantages in terms of video quality. Thus, in my first contribution, I present an adaptation method for HAS of VBR videos. To deal with variations of the video bitrate, a local average video bitrate is used as the representative bitrate of a video version. A buffer-based algorithm is then proposed to conservatively adapt video quality. Through experiments in the mobile streaming context, we show that our method can provide high video quality even under strong variations of connection throughput and video bitrate.

The recently ratified HTTP/2 protocol provides Server Push feature that helps reduce request-related overheads (e.g., in terms of energy, processing, bandwidth) for clients, servers, as well as network nodes. In my second contribution, I develop an adaptation method for HTTP/2-based adaptive streaming that not only leverages the Server Push feature of HTTP/2 but also provides buffer stability and gradual quality transitions. Since the method avoids playback interruptions and enables smoother viewing experience, it enhances the overall video quality.

In my third contribution, I focus on a particularly challenging use case of live streaming. Although the majority of the video content streamed over the Internet is video-on-demand (VoD), the amount of live streaming is growing rapidly. In live streaming, the client's buffer is just 10 seconds or less (to enable a small capture-to-display delay); thus, this small buffer could be depleted easily due to strong throughput variations. To maintain seamless streaming, the client may switch to a very low video version, leading to a drastic quality change. We develop a quality-driven adaptation method to meet the tradeoff between the requirements of buffer stability and smooth quality transitions. Our experimental results show that the proposed method can provide high video quality under a small buffer size of 10 seconds.

The low-delay live streaming is a special case of live streaming, where the buffer size is limited to just few seconds. For specific services such as video surveillance, low delay is considered a mandatory requirement, but it is very challenging to guarantee seamless streaming in condition of low delay. In my final contribution, I present a probabilistic approach to adaptively decide the video bitrate by taking into account the instant buffer level. The experimental results with a buffer size of 4 seconds show that compared to traditional methods, the proposed one can significantly reduce the number of buffer underflows while providing similar video bitrates.

1

Introduction



video is just a series of images shown quickly one after another on a screen to create an illusion of continuity. However, it took almost 30 years after the first TV receiver was commercially available (i.e., in the late 1920s) for television to achieve a further step by enabling audiences to watch video content at their own home. Also, video content providers offered users “live” transmission, that is, watching the event as it happens. However, due to the limitations of broadcasting technologies in the middle of the 20th century, users had few choices of selecting a broadcast station. Besides, the time shifting, an important feature allowing users to watch the content at different times of the day, was unavailable.

After 1969, the ARPANET (Advanced Research Projects Agency Network [1]) and then

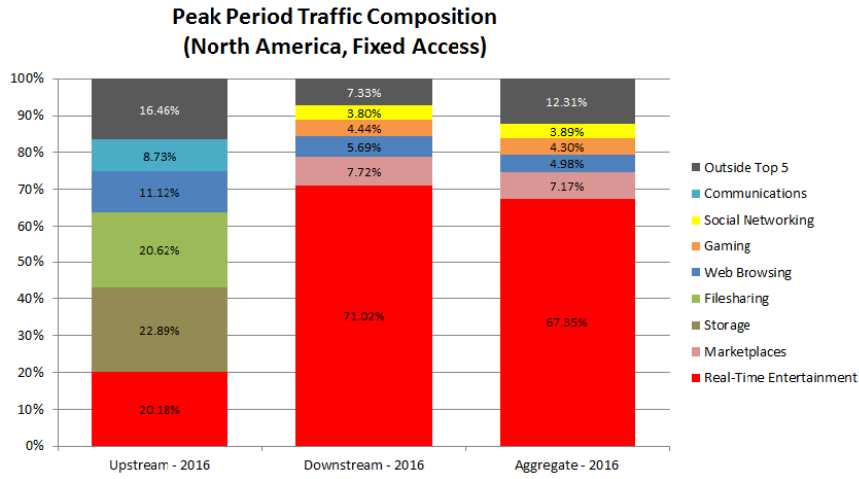


Figure 1.1: Peak period traffic composition on North America fixed access networks [5].

the Internet arrived. Users were offered “on-demand” streaming services. Yet, Internet-based streaming only really became popular in the 1990s. Two key factors responsible for that fact are the large computing and the high bandwidth requirement of video streaming. To stream a video over the Internet, each image must be received by the user’s device before the predefined time in order to be ready for playout. Because a lot of images per second (i.e., at least 24 images per second in high quality services) is required to create illusions and the amount of bytes for each image is significant, the bandwidth requirement to stream a video was too high compared to the bandwidth availability in the early day of the Internet. Compression, therefore, is necessary to reduce the amount of data exchange, but a higher compression obviously requires more complex computing. In the middle of the 1990s, both the computational and bandwidth requirements were achieved, leading a number of successful commercial streaming applications such as QuickTime [2], ActiveMovie [3], and RealPlayer [4].

Today, video streaming has become a major Internet service. Youtube alone generates billions of views for over a billion global users every day [6]. Sandvine in its 2016 Global Internet Phenomena Report predicted that real time entertainment services (i.e., streaming audios and videos) account for 80% of North American downstream in peak hours on fixed access networks in 2020, up from 71% in 2016 (see Fig. 1.1) [5]. Many major sport events such as Olympic Games and FIFA Worldcup are live-streamed successfully over the Internet with only a few sec-

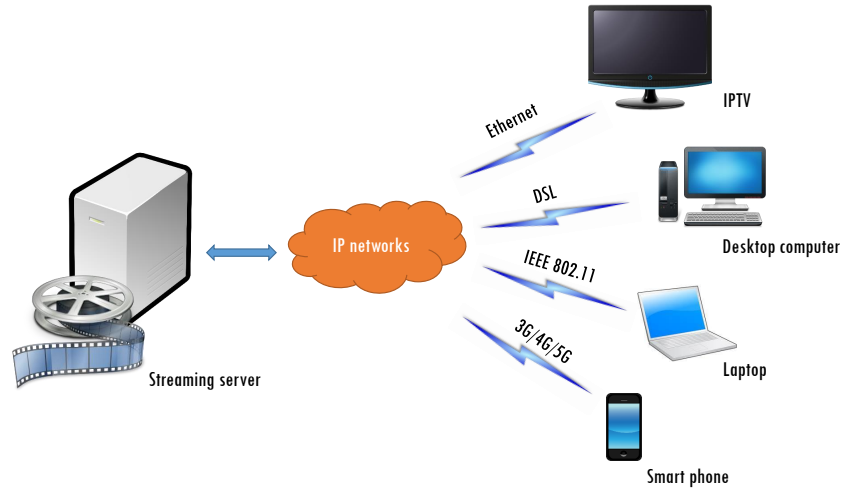


Figure 1.2: Different kinds of devices with different connections connect to a streaming server.

onds delay. It was reported that 8.5 million people watched the League of Legends Season 3 eSport finals at the same time via Twitch livestream [7].

Broadband connections and high-performance mobile devices produced a new mindset: *watch what I want, at any time, and at any location*. As illustrated in Fig. 1.2, a user can use many kinds of devices to access to a vast number of videos via different connections (with different Internet access rates). Youtube recently revealed that more than 50% of YouTube views in 2016 came from mobile devices [6]. Note that five years ago, the mobile streaming traffic accounted only for less than 20%.

1.1 Motivations

Although there are more and more Internet-based streaming applications, the Internet as a best-effort network is not designed to stream video content. The main challenge of streaming videos over the Internet is the throughput fluctuation caused by today’s heterogeneous networks. Due to throughput fluctuations, it is impossible to delivery videos with a fixed video bitrate. Hence, all new streaming standards developed since 2008 are based on *Adaptive streaming* technology [8]. The key feature of this technology is the ability to change the video bitrate according to the instant throughput and to the capability of the device used to play back the video (e.g., screen resolution, current buffer level, etc.).

On one hand, adaptive streaming allows clients with different Internet access rates and different capabilities to stream videos with different video bitrates. A smart phone with a low-speed 3G connection can receive a lower video bitrate (a lower quality) while a desktop computer with a fiber connection can receive a higher video bitrate (a higher quality). On the other hand, adaptive streaming enables a client to adapt to the connection throughput if the throughput varies during a streaming session. It is very important for mobile users, who often observe throughput fluctuations, since this technology reduces significantly the number of playout interruptions and so enhances the quality of service.

Over the past few years, *HTTP adaptive streaming (HAS)* has emerged as a *de facto* standard for streaming videos over the Internet [9, 10]. Compared to the other approaches using Real-time Transport Protocol (RTP)/Real-Time Control Protocol (RTCP) [11] or Real-Time Streaming Protocol (RTSP) [12], the use of HyperText Transfer Protocol (HTTP) [13, 14] offers several advantages. The most important benefit is that HAS is very cost effective. By using HTTP, a streaming provider is able to reduce cost throughout maintaining standard Web servers rather than licensed (and expensive) media control servers. Using HTTP, HAS takes advantage of a vast delivery infrastructure that was originally created to Web traffic. Additionally, as HTTP is firewall-friendly, media packets can traverse firewall and NAT devices easily. Due to these advantages, HAS has been adopted by major streaming applications today - including Youtube¹, Netflix², and Amazon Instant³. Since 2012, a global standard called Dynamic Adaptive Streaming over HTTP (DASH) has been being developed by Moving Picture Experts Group (MPEG) to enable the interoperability in the industry [9, 15]. The second edition of DASH was released in 2014 [16]. At the time of writing, the MPEG-DASH experts are working towards to the third edition of this standard.

HAS makes it possible to adapt the video bitrate according to the connection throughput, but no standards for HAS specify how to do this. At the start of this study, there was a few studies available on the subject [17]. Moreover, compared to fixed networks, mobile networks

¹<https://www.youtube.com>

²<https://www.netflix.com>

³owned and operated by Amazon.com

pose additional challenges to video streaming services due to limited cellular converge, high latency, and the variety of today's mobile devices. Video traffic, however, accounted for more than half of all mobile data traffic and is expected to reach 75% in 2020 [18]. Therefore, a new adaptation method optimized for mobile clients has received a lot of attention from both academia and industry. Motivated by the fast growth of mobile video traffic and the untapped potential in HAS technology, we aim to develop adaptation methods that effectively cope with throughput variations of a mobile connection to improve user experience.

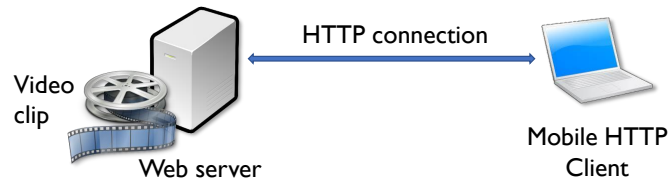


Figure 1.3: The general HAS architecture considered in the thesis.

Figure 1.3 shows the general architecture of HAS in mobile networks. A mobile HTTP client accesses to a video stored at an HTTP streaming server (i.e., a standard Web server) via an HTTP connection. In this thesis, we focus on the problems of (i) variable bitrate (VBR) video streaming, (ii) HTTP/2-based video streaming, and (iii) live streaming. The three problems are at the server, the transport protocol (i.e., HTTP), and the client, respectively. The detailed motivations are as follows:

1. *Adaptive streaming of VBR videos:* A video stored in the streaming server can be encoded using constant bitrate (CBR) encoding or variable bitrate (VBR) encoding. As the video bitrate of a CBR video is predictable, most existing adaptation methods focused on streaming CBR videos. However, compared to CBR encoding, VBR one has an advantage in terms of video quality [19]. Therefore, we, in this thesis, are interested in adaptive streaming of VBR videos.
2. *Adaptive streaming over HTTP/2:* Since 2015, the new major revision of HTTP (i.e., HTTP/2*) has been standardized with some prominent features that effectively support

*Hereafter, we use two terms HTTP and HTTP/2 to indicate HTTP version 1.1 and HTTP version 2, respectively.

video streaming [20]. It was expected that HTTP/2 will become a major streaming protocol in the near future. This motivated us to improve video quality for adaptive streaming over HTTP/2, leveraging new HTTP/2 features.

3. *HTTP live streaming*: Live streaming refers to video streaming with a low latency (i.e., a end-to-end delay of 10 seconds or less), which is important for specific services (e.g., video surveillance, sports live streaming). To reduce the latency in a live streaming service, the client buffer size (in time units) should be minimized. However, a such small buffer could be easily depleted due to strong throughput variations. Since most adaptation methods cannot effectively cope with throughput variations when the buffer is small, we develop adaptation methods towards to high-quality live streaming with a very low latency.

The work is actually an extension of our previous work [21], where the major context of *HTTP/1.1-based on-demand adaptive streaming of Constant Bit Rate (CBR) videos* is investigated. In this thesis, we further explore the three other contexts as shown in Fig. 1.4.

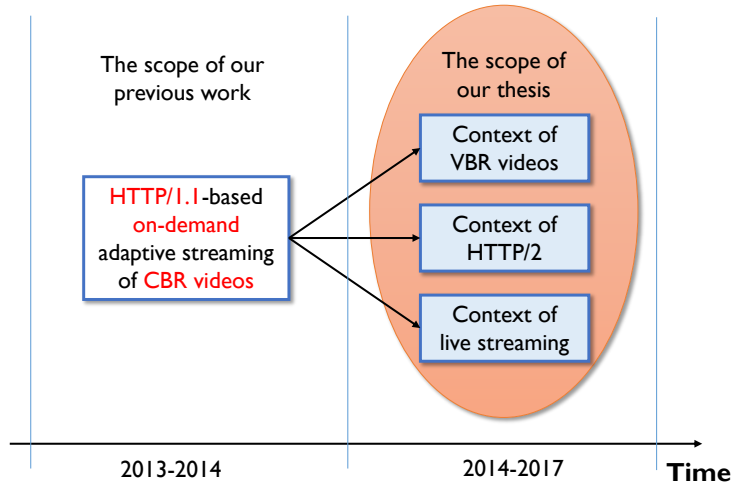


Figure 1.4: The scope of our studies.

1.2 Limitations

Multimedia streaming can be divided into non-interactive streaming (e.g., radio broadcasting) and interactive streaming (e.g., video conferencing). As the focus of this study is how to

improve video quality of a single client in a HAS system, we focus on a specific non-interactive streaming, where mobile users streams videos from an original server. We do not restrict ourselves to multiple clients that share a common bottleneck, but look at the system composed of one server and one client.

In order to maintain the cost-effective benefit of HAS, we intentionally limit modifications to the client side. The server, the transport layer protocol, and delivery networks are basically non-invasive. In other words, the adaptation approaches that require the modifications of the Web server, Transmission Control Protocol (TCP), or mobile IP networks that are used to deliver videos are not fully discussed in this thesis.

As mentioned earlier, most streaming applications adopt the CBR encoding rather the VBR encoding. Thus, when investigating HTTP/2-based adaptive streaming and HTTP live streaming, we consider CBR videos only. The solutions to (i) HTTP/2-based adaptive streaming of VBR videos and (ii) HTTP live streaming of VBR videos will be reserved for our future work.

Another limitation is that we are only interested in the steady stage, where the client receives and playouts video data simultaneously. It is because, the initial delay (i.e. when the client buffers a certain amount of media data at the start of a session) is not a major performance issues [22]. Although a shorter delay is preferable, a longer delay up to several seconds will be tolerated, especially if a user intends to watch a video [23]. During the period of the steady stage, if a video interruption (caused by buffer underflows) happens, the client switches back to the initial delay.

1.3 Main contributions

In this thesis, we present our adaptation solutions for HAS in the three mentioned contexts. Note that our ultimate goal is to provide mobile users with the best possible user experience. The following list briefly summarizes our contributions:

1. *Adaptation method for HTTP adaptive streaming of VBR videos.* To smooth-out the

instantaneous bitrate of a VBR video, we use a local average bitrate as the representative bitrate of a video version. A buffer-based algorithm is proposed to conservatively adapt video quality. Through experiments with multiple VBR videos, we show that our method can provide high video quality even under strong variations of connection throughput and video bitrate.

2. *Seamless HTTP/2 Push-based adaptive streaming with gradual quality transitions.* The recently ratified HTTP/2 protocol provides Server Push feature that helps reduce request-related overheads (e.g., in terms of energy, processing, bandwidth) for clients, servers, as well as network nodes. We propose an adaptation method that not only leverages the push feature of HTTP/2 but also provides buffer stability and gradual quality transitions. Since the method avoids playback interruptions and enables smoother viewing experience, it enhances overall video quality.
3. *Quality-driven adaptation method for HTTP live streaming.* In this context of live streaming, throughput variations easily lead to a large number of playback interruptions due to small buffer sizes. To maintain seamless streaming, the client may drastically switch to a low quality and such sudden quality reductions definitely have negative impact on user experience. In [24], it is found that users may tolerate drastic quality switches in some specific cases. So, we develop a quality-driven adaptation method that improve user experience by providing smooth quality transactions.
4. *Probabilistic adaptation method for HTTP low-delay live streaming.* The low-delay live streaming is typically used in a specific service that requires a delay of few seconds (i.e. lower than a typical delay of 10 seconds in normal HAS live streaming). Because of a smaller buffer size, guaranteeing seamless streaming is more challenging. In fact, most adaptation methods cannot support low-delay live streaming, including our quality-driven adaptation method mentioned above. In this thesis, we present a probabilistic adaptation method to maintain seamless streaming. The proposed method adaptively decides the video bitrate by taking into account the instant buffer level and the probability of

buffer underflow. The experimental results with a buffer size of 4 seconds show that the proposed method can significantly reduce buffer underflows while providing high video bitrates.

Parts of contributions were published in refereed journals and/or international conferences. The list of main publications that are directly related to this thesis is as follows:

- *HTTP adaptive streaming of VBR videos* (Chapter 3). This study was published in the Mobile Information Systems, 2016 [25]. A part of the result was presented at International Conference on Computing, Management and Telecommunications (ComManTel 2015), Vietnam, Dec. 2015. [26].
- *Adaptive streaming over HTTP/2* (Chapter 4). This study was published in the IEICE Transactions on Communications, Vol. E100-B, No. 5, May 2017 [27]. A part of the result was presented at IEEE Global Conference on Consumer Electronics (GCCE 2016), Kyoto, Japan, Oct. 2016 [28].
- *HTTP live streaming with perceptual quality control* (Chapter 5). This study was presented at IEEE International Conference on Communication Workshop, pp. 1771-1776, London, UK, Jun. 2015 [29].
- *Low-delay live streaming over HTTP* (Chapter 6). This study was published in the IEICE Transactions on Information & Systems, Vol. E100.D, No. 2, pp. 379-383, Feb. 2017 [30].

1.4 Thesis outline

This thesis includes two main parts, which is organized as follows:

Part I, “Background of the study”, includes Chapter 2, providing the necessary background information on the HAS technology. We start Chapter 2 with an overview of Internet-based multimedia streaming. This chapter also presents the principles of HTTP adaptive streaming system and quality influence factors.

Part II, “*Solutions to improve video quality*”, focuses on adaptation methods to improve video quality in HAS over mobile networks. We respectively present our adaptation solutions for HTTP adaptive streaming of VBR videos in Chapter 3, for adaptive streaming over HTTP/2 in Chapter 4, for HTTP live streaming in Chapters 5 and 6.

Finally, we conclude our thesis in Chapter 7 with a summary and an outlook on our future work.

2

Background



THIS chapter provides the necessary background information on the technologies used in the present thesis. We start with an overview of Internet-based multimedia streaming in Section 2.1, including the evolution to HTTP Adaptive Streaming (HAS). In Section 2.2, we present a detailed description of the operating principles of HAS. Section 2.3 discusses quality influence factors for HAS, which are performance metrics in our experiments. Finally, we summarize this chapter with Section 2.4.

2.1 Internet-based video streaming

In this section, we first briefly go through the historical development of Internet-based streaming, or, as it is often called, Over-the-Top (OTT) streaming. Then, we present the Internet-

based video streaming landscape in Subsection 2.1.2, where existing streaming services are categorized based on a few core features. Finally, we discuss the evolution from datagram streaming to HTTP Adaptive Streaming (HAS), highlighting the advantages/disadvantages of HAS compared to the conventional streaming.

2.1.1 History of Internet-based video streaming

After the early international standards on video compression (i.e., H261 and MPEG-1) were released in the beginning of the 1990s, video technologies started to establish themselves on local desktop computers. At that time, a video could be digitized, encoded, and stored as video files at servers. The straightforward way to obtain a video, enabled by the benefit the Internet, was to download a video file to the local machine where it could be decoded and then be played back on a screen.

However, this manner created a drawback that a user has to wait until the download of a video file is complete to watch the video, which could take a very long time. Furthermore, he/she has to store the video file in his/her local machine, which is another problem because of storage limitations. A solution to these problems is video streaming. In video streaming applications, the client starts to playout within a few seconds after it starts receiving video data from the server. In other words, the client will play from one location in the video while still receiving data for later parts of the video. This streaming technique is used to avoid downloading a whole video (with a excessive delay) before the playout begins.

In the Internet media streaming era, Internet-based video streaming can be classified into three major categories: (i) Unicast streaming, (ii) IP multicast, and (iii) Application-layer multicast. An overview of video streaming technologies is given in Table 2.1.

Unicast streaming

Since the 1990s, a lot of effort was concentrated on the development of streaming solutions for unicast streaming. Most of the early work on packet video transmission aim to provide real-time transmissions by leveraging resource reservation techniques such as Resource ReSer-

Table 2.1: Video streaming technologies.

Streaming technology	Properties
Unicast streaming	Typically use RTP/UDP or HTTP/TCP to deliver videos. Recently, HTTP adaptive streaming (HAS) has emerged as a <i>de facto</i> standard for streaming videos over the Internet
IP multicast	Can delivery videos over dedicated Internet Protocol Television networks. However, IP multicast does not widely deployed due to both the technical reasons and economic reasons
Application-layer multicast	Use dedicated protocols (e.g., SplitStream or BitTorrent) to deliver videos. Some P2P streaming applications, e.g. PPLive and PPStream, are extremely successful in regional markets

Vation Protocol (RSVP) [31] and Integrated Services (IntServ) [32]. However, RSVP is a transport layer protocol and it is rarely deployed for streaming purpose today. Another important work is open standard Real-Time Transport Protocol (RTP) [11]/Real-Time Control Protocol (RTCP) [33]/ Real-Time Streaming Protocol (RTSP) [12] suite. These protocols are used to establish and control a streaming session, and to help an application maintain a good Quality of Service (QoS). Additionally, they are network protocols and very flexible. Thus, the usage scenarios range widely from low-delay video conferencing to on-demand video streaming. In the literature, recent RTP related work have presented in [34–36] with regard to the Web Real-Time Communication (WebRTC) framework [34], RTP multimedia congestion control [35], and multipath RTP [36].

Although the specifications of RTP/RTCP do not specify their underlying (transport layer) protocols, streaming providers typically use User Datagram Protocol (UDP) as the transport protocol. UDP, as defined in [37], provides the means to send data with a minimum of protocol mechanisms. It does not offer mechanisms to guarantee reliability and to cope with network congestions. Such mechanisms, if required to transmit media data effectively, have to be implemented by a higher protocol layer. As a consequence, streaming systems like these become very complex. Another disadvantage of RTP-based streaming is that the server has to keep track of the state of every streaming session. So, the implementation of complex RTP-based streaming

servers is typically costly. It is a main reason leading to the use of HTTP in video streaming. We will present the evolution from datagram streaming to HTTP adaptive streaming in the Subsection 2.1.3.

IP multicast

IP multicast video streaming is proposed as a solution that provides multipoint delivery directly in IP networks. In its basic form, IP multicast delivers the same video stream to each user of a multicast group. This causes the problem because users may have very different Internet access rates. To this problem, Receiver-Driven Layered Multicast (RLM) is proposed that creates multiple multicast groups and streams a specific representation of the video content (with a specific video bitrate) to each group [38]. Then, a user is supposed to find a right group, of which the video representation is suitable. In [39], an SDN-based framework is presented that can improve the video quality of IP multicast streaming services.

Unfortunately, despite the intensive researches and large-scale pilot projects (e.g., [40]), the widespread support of IP multicast is unavailable due to multiple technical reasons and economic reasons [41]. Furthermore, the improved performance of application-layer multicast issues a challenge to IP multicast streaming [42].

Application-layer multicast

Today, multicast distribution is often accomplished via application-layer multicast or through multiple separate unicast streams. An effective approach to overcome the inefficiency of parallel unicasts was Peer-to-Peer (P2P) streaming [43–45]. With P2P streaming, most media data are exchanged directly among peers, without the presence of the original server. Hence, the service provider is relieved from the burden to serve thousands or millions of parallel unicast streaming sessions. The upload/download data rate increases according to the number of connected peers, makes the system highly scalable. P2P streaming is now extremely successful in China, the list of big companies include Kankan (owned and operated by Xunlei¹), PPTV² (formerly

¹<http://xunlei.com>

²<http://www.pptv.com>

PPLive), and PPs³ (formerly PPstream). Kankan as a P2P-based video-on-demand pioneer in China, serves over 20 million users every month [46].

Although P2P streaming is highly scalable, the transmission and switching capacity of the Internet core might increase slower compared to the traffic demand of the large number of P2P-enabled applications [46]. As a consequence, P2P clients, who generate an excessively high amount of upstream traffic serving other peers, may be restricted or even blocked by network operators. Besides, ensuring network stability in the presence of malicious attacks and congestion will continue to be a significant challenge for P2P streaming.

2.1.2 Internet-based video streaming landscape

This section presents the big picture of video streaming, which provides the background for the description of HAS in the next section. Due to the diversity of video streaming, we categorize them along six attributes: delay requirement, video encoding mode, transport layer protocol, adaptivity, control location, and distribution method. Table 2.2 summaries our categories.

Table 2.2: Internet-based video streaming landscape. Control location refers to the location of adaptation engine (if any) that decides which and when media parts are transferred.

Attributes	Streaming types
Delay requirement	Interactive (less than 150ms), low-delay (less than a few seconds), live (less than 10s), VoD (not applicable)
Video encoding mode	Constant bitrate (CBR) mode, Variable bitrate (VBR) mode
Transport layer protocol	UDP, TCP
Adaptivity	Adaptive, not-adaptive
Control location	Server-driven, receiver-driven, network-driven, hybrid
Distribution method	Unicast, multicast, P2P, hybrid

³<http://www.pps.tv>

Delay requirement

One fundamental approach of video streaming is the capture-to-screen delay requirement it is designed for. Note that the capture-to-delay delay (or live latency⁴) is measured from the time instant the event is successfully captured by a camera to the time instant this event is displayed on the screen. In our study, we distinguish between interactive streaming, low-delay live streaming, live streaming, and on-demand streaming.

An example of interactive streaming is video conferencing between two or more interacting speakers. To maintain a conversation without annoying the users, the delays from the time instant that a user speaks until the others listen should be less than hundreds of milliseconds. Specifically, delays smaller than 150ms are not perceived by a human listener [46].

Without interactive streaming, the others are one way streaming, that is, a video content is transmitted in one-way from an original server to a client. While the low-delay live streaming and live streaming impose limitations on the delay, on-demand streaming does not have any delay requirement. The low-delay live streaming is typically used in specific services that require shorter reaction times such as video surveillance, streaming of sport events. In live streaming, the low delay is an important requirement [47]. To achieve a low delay, the streaming designer should take into account encoding/decoding time, round trip time, buffering delay, etc. [46]. The summary of delay requirements for specific streaming types is shown in Table 2.1.2.

Video encoding bitrate

Most today's video compression codecs combine spatial image compression and temporal motion compensation. Motion compensation, in turn, is an algorithmic technique used to predict a frame, given the previous and/or next frames by accounting for the motions of the camera and objects in the video. After being encoded, a compressed video includes a set of Intra (I) frames, Bidirectional predicted (B) frames, and Predicted (P) frames. I frames are largest because they only use spatial image compression (or intra frame compression). Meanwhile, B and P frames are much smaller because they use previous I frames or other P frames for im-

⁴In this thesis, we use the terms capture-to-screen delay and live latency interchangeably.

age prediction. Again, their size reductions depend on the motions of the camera or objects. Hence, if the quality of video remains unchanged, the frame size vary according to video content, making the output video bitrate (in bitrate per second) naturally varying. The encoding method that keeps the video quality stable is referred to as variable bitrate (VBR) encoding.

As the bitrate predication for a VBR video is not easy, a so-called constant bitrate (CBR) encoding is used to keep the output video bitrate nearly unchanged (by changing video quality during the whole video). CBR videos, because of their predictable traffic patterns, makes the tasks of video streaming and network resource allocation easier, although it also permits the video quality variable. Recently, many commercial services such as Youtube, Facebook live streaming have adopted the CBR mode [48, 49]. Figure 2.1 illustrates the change of instant bitrate and instant quality of a video encoded using the two modes. We see that given the same average video bitrate as a VBR video, a CBR video provides variable video quality but more stable video bitrate.

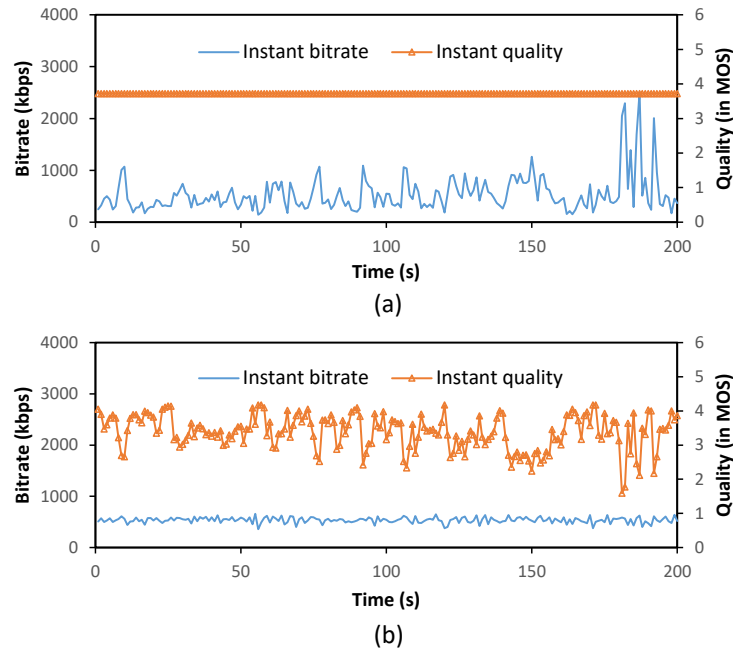


Figure 2.1: The instant bitrate and instant quality of a video encoded in (a) a variable rate and (b) a constant rate. Two encoded videos have the same average video bitrate. Here, the instant quality is measured in Mean Opinion Score (MOS) [50].

Transport protocol

In early 1990s, the choice of the transport layer protocol⁵ for a video streaming service was typically UDP. At that time, it was believed that video streaming would never work well over TCP due to throughput variations caused by (i) TCP's congestion control and (ii) retransmission mechanism. However, TCP has later become the common choice. It is due to the choice of TCP provides many advantages while the main challenge, throughput variations, can be dealt with using adaptive streaming and today's broadband connections. We present the evolution from datagram streaming to HTTP adaptive streaming in Subsection 2.1.3.

Adaptivity

An important feature of a video streaming technology is its capability to adapt the video bitrate to the network conditions, device capabilities, and user context. If a streaming approach is non-adaptive, a user is likely to observe a poor quality of service because non-adaptive streaming definitely results in a lot of video interruptions if the throughput is time-varying (in mobile networks, for example).

Control location

The adaptation engine (if any) can be implemented at the server, client, or network nodes, depending on the streaming technology. Traditionally, this functionality has been deployed on the server side. However, this leads to complexity issues, since the server has to keep state for each stream as well as implement various mechanisms to control this stream. Moreover, to maintain a good quality of service, the server frequently asks/updates a client's status via an out-of-band channel, creating an open loop that results in an additional delay. As a consequence, most non-interactive streaming services (e.g., HAS-based services) deploy the adaptation engine on the client side. There are ongoing efforts to create an efficient framework for network-assisted/server-assisted client-driven streaming services. An example is Assisted DASH (SAND), a recent extension to the MPEG-DASH standard [51].

⁵Note that the transport layer protocol (e.g., TCP, UDP) is distinguished from the transport protocol determined by video streaming services. In HAS, the transport protocol is HTTP (not TCP).

Distribution method

As already mentioned, to overcome the global inefficiency of multiple concurrent unicast sessions, alternative distribution methods such as IP multicast and P2P streaming have been developed. We already discussed these two kinds of video streaming in Subsection 2.1.1.

2.1.3 Evolution from datagram streaming to HAS

This section briefly goes through the evolution of HTTP adaptive streaming from the datagram streaming (Fig. 2.2). We will discuss the advantages/disadvantages of datagram streaming and of progressive download streaming in details.

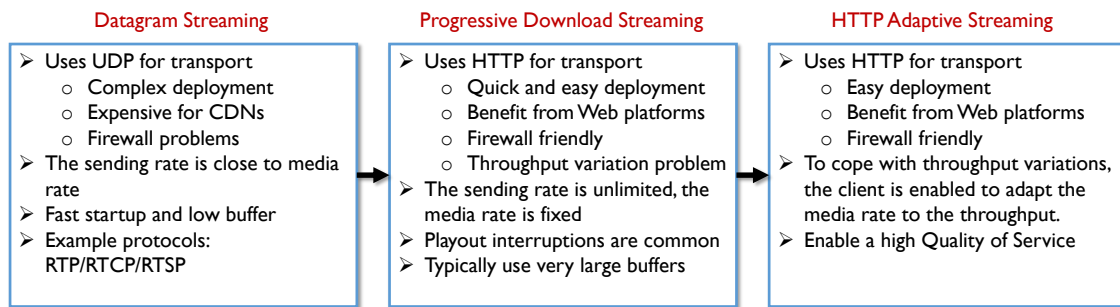


Figure 2.2: The evolution from datagram streaming to HTTP adaptive streaming.

Datagram streaming

In the 1990s and early 2000s, the usual way to deliver videos over the Internet was to use the open standard Real-time Transport Protocol (RTP)/Real-Time Control Protocol (RTCP)/Real-Time Streaming Protocol (RTSP) protocol suite. RTP, developed by the Internet Engineering Task Force (IETF), is an Internet-standard protocol for the transport of real-time data. The RTP standard actually defined a pair of protocols: RTP and RTCP. The RTP is used for exchange of multimedia data, while RTCP is the control part and is used to get feedback from the video receiver about its perception of the connection and the media player's state. Although RTP/RTCP is designed to be independent of the transport layer, it is typically encapsulated in UDP (other options include Datagram Congestion Control Protocol (DCCP) [52, 53] and the Stream Control Transmission Protocol (SCTP) [54]). In general, RTP streaming systems

can have full control over packet retransmission, enabling them to efficiently supports video streaming.

However, a problem with RTP is that it requires out-of-band signaling and different protocols such as Real Time Streaming Protocol (RTSP) [12] and Session Initiation Protocol (SIP) [55]. For example, RTSP is used to negotiate media streaming session parameters and to control the media streaming session (e.g., pause, reposition, fast forward, or tear down). In addition to this, RTP-based streaming applications have three major problems:

- Packet-level control means that the implementation becomes very complex as it has to deal with packet loss, congestion control, flow control, etc.
- RTP-based streaming receives the lack of support from Content Delivery Networks (CDNs) because RTP-based systems require specialized solutions for caching and load balancing (while most CDNs are optimized for HTTP due to its massive popularity [56]).
- Firewalls and network address translation (NAT) routers frequently cause problems with datagram transport protocols. This typically applies to UDP, the most common datagram-based transport protocol.

Progressive download streaming

Because of these above problems, most of the industry adopted HTTP (over TCP) as the transport protocol. This part focuses on the second evolutionary step, that is, progressive download streaming using HTTP (see Fig. 2.2). In this approach, the video file is uploaded to a regular Web server no differently than other Web objects (text file, picture, etc.). The client simply accesses the Web server to download a video part and then plays out the buffered data while it is downloading future parts of the same video. There are several benefits to this simple approach including: (i) the easy implementation and the scaling of Web servers, (ii) the CDN support thanks to HTTP's massive popularity, and (iii) the firewall friendliness, since TCP packets easily pass through almost any firewall and NAT routers.

Unfortunately, the main drawback of progressive download compared to datagram protocols is that the quality of the media must be chosen (manually by the user, for example) at the start of a streaming session, it is likely to result in playout interruptions because the TCP throughput varies during the session. A solution to the problem of progressive download is to use a significantly larger buffer, but it is also limiting progressive streaming's suitability for real-time communications. Another solution is to download the video at the rate that is by a safety margin ahead of the playout rate in order to accommodate for TCP throughput variations [57]. In [58], the authors show that TCP-based streaming offers good performance when the achievable TCP throughput is twice the video bitrate. In fact, TCP throughput varies strongly due to high bit error rates [59]. Yet, modern mobile networks (e.g., 3G networks), however, have techniques to deal with this limitation such as adaptive signal modulation schemes to reduce bit error rates [60].

It was seen that the limitation of implementations of traditional HTTP approaches is that they rarely support dynamic bitrate adaptation in an efficient manner, leading poor performance in time-varying networks. This problem leads to the development of HTTP adaptive streaming, the important streaming standard over the past few years.

HTTP adaptive streaming and MPEG-DASH standard

HTTP Adaptive streaming was first introduced by Move Networks in 2007 [61, 62]. In HAS, a streaming provider generates multiple versions of an original video, each is chopped into short segments. During a streaming session, an adaptation method at the client is responsible for deciding which video bitrate should be requested for each segment. By allowing a client to adapt the video bitrate to the throughput, HAS reduces the interruption rate and thus enables a higher Quality of Service, compared to Progressive download streaming.

After the first launch in 2007, HAS was commercially adopted by dominant companies in parallel, including Microsoft Corporation (with Microsoft Silverlight Smooth Streaming - MSS) [63], Apple Inc. (with HTTP Live Streaming - HLS) [64], and Adobe Systems Inc. (with Adobe HTTP Dynamic Streaming - HDS) [65]. Despite the wide deployment of HAS,

these solutions are basically incompatible, although they share the similar idea.

The first HAS related standard was published for Universal Mobile Telecommunications Systems - Long Term Evolution (UMTS-LTE) by 3GPP in 2009. This work was then further improved in collaboration with MPEG and finally, the international standard called MPEG-DASH was released in 2012 [15]. The second edition of MPEG-DASH was released in 2014 [16]. At the time of writing, the MPEG experts are working towards to the third edition of this standard to enable virtual reality and 360 degree video content over HAS. In addition to the standardization, since 2012, the DASH industry forum (DASH-IF⁶) has been introduced to enable smooth deployment of MPEG-DASH in practice. Currently, the forum has 67 members spread throughout the world, including Microsoft, Apple, Netflix, Qualcomm, Ericsson, Samsung, and many others. Those, who are interested in HAS, can find important streaming players and networking market in DASH-IF.

2.2 HAS principles

The general architecture of HTTP adaptive streaming consists of servers, delivery networks, and clients. Figure 2.3 shows the principle of HAS in a typical streaming including a server and a client. Here, video segments together with their metadata are hosted at the server and will be requested by the client. A client initiates a streaming session by downloading a metadata file, which contains a description of different versions and segments. After that, an adaptation engine deployed at the client takes responsibility for deciding which segments are requested, based on the metadata and the current status of the terminal/networks. The video is therefore delivered to the client via a sequence of HTTP request-response transactions. The following part will introduce each component in HAS system in details.

2.2.1 Video segments and the metadata

Representation and Segment. At the server, an original video is encoded at multiple representations (or versions), using a video codec, for example, H.264, VP9, or H.265/HEVC (High

⁶<http://www.dashif.org/>

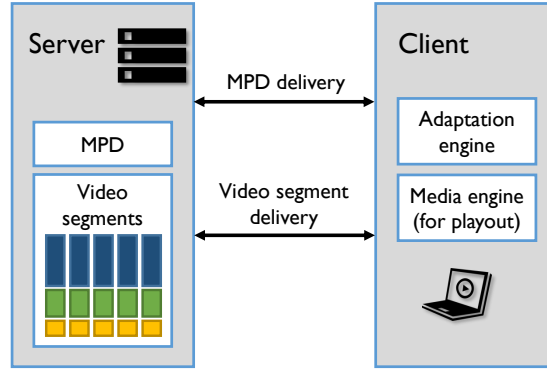


Figure 2.3: Principle of HTTP adaptive streaming in a typical system. Here, MPD refers to Media Presentation Description (or metadata).

Efficiency Video Coding). The representations differ w.r.t various characteristics such as resolution, frame rate, encoding mode (CBR mode or VBR mode), etc. The perceptual quality of a representation is dependent on a set of parameter settings used in the encoder, and it may vary over time if the CBR mode is applied. Typically, the representations differ w.r.t their video bitrate, that is, the number of bits representing one second [51]. A representation given a higher video bitrate usually has a higher perceptual quality.

Each representation is further chopped into short segments, of which the playtimes are equal. In practice, the segment duration is typically from 2 to 10 seconds [10, 66]. To allow seamless quality switching, every frame in a segment must be encoded without any references to/from other segments. So, each segment could be considered as a stand-alone video clip, the playback can start at the beginning of any segment in the stream given the metadata.

As shown in our previous work for CBR videos [67], different sets of representations have different effects on the behavior of an adaptation method. We show that a set of equally-spaced video bitrates is not preferable due to the non-linear relationship between the video bitrate and the video quality. An potential solution to this issue is to use Just Noticeable Distortion (JND) [68], but obtaining the JND value of a video version is very time consuming. In [69], the authors select video bitrates, based on Peak Signal-to-Noise Ratio (PSNR). One of their dataset is used in our experiments (see Table 2.3). In [70], the authors propose criteria to find an optimal set of representations that optimizes QoE. In later work [71], they, however, demonstrate

that the proposed method in [70] is not flexible enough to stream videos to heterogeneous clients when the network is overloaded.

Table 2.3: Representation rates for 2-second dataset.

Resolution	Encoding Level	Representation Rates
320 x 240	1 - 2	100, 150 kbps
480 x 360	3 - 6	200, 250, 300, 400 kbps
854 x 480	7 - 10	500, 700, 900, 1200 kbps
1280 x 720	11 - 13	1.5, 2.0, 2.5 mbps
1920 x 1080	14 - 17	3.0, 4.0, 5.0, 6.0 mbps

Media Presentation Description. In MPEG DASH’s terminology, the metadata file is called Media Presentation Description (MPD) [16]. It is an Extensible Markup Language (XML) file, containing information such as general stream meta information (e.g., file name, encryption information, VoD or Live), types of available stream (e.g., audio, video, subtitles), available representations (including information about codec types), segment indexes, and links to individual segments. Furthermore, if VBR encoding has been used to generate video segments, the information about the size of individual segments may be conveyed by the MPD file. However, the amount of data required to convey the information may be large and most existing MPD don’t contain this information. In the second edition of MPEG-DASH, the client can request the information of a segment size before downloading the actual segment. Specifically, the information is signed to the client using DASH Events [16]. More details about the setting of video segment and MPD can be found in [72, 73].

2.2.2 Video delivery

To deliver videos over the Internet, the MPEG-2 transport stream (M2TS) [74] and ISO Base Media File Format (MP4) [75] are popularly used [22]. In HAS, we only consider transmissions of video segments at the application layer. After obtaining the metadata file, the client issues a series of HTTP requests to download video segments, typically in chronological order, selecting a representation for each of them from the set of available representations.

The decision on which and when representations are requested should be based on the available throughput. Figure 2.4 shows an illustration of switching representations according to the throughput in HAS.

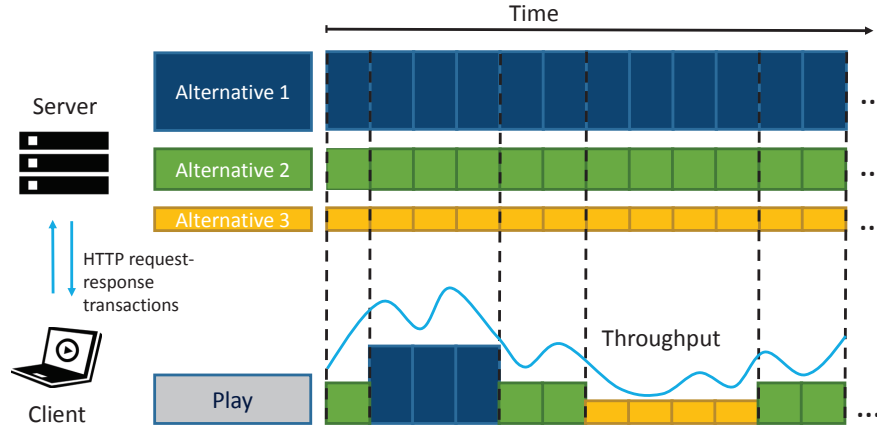


Figure 2.4: Media delivery in HTTP adaptive streaming.

The target protocol of existing HAS systems is HTTP/1.1, where every segment is delivered using a request-response pair [27]. The recently ratified HTTP version 2 (HTTP/2) provides a new feature called “server push”, which enables the client to send one request and then receive multiple objects. This feature has the potential to reduce the request-related overhead and improve the efficiency of low-delay streaming [72, 76].

2.2.3 HAS client

The block diagram of an HTTP client is shown in Fig. 2.5. Depending on the client’s request and the version of HTTP, the server responds by one or multiple segments with the same bitrate. All segments downloaded from the server are stored in the *Playback buffer* before moving to the *Player*. A key component of the client is *Adaptation engine*, where an adaptation method is deployed to make decisions on which media parts are requested. A good decision for the next request should be based on the current buffer level (observed from the Playback buffer), an estimated throughput (provided by the Throughput estimation component) and the Metadata. Note that in some studies, the throughput estimation can be considered as a apart of the adaptation method. While the metadata can be received at the beginning of the

streaming session, the client has to estimate the throughput and measure the buffer level during the whole session.

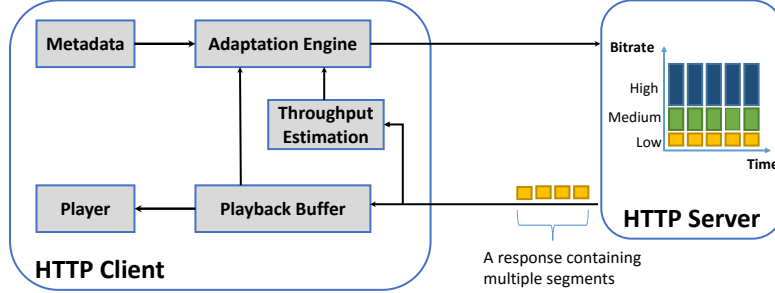


Figure 2.5: Block diagram of an HAS client.

Throughput estimation

In general, the available bandwidth in mobile IP networks is time-varying regardless of the underlying technology or transport protocol used for any content transmission [77]. Especially, TCP, the underlayer of HTTP, is notorious for its throughput fluctuations. In HAS, the monitoring and the estimation of the available connection throughput are often done above the HTTP layer [73]. The cross-layer approach is available (e.g., in [78, 79]) but not popular in practice due to its complex implementation.

In the simplest way, the client can use the instant throughput, which is average throughput $T(l)$ of the last segment l , as the throughput estimate for future segments. However, due to strong throughput variations of mobile networks, the use of the instant throughput may result in short-term fluctuations. A popular solution to cope with this problem is to use the smoothed throughput $T^s(l)$ as follows:

$$T^s(l) = \begin{cases} (1 - \delta) \times T^s(l-1) + \delta \times T(l) & \text{if } l > 1, \\ T(1) & \text{if } l = 1, \end{cases} \quad (2.1)$$

where δ is a weight in the range $[0,1]$.

Playback buffer

The playback buffer (simply referred to as the buffer⁷) plays an important role in coping with throughput fluctuations. It is to absorb throughput variability and to protect the video stream from interruptions. The larger the buffer size is, the lower the probability of interruption is. Note that the buffer level and the buffer size in our study are measured in seconds (not data size) due to bitrate adaptation.

In an HAS session, the client should buffer some amount of video data before it can start playing. This period is referred to as the *initial delay*. After the initial delay, the client switches to the *steady stage*, where it plays and downloads video segments simultaneously. During the steady stage, if the buffer level is large enough, the client can provide users with smooth video quality even under strong throughput variations. However, if the buffer level is low, the user faces video interruptions due to buffer underflows.

Here, one need is to distinguish between on-demand streaming and live streaming. As discussed earlier in Subsection 2.1.2, live streaming imposes limitations on the delay, since the content in live streaming is being recorded, encoded, and published to the server while being streamed. Therefore, unlike on-demand streaming where the buffer level could be increased (by simply requesting the lowest quality segments, for example), the buffer level in live streaming is limited. Especially for low-delay live streaming, the buffer level cannot exceed a few seconds, this poses a challenge to adaptation methods.

We remark, however, that even with VoD, where all complete video segments are available for download, the amount of buffered data is typically bounded from above. This is because, in a streaming session, a user may quit anytime and big amount of unused data in the buffer would reduce the utilization of network capacity. The buffer size for on-demand streaming is typically set to 30 seconds.

⁷Hereinafter, we use the term the buffer to refer the client buffer.

2.3 Quality influence factors for HAS

Video quality for HAS is currently a very active and fast developing research area [80–82]. TCP, the underlayer of HTTP, provides the reliability and thus eliminating negative impacts due to packet losses and jitters. Instead, they are translated into throughput variations. So, the main quality influence factors for HAS includes: (i) initial delay, (ii) interruptions, (iii) perceptual quality, and in case of live streaming, (iv) live latency (Fig. 2.6). The following parts, we describe each of these factors in more details.

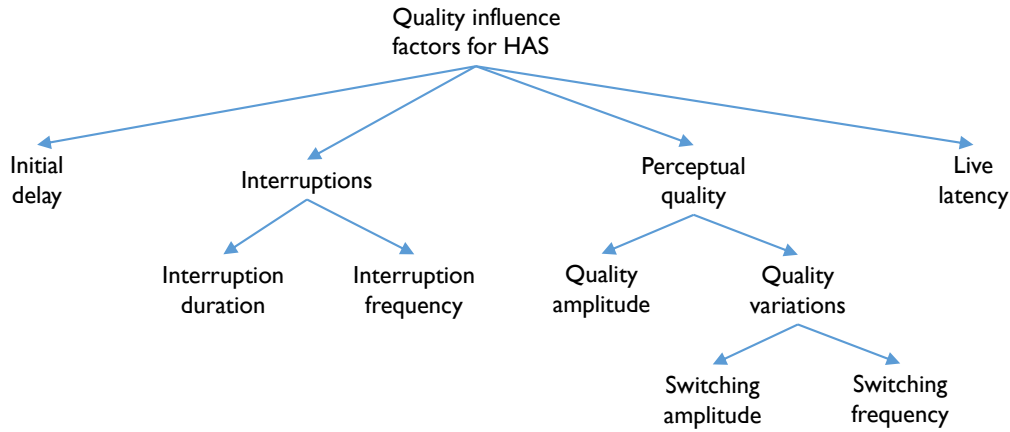


Figure 2.6: Taxonomy of quality influence factors for HAS.

2.3.1 Initial delay

Initial delay is always present in a video streaming as some amount of data must be transferred before the decoding and playback can begin. This phase is sometimes termed initial buffering. In the initial delay, the client does not need to fill its buffer with a big amount of data, since this may cause a long initial delay. It is found that there is a logarithmic relationship between the waiting times and the user experience [22]. Nevertheless, a larger buffer level helps the client efficiently avoid buffer underflows. So, the amount of initially buffered data is a trade off between the actual length of the corresponding delay and the risk of buffer underflows [22].

Reference [23] finds the fundamental difference between initial delay and interruption. This shows that initial delays are preferred to interruptions by around 90% of users. An initial

delay of 16 seconds may reduce the perceived quality only marginally, especially if the user intend to watch that video [22, 23]. The study in [83] confirms for mobile users that initial delay is less important than other factors and it is less critical for having a good video quality.

2.3.2 Interruptions

Interruption is the freezing of video playback due to buffer underflows. Specifically, a buffer underflow is followed by a rebuffering period, where the client has to buffer quickly some amount of video data to resume playback. In some streaming services, a rebuffering could be treated similarly to an initial buffering. However, the impacts of the two kinds of waiting time is different on users. In contrast to initial buffering, which is waiting time before the service and is well known, interruption happens unexpectedly within the service and thus, it is perceived much worse [84].

In [85], the authors show that a longer duration of interruption degrades the video quality. It is found that short interruptions are worse than a long ones [86]; and interruptions at irregular intervals are worse than the periodic interruptions [87]. In [88], the authors show that there is an exponential relationship between the number of interruptions and video quality. Although they find that users can tolerate at most one interruption of a few seconds during a session, many studies agree that interruptions should be avoided to have a good video quality [22].

2.3.3 Perceptual quality

In HAS, a higher video bitrate typically provides a higher video quality. The work in [89] observes a strong effect of recent video quality, that is, a higher quality in the end of a session results in higher user experience. In [90], it is found that besides the average video bitrate, the time on each individual layer has a significant impact on video quality.

In classical HTTP-based streaming (i.e., progressive download streaming), the key influence factors on video quality are initial delay, interruption and quality amplitude [88, 91]. In HAS, the client changes the delivered video quality during a session, which introduces quality

variation as an additional influence factor on perceptual quality [92, 93]. In [93], it is found that the video adaptation strategy related parameters (i.e., representation switches) have to be considered on a larger time scale (up to some minutes) and that they are more important than video encoding related parameters (e.g., resolution, frame rate, quantization parameter, etc.), which only influence in the order of a few seconds.

The quality variation in a session is determined by switching amplitudes and the switching frequency. It is worth noting that each switching is represented by a non-zero quality change (e.g., video bitrate, video version, etc.) of the two conservative video segments. A switching amplitude indicates the degree of a quality change while the switching frequency could be represented by a number of switches in the whole session. In [94, 95], the authors show that a change of the video bitrate improves or decreases the video quality according to the switching direction, but downgrading the video bitrate has a stronger impact on video quality than upgrading the video bitrate. An analysis of quality variation in [96] show that continuously switching down to some intermediate representations provides a better video quality than suddenly declining to the target representation. However, if the instant representation should be increased, an abrupt increase of representation may even improve the perceived quality since users are happy to perceive the quality improvement [97]. Besides, the frequency of the quality switches should be minimized for a high video quality [89].

2.3.4 Live latency

In case of live streaming, an additional quality influence factor that plays an important role is the live latency (or the capture-to-display delay). As discussed in [98], the main delay components include content preparation, segmentation delay, asynchronous fetch of media segments, HTTP download time, buffering time in the buffer, and decoding time. Among these components, the buffering time in the buffer, which depends on the current buffer level, contributes a significant part to the live latency, especially in on-demand streaming where the buffer size is up to tens of seconds.

Currently, live adaptive streaming over HTTP exhibits a latency of a few seconds [51, 67].

Throughout extensive experiments, the author in [99] shows that the minimal buffer size of 10 seconds is required for HTTP/1.1-based mobile streaming in order to obtain an acceptable QoE with a relatively low number of video interruptions. So, the latency should be reduced in order to help low-delay streaming services (e.g., video surveillance, video conference, sport live streaming) improve quality of service.

2.3.5 Maximizing video quality

Note that the individual factors described above cannot be considered separately. For example, in order to avoid interruptions and to minimize the number of quality transitions, a client may always select the lowest video quality, which obviously degrades the overall quality if the available network capacity actually allows a higher media bit rate. On the other hand, maximizing the video quality by always selecting the highest representation often leads to an unacceptably high number of interruptions.

Some of the described factors, such as switching amplitude and switching frequency, have not been part of traditional quality metrics for progressive download streaming, but have a tremendous impact on video quality for HAS. Still, the impacts of the individual factors are still not fully understood yet [100]. Nevertheless, the number of studies on this topic has been dramatically increasing with the growing popularity of streaming services, so that a lot of valuable insights are available to help improve video quality for HAS [22].

Many studies suggest that the number and duration of interruptions have the most severe impact on video quality, especially in case of live streaming [101]. In particular, users are willing to accept a higher initial delay and higher video distortion, if it helps to minimize either the number or the duration of interruptions [23, 102]. Again, many studies agree that interruptions should be avoided to have a good video quality.

As for perceptual quality, some recent studies come to the conclusion that a slightly lower quality amplitude might be tolerated if it helps reducing the amount of representation changes [83]. However, generally, the quality amplitude has to be maximized first, and the number and frequency of quality switches are less important [103].

In recent years, a number of quality models have been proposed to estimate the overall perceptual quality, which is based on the average [104, 105], the median [106], the minimum [106], the standard deviation [104] of the segment quality values as well as the switching frequency [104, 107]. In [100], the histogram of segment quality values and the histogram of quality gradients in a session are used to model the overall perceptual quality; but this model is limited to VBR videos and it does not consider interruption factor. So, modeling the quality of experience for HAS is still an open question. A recently comprehensive survey on quality of experience of HAS can be found in [22].

2.4 Summary

In this chapter, we have presented the background of our study. We carefully described of the operating principles of HAS. After that, the three quality influence factors for HAS (i.e., the interruption, perceptual quality, and live latency), which are major performance metrics in our experiments, were discussed in details. The next four chapters will present our solutions to the three streaming contexts.

3

Adaptive Streaming of Variable Bitrate videos



So far, existing adaptation methods have mostly focused on CBR (constant bitrate) videos. The research on HAS for VBR (variable bitrate) videos is still limited. The problem with VBR videos is that, even though the throughput is stable, strong fluctuations of video bitrate may result in buffer underflows [108]. Our previous work in [108] is the first study on HAS that supports VBR videos by estimating both the instant bitrate and the instant throughput. In the context of managed IPTV networks, where the bandwidth is allocated in advance, the delay-quality tradeoff of a VBR video is optimally achieved by replacing some high-bitrate segments with low-bitrate segments [109]. In [110], a

buffer-based adaptation method is proposed for VBR video streaming by using a partial-linear buffer prediction model along with a strategy to select versions in different buffer ranges. However, this method does not consider bitrate values, and so it cannot avoid sudden changes of quality when video bitrate and throughput are drastically varying.

In this chapter, we present a novel adaptation method which can effectively support VBR videos. By extending our preliminary work in [111], the proposed method can cope with variations of throughput as well as video bitrate. Especially, our method takes into account the moving average of bandwidth and video bitrate to provide stable streaming quality without sudden changes. The experimental results in the mobile streaming context show that our approach can provide consistent VBR video streaming with smooth video quality and stable buffer level. To the best of our knowledge, this is the first method that can provide smooth version transitions for VBR video streaming over HTTP. It should be noted that our method does not require the exact values of video segment bitrates to make decisions.

The organization of this chapter is as follows. Section 3.1 presents the adaptation related work. The principles of our method as well as the algorithm description are presented in detail in Section 3.2. Section 3.3 provides our experimental results and discussions. Finally, Section 3.4 summarizes this chapter.

3.1 Related work

3.1.1 Adaptation method overview

In general, an adaptation method needs to answer two key questions: 1) should the current version be maintained? and 2) if not, which version should be switched to? As discussed in [67], existing methods can be divided into a throughput-based group and a buffer-based group. This is a rough division because buffer level is strongly affected by the throughput.

Throughput-based group: Throughput-based methods decide the bitrate based on a throughput estimate only. They are different in the ways they estimate or use the throughput. The work in [112] found that sessions sharing similar key features (e.g., the same ISP geographical

region) have similar throughput behaviors. A history-based algorithm called CS2P (Cross Session Stateful Predictor) is then developed to estimate the throughput. The similar history-based approach is seen in [113]. Note that these history-based throughput estimations methods can be applied to any kind of Internet-based services (not only video streaming services).

In HAS, the simplest way is to use the measured throughput right after having fully received a segment (called instant throughput) as the throughput estimate of the next segment. Another approach is to use a smoothed throughput measure [114] to avoid short-term fluctuations, which are a drawback of using instant throughput. However, this may cause late reaction of the client to large throughput drops. In [10], we propose a throughput estimation method that has the advantages of both instant throughput and smoothed throughput. Furthermore, other studies also present different ways to obtain the estimated throughput based on sampled throughput values and RTT [108], probing [62], or Markov chain [115].

Once the client obtains the estimated throughput, the version can be decided in many ways. A simple solution is using a safety margin to compute an appropriate version for the next segment [10]. In [116], the video version is controlled by a TCP-like mechanism, where a measure proportional to the instant throughput is used as the key input.

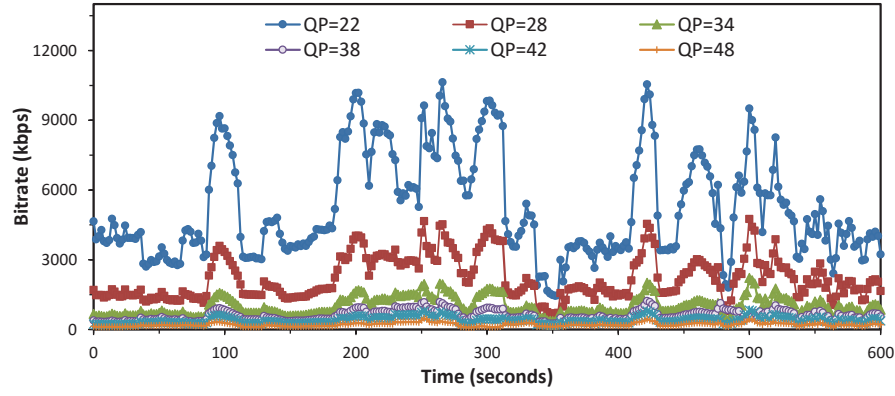
Buffer-based group: Because throughput-based methods often lead to strong bitrate variations, most recent studies have focused on buffer-based methods that can leverage the client buffer to support bitrate stability (e.g., in [117–119]). Note that buffer-based methods also leverage throughput estimate in making decisions. In [117], we introduce a trellis-based method that represents all possible changes of the versions and corresponding buffer levels in the near future. Thus, this approach can make good decisions on the bitrates of some future segments. In [118], the adaptation problem is formulated as an optimization problem of some next segments, which is similar to [117]. In [119], the proposed method selects the bitrate using a function of the current buffer level without considering any throughput measures. However, as the buffer level always fluctuates together with throughput variations, deciding the bitrate to avoid bitrate variations is not easy. If the throughput varies strongly, the method requires a big buffer size to prevent interruptions.

In [120], an adaptation method is proposed that uses Markov Decision Process to decide the video bitrate. Besides, the video quality can be controlled by using control theory (e.g., [121–124]), fuzzy logic (e.g., [125–127]), and machine learning (e.g. [128–131]). The common issue of such methods, however, is that they have some tuning parameters, which are not easy to set even for streaming experts [73].

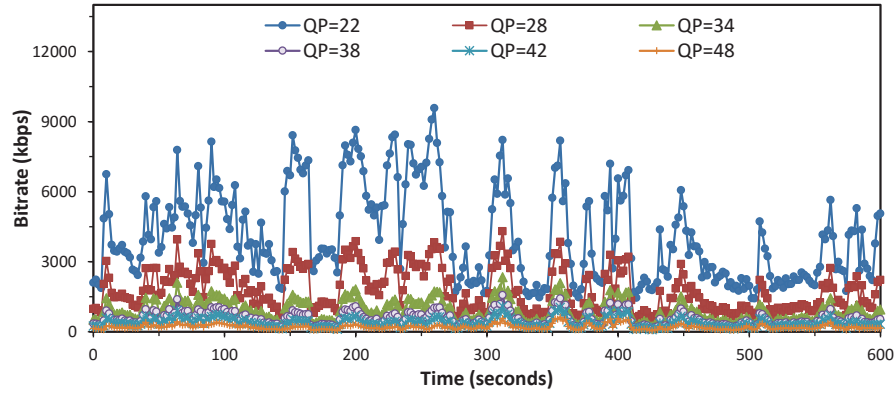
In existing buffer-based methods, an efficient approach in terms of implementation and performance is to divide the buffer into multiple ranges by some empirical thresholds, which remain unchanged during a whole session. The basic idea is that, when the throughput goes down, the current bitrate can be maintained if the buffer level is still within a “safe” buffer range [132–134]. In [133], if the current buffer level is in the range of 35% ~ 50% of the maximum buffer level, the throughput estimate for the next segment is the same as the previous throughput. If the buffer level is reduced into lower ranges, the throughput estimate is equal to the previous throughput multiplied with a down-scaling factor. The bitrate being less or equal to this throughput estimate is then selected for the next segment. The methods proposed in [132] and [134] are similar and can maintain a smooth bitrate curve in on-demand streaming. However, when the throughput is suddenly reduced, these methods may lead to drastic decreases of buffer level since they gradually change the bitrate without considering resulting buffer levels. As a consequence, the client has to jump to the lowest bitrate to avoid buffer underflows when the buffer level is reduced into a dangerous range, resulting in a large bitrate change. Other papers that propose adaptation methods that uses thresholded buffers are [135] and [136].

3.1.2 Adaptation methods for VBR video streaming

Yet, most of the existing methods have been developed only for the context of CBR videos, where the bitrate of a version is constant. Compared to CBR videos, videos encoded in VBR mode have important advantages in terms of quality and network resource usage [19]. However, the variations of video bitrate over time, together with throughput fluctuations, result in a big challenge for HTTP adaptive streaming [108]. Two examples of how bitrates of different versions vary, especially in some scene changes, are illustrated in Fig. 3.1. Detailed information



(a) Sony demo



(b) Terminator 2

Figure 3.1: Bitrates of the versions of two test videos [137].

of these videos will be described in Section 3.3.

Our previous work in [108] is the first study on VBR video streaming over HTTP. Besides throughput estimation, this method also considers the estimation of the instant video bitrate, which can be divided into 1) intra-stream estimation and 2) inter-stream estimation. The former estimates the bitrates of segments within a version, while the latter estimates the bitrates of segments across different versions. This method can provide a very stable buffer, and moreover, can support a CBR-like streaming service from VBR videos. In [110], a buffer-based adaptation method for VBR videos is proposed, where the buffer is divided into multiple ranges. In order not to take into account varying video bitrates, this method uses a partial-linear trend prediction of the buffer level for choosing versions in different buffer ranges. If no significant change of buffer level is estimated, the client will maintain the current version for the next segment. However, this method still causes sudden version changes if the actual buffer level declines drastically.

In this study, we propose a novel adaptation method that can effectively support VBR videos in on-demand streaming. The distinguishing features of our method include:

- To cope with strong variations of video bitrates, we propose using a local average bitrate as the (moving) representative bitrate of a version. Since this representative bitrate is stable in short term, it helps the client clearly differentiate the available versions and make a good version selection at each time instance.
- Because the client does not have enough information about the segment bitrates of all versions, we provide an algorithm to obtain an estimated representative bitrate of each version.
- Regarding the first key question, quality stability is effectively maintained by 1) using a smoothed throughput estimate when the buffer is not in danger, 2) using the representative bitrate, and 3) being conservative in quality switching.
- Regarding the second key question, smooth transitions of quality are supported by avoiding jumping simply to the lowest version in panic case and by early switching down when there is a throughput-bitrate mismatch.

3.2 Adaptation solution

In this section, we present a new buffer-based quality adaptation method for VBR video streaming. Some notations along with their definitions used in the chapter are provided in Table 3.1. It should be noted that for clear explanation purpose, we use a specific set of notations in each chapter.

3.2.1 Handling throughput and bitrate fluctuations

Generally, based on the measured throughput and the video bitrate, the client should choose an appropriate version for the next segment. Suppose that, after receiving the current (or last) segment i of version I_i , the client measures the bitrate B_{i,I_i} and throughput T_i of this segment.

Table 3.1: Notations and definitions.

Notation	Definition
T_i	The throughput of segment i
T_{i+1}^{est}	The throughput estimate of segment $i + 1$
β_{cur}	The current buffer level
β_{min}	The minimum buffer threshold
β_i^{tb}	The flexible buffer threshold
β_{max}	The buffer size and also the target buffer level
$B_{i,k}$	The bitrate of the segment i in version k . It could be the actual value $B_{i,k}^o$ or the estimated value $B_{i,k}^e$
$B_{i,k}^{rep}$	The representative bitrate of version k at segment i
V	The number of available video versions
I_i	The index of the version which is chosen for segment i (version of higher quality has a higher index value)

Now the client will decide the version I_{i+1} for the next segment $i + 1$. Our proposed method will also leverage the client buffer to cope with the fluctuations of both the throughput and the video bitrate. Depending on the current buffer level β_{cur} , the client will decide whether the version should be increased, decreased, or maintained.

For the version selection of the next segment, it is necessary to estimate the throughput based on the throughput history of received segments. To avoid the effects of short-term fluctuations of instant throughput, we use a smoothed throughput measure T_i^s [10, 132] as the throughput estimate T_{i+1}^{est} for the next segment $i + 1$:

$$T_{i+1}^{est} = T_i^s = \begin{cases} (1 - \delta) \times T_{i-1}^s + \delta \times T_i & \text{if } i > 0, \\ T_i & \text{if } i = 0, \end{cases} \quad (3.1)$$

where δ is a weighting value, which is set to 0.1 in this chapter.

As video bitrate is highly fluctuating, we propose using a representative bitrate for each version, which can differentiate the available versions and can also be appropriate for maintaining quality stability. Denote $B_{i,k}^{rep}$ the representative bitrate for version k at segment index i . In our method, $B_{i,k}^{rep}$ is calculated as the average bitrate of N recent segments of version k .

The problem is that the client only knows the bitrates of the received segments, which may belong to different versions. So, after receiving each segment i , we will estimate the segment bitrates of other versions with the same index i . This is enabled by the bitrate estimation method proposed in our previous study [108], where the segment bitrates of other versions are estimated from the bitrate of the received segment using the inter-stream bitrate prediction. Specifically, the estimated bitrate $B_{i,k}^e$ of version k can be calculated from the (actual) bitrate $B_{i,n}^o$ of the received segment i with the selected version n as follows:

$$B_{i,k}^e = \mathfrak{V} \times B_{i,n}^o \times 2^{\frac{QP_n - QP_k}{6}} \quad (3.2)$$

where QP_k and QP_n are the quantization parameter (QP) values of the versions, and $\mathfrak{V} = 1.05$ is an empirical factor used as the compensation for the approximation error of the model [108]. In our notation (Table 3.1), bitrate $B_{i,k}$ can be either the actual bitrate $B_{i,k}^o$ or the estimated one $B_{i,k}^e$. Once obtaining the bitrates of all segments, the representative bitrate of each version at segment index i is calculated as the average of the bitrates of segment i and $N - 1$ previous segments in that version. The algorithm to calculate the representative bitrates is provided in Algorithm 3.1. In this algorithm, the current representative bitrate is actually computed using the previous representative bitrate. Also, it should be noted that, when $i < N$, N will be set to i . In Section 3.3, we will investigate how different values of N affect the performance of our method.

3.2.2 Adaptation algorithm

Our algorithm will address the two key questions above, so as to avoid rebuffering and to reduce the number and the degree of (version) switches. Based on the current buffer level of the client, we define four possible cases in a streaming session, which are uptrend, stable, downtrend, and panic cases. In these four cases, the client will be likely to switch up, maintain, switch down, or aggressively decrease the version. For this purpose, our method divides the buffer into three ranges with thresholds β_{min} and β_i^{th} ($\beta_{min} < \beta_i^{th} < \beta_{max}$). Here β_{max} is the buffer size and

Algorithm 3.1: Representative bitrate computation (after receiving the last segment i and for all video versions).

Input: $N, B_{i-1,k}^{rep} |_{1 \leq k \leq V}, B_{i-j,k} |_{0 \leq j < N, 1 \leq k \leq V}$
Output: $B_{i,k}^{rep} |_{1 \leq k \leq V}$

```

1 for  $k \leftarrow 1, 2, \dots, V$  do
2   // Check if bitrate is the original bitrate
3   if  $k = I_i$  then
4      $B_{i,k} \leftarrow B_{i,k}^o$ ;
5   // Otherwise the bitrate is the estimated bitrate
6   else
7     Estimate  $B_{i,k}^e$  by (3.2);
8      $B_{i,k} \leftarrow B_{i,k}^e$ ;
9   // Compute  $B_{i,k}^{rep}$ 
10   $B_{i,k}^{rep} \leftarrow B_{i-1,k}^{rep} + (B_{i,k} - B_{i-N,k})/N$ ;

```

also the target buffer level of the adaptation method.

When the current buffer level exceeds β_{max} , the uptrend case is activated. (For the reason why the buffer level could be higher than β_{max} , please refer to our previous work [67]). However, it is not good if the client frequently goes back and forth between the uptrend case and downtrend case. To avoid this fluctuation, our method switches up to the next higher version ($I_i + 1$) only if the representative bitrate B_{i,I_i+1}^{rep} of that version is smaller than the throughput estimate of the next segment (i.e. $B_{i,I_i+1}^{rep} < T_{i+1}^{est}$); otherwise, the client will maintain the current version.

The stable case is determined by the condition $\beta_i^{th} \leq \beta_{cur} < \beta_{max}$. In this case, the buffer level is judged as in a very safe condition, so no change in quality is needed. The client just maintains the current version to avoid unnecessary switches.

The downtrend case is activated when $\beta_{min} \leq \beta_{cur} < \beta_i^{th}$. In this case, the client needs to carefully decide the requested version in order to avoid buffer underflows as well as sudden quality changes when the throughput and/or the video bitrate change drastically. In general, the version should be decreased; however, it is unnecessary to switch down always when the buffer level is in this range. In this process, we define the target bitrate for the next segment $i + 1$ as the highest representative bitrate, which is lower than the throughput estimate:

$B_{i+1}^{tar} = \max\{B_{i,k}^{rep} | B_{i,k}^{rep} < T_{i+1}^{est}\}$. If the instant bitrate and the representative bitrate do not exceed the target bitrate ($B_{i,I_i} \leq B_{i+1}^{tar}$ and $B_{i,I_i}^{rep} \leq B_{i+1}^{tar}$), the current version will be maintained. Otherwise, the client will switch down to the next lower version.

We can see that sometimes the instant throughput might be much smaller than the instant bitrate. Even though the buffer is not in danger yet, the downtrend case should be activated earlier to avoid having to quickly reduce the version in the future. This is enabled by increasing the value of β_i^{th} . In our method, β_i^{th} is controlled using a logistic function as follows:

$$\beta_i^{th} = \beta_{max} - \frac{1}{1 + e^\sigma} \times (\beta_{max} - \beta_{min}) \quad (3.3)$$

$$\text{where } \sigma = 1 - \frac{T_i}{B_{i,I_i}}. \quad (3.4)$$

In this way, the larger the mismatch between T_i and B_{i,I_i} becomes, the higher the value of β_i^{th} will be, while still satisfying the condition $\beta_{min} \leq \beta_i^{th} < \beta_{max}$.

The final case, called the panic case, is when the buffer level is in danger, i.e. $\beta_{cur} < \beta_{min}$. To ensure that the buffer will not be empty, the client will aggressively switch down the version. Yet, instead of jumping directly to the lowest version, the client employs the method of our previous work [108] in this case. Specifically, the client will choose a version, of which the instant bitrate is the highest but still lower than the instant throughput:

$$I_{i+1} = \arg \max_{1 \leq k \leq V} \{B_{i,k} | B_{i,k} < T_i\}. \quad (3.5)$$

The algorithm of our method is summarized by pseudo code as in Algorithm 3.2.

Generally, it can be seen that our method tries to provide a smooth video stream by two techniques. First, it is somewhat conservative in increasing the quality because that is allowed only when the buffer is full. In addition, the selected version is constrained by the long-term values of throughput and bitrate. Second, the downtrend case is also conservative by avoiding continuously switching down the quality. Meanwhile, in the panic case, we take into account the instant bitrate and instant throughput. Thus, the client can switch down gradually when

Algorithm 3.2: Adaptation algorithm for VBR video streaming.

```

Input:  $T_i, \beta_{cur}, I_i, B_{i,I_i}, B_{i,I_i}^{rep}, T_{i+1}^{est}$ 
Output:  $I_{i+1}$ 
1 // Uptrend case
2 if  $\beta_{cur} > \beta_{max}$  then
3   if  $B_{i,I_i}^{rep} < T_{i+1}^{est}$  then
4      $I_{i+1} \leftarrow I_i + 1;$ 
5   else
6      $I_{i+1} \leftarrow I_i;$ 
7 // Stable case
8 else if  $\beta \in [\beta_i^{tb}, \beta_{max})$  then
9    $I_{i+1} \leftarrow I_i;$ 
10 // Downtrend case
11 else if  $\beta \in [\beta_{min}, \beta_i^{tb})$  then
12    $B_{i+1}^{tar} \leftarrow \max\{B_{i,k}^{rep} | B_{i,k}^{rep} < T_{i+1}^{est}\};$ 
13   if  $B_{i,I_i} \leq B_{i+1}^{tar}$  and  $B_{i,I_i}^{rep} \leq B_{i+1}^{tar}$  then
14      $I_{i+1} \leftarrow I_i;$ 
15   else
16      $I_{i+1} \leftarrow I_i - 1;$ 
17 // Panic case
18 else
19   Select  $I_{i+1}$  based on Eq. (3.5);

```

possible; and there is no need to switch to the lowest version if the instant bitrate of a higher version still meets the throughput constraint.

3.3 Experiments and Discussions

In this section, we will evaluate our proposed method and two reference methods in the context of on-demand VBR streaming over mobile networks, focusing on the behaviors of version switching and buffer level after the initial buffering stage. Two bandwidth traces, a simple one and a complex one, are employed in our experiments.

3.3.1 Experiment setup

Our test-bed organization used for the experiments is similar to that of [67], which consists of an HTTP Web server, a streaming client and an IP network as in Fig. 3.2. The IP network

includes a router and wired connections connecting the server and the client. The server is an Apache HTTP server of version 2.2.21 running on Ubuntu 12.04. Our test-bed uses DummyNet tool [138] installed at the client side to emulate network characteristics. The packet loss rate is set to 0%, assuming that the bandwidth trace used in the experiments already takes into account the fluctuations caused by packet loss. RTT value of DummyNet is set to 40ms. The client is implemented in Java and runs on a Windows 7 notebook with Core i5 2.6GHz CPU and 4GB RAM.

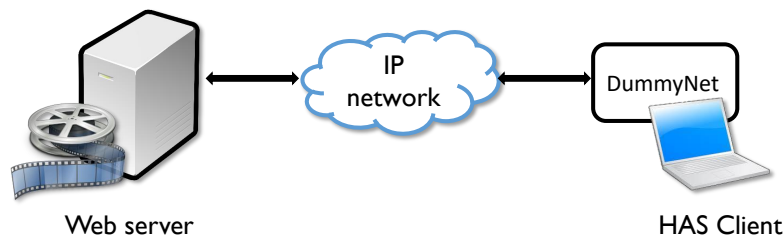


Figure 3.2: Test-bed organization for experiments.

The test videos are “Sony Demo” and “Terminator 2” [137] with a frame rate of 30fps and a resolution of 1280x720. The duration of each video is 600 seconds. We encode 6 VBR versions by the High profile of H.264/AVC [139], corresponding to 6 different values of QP, namely 22, 28, 34, 38, 42 and 48. Each version is divided into small video segments of 2 seconds. The version index, QP, and the average bitrate of each version are listed in Table 3.2. The bitrate traces of the video versions are shown in Fig. 3.1.

Table 3.2: Version information of the two test videos.

Index	QP	Average bitrate (kbps)	
		Sony Demo	Terminator 2
1	48	203.77	201.55
2	42	390.75	377.97
3	38	602.96	567.02
4	34	991.32	882.29
5	28	2194.05	1798.93
6	22	5180.58	4127.86

As we focus on on-demand streaming, the buffer size of the client is set to 50s (i.e. 25 seg-

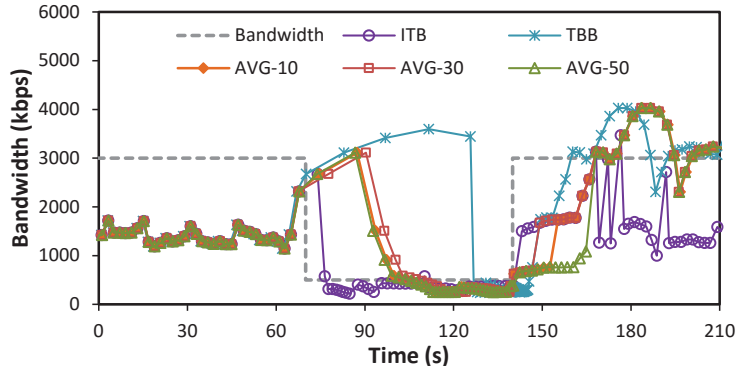
ment durations), which is similar to previous work (e.g. [110, 111]). For comparison, two reference methods which are the instant throughput- instant bitrate based method [108] (called ITB) and the buffer-based method with trend prediction [110] (called TBB) are employed. The TBB method is implemented with buffer thresholds $(B_{min}, B_{low}, B_{high}, B_{max}) = (10s, 20s, 40s, 50s)$, which are the same as the settings in [110]. In our method, β_{min} and β_{max} are 10s and 50s, respectively. Also, we investigate whether the number of segments N used for representative bitrates affects our method's performance. The values of N considered are 10, 30 and 50, and these options of our method are referred to as AVG-10, AVG-30 and AVG-50, respectively.

3.3.2 Simple bandwidth scenario

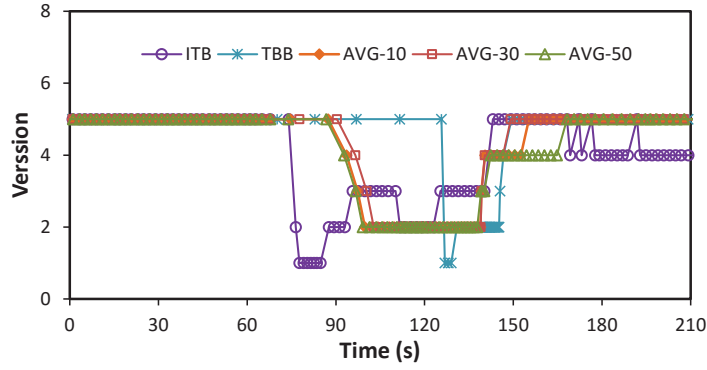
First, we investigate the performance of the methods in a simple bandwidth scenario, when the available bandwidth drops suddenly. The bandwidth has a rectangular shape with two bandwidth levels, 2500 kbps and 500 kbps, as shown in Fig. 3.3a. This case is important in evaluating adaptation methods because we need to know how they perform when the bandwidth drops drastically. In this case, the Sony Demo video is used.

Figure 3.3 shows the comparisons of requested versions, bitrates and buffer levels of the three methods. It is clear that the TBB method tries to maintain the high quality (version 5) for too long even when the available bandwidth drops and stays at the low level for a long interval, resulting in the worst buffer level curve (Fig. 3.3c) as well as a drastic drop of quality (around $t = 125s$, from version 5 to version 1 in Fig. 3.3b). As for the ITB method, because it requests versions following the variations of instant throughput and instant video bitrate, the quality is aggressively changed over time while the buffer level variations are very small.

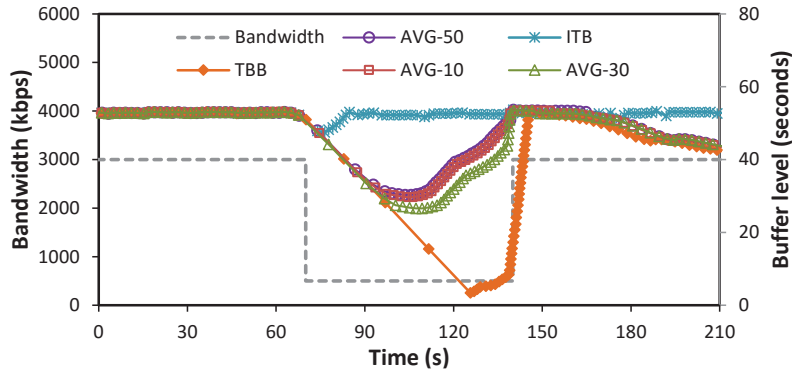
As for our method, all the three options have similar behaviors in terms of bitrate, version switch, and buffer level. Our method reduces the video quality gradually with no switches larger than 1 while the buffer level is higher than 25 seconds. Also, the minimum version provided by our method is 2, while the other two methods sometimes jump to version 1.



(a) Segment bitrate



(b) Requested version



(c) Resulting buffer level

Figure 3.3: Adaptation results of the three methods in simple bandwidth scenario.

3.3.3 Complex bandwidth scenario

In this part, we evaluate the adaptation methods with a complex bandwidth trace (Fig. 3.4), which was obtained from a mobile network [133]. Both test videos, “Sony Demo” and “Terminator 2”, are employed in this scenario.

Figure 3.5 shows the experimental results for the “Sony Demo” video. As seen in Fig. 3.5a,

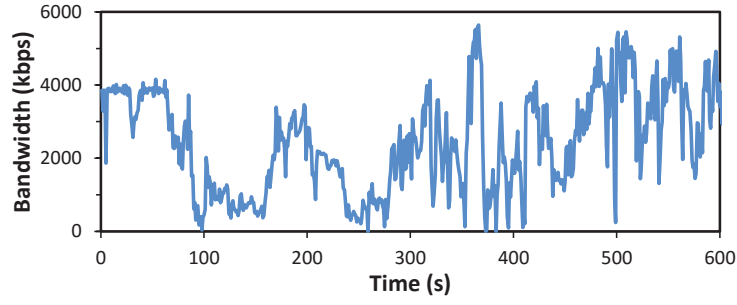


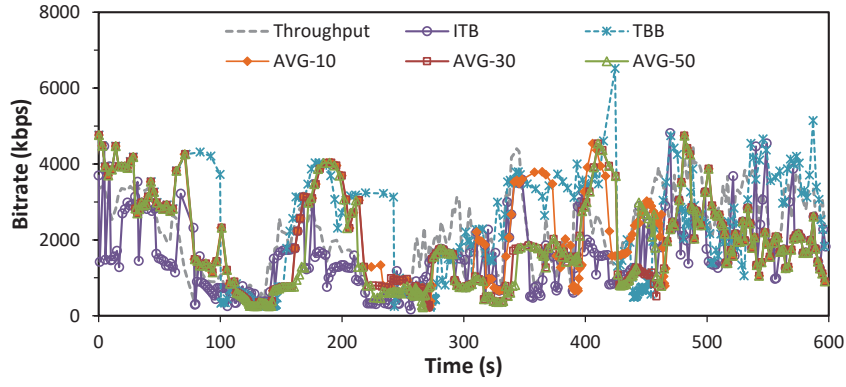
Figure 3.4: The bandwidth trace used in the complex bandwidth scenario.

it is very difficult to use the curves of segment bitrates to compare the adaptation methods. From Fig. 3.5b, it is obvious that the TBB method's behavior is similar to that in the simple bandwidth case. Usually, this method tries to maintain a high version as long as the buffer level allows. This behavior results in sudden drops of quality (e.g. from version 6 to version 3 at 100s, and from version 5 to version 1 at 240s). Also, the buffer level curve of this method is the worst (Fig. 3.5c), as in the previous scenario.

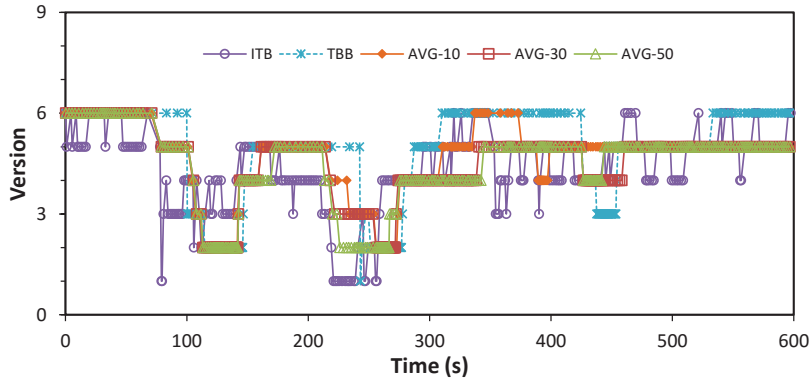
As for the ITB method, its bitrate curve closely follows the throughput curve, resulting in a highly fluctuating version curve with many switches, including switches of large degrees. Anyway, this method has the most stable buffer level curve as seen in Fig. 3.5c.

Meanwhile, our method provides both quality stability and buffer stability. The version curves of our method (Fig. 3.5b) have no sudden switches. Moreover, when the throughput decreases, the selected version of our method is always higher than or equal to those of the other methods. The buffer level curve of our method is not as stable as that of the ITB method; however, it is always higher than that of the TBB method. Compared to the TBB method, our method is more conservative in switching-up and less conservative in switching-down.

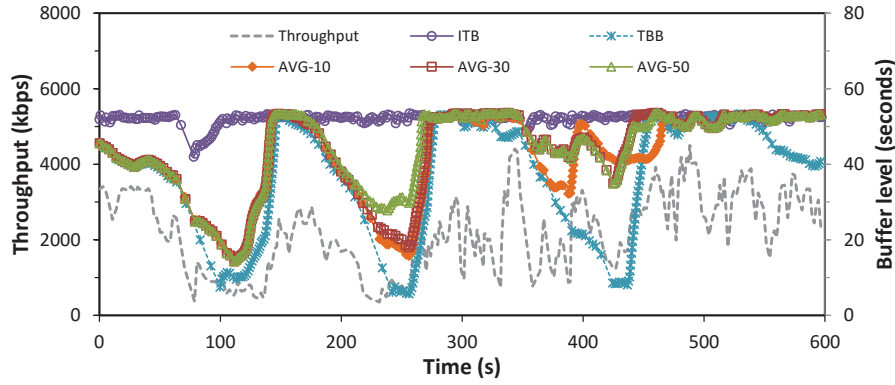
Some statistics of the adaptation results are provided in Table 3.3. The statistics are related to bitrate, requested version, version switch, and buffer level. In terms of bitrate, the ITB method has the lowest average bitrate. Its quality is also very fluctuating with many switches and large switch degree/deviation. The TBB method has the highest average bitrate since this method tends to select and maintain the best possible quality. However, its average version is interestingly lower than that of our AVG-10 option (4.19 vs. 4.30). Even, the ITB method has



(a) Segment bitrate



(b) Requested version



(c) Resulting buffer level

Figure 3.5: Adaptation results of the three methods in the complex bandwidth scenario with “Sony Demo” video.

very low average bitrate, but its average version is higher than that of the TBB method. In fact, the average version values of all methods are very similar (about 4.2 or 4.3). Among the three options, the AVG-10 option is the most aggressive as it always tries to request higher versions; however, this causes a little more switches than other options. These points will be explained

Table 3.3: Statistics of the reference methods and the proposed method with three different options for “Sony Demo”. Here STD, is the standard deviation.

Statistics	ITB	TBB	AVG-10	AVG-30	AVG-50
Average bitrate (kbps)	1555.2	1951.5	1777.7	1632.0	1637.1
Average version	4.29	4.19	4.30	4.18	4.16
Maximum version	6	6	6	6	6
Minimum version	1	1	2	2	2
Number of switches	94	18	17	15	15
Maximum switch degree	4	4	1	1	1
STD of switch degrees	0.56	0.37	0.23	0.22	0.22
Minimum buffer level (s)	42	5.2	15.1	14.3	14.4
STD of buffer levels (s)	1.59	15.31	11.49	11.41	10.43

and discussed further in the next part.

Moreover, our method provides the smallest values in terms of the number of switches and the degree of switches. In addition, the minimum version of our method is higher than the other methods. The minimum value and standard deviation (STD) of buffer level of our method are also much better than those of the TBB method. For more information about the buffer, the cumulative distribution functions (CDF) of the buffer levels are provided in Fig. 3.6a. Again, it is evident that the buffer of our method is much more stable than that of the TBB method.

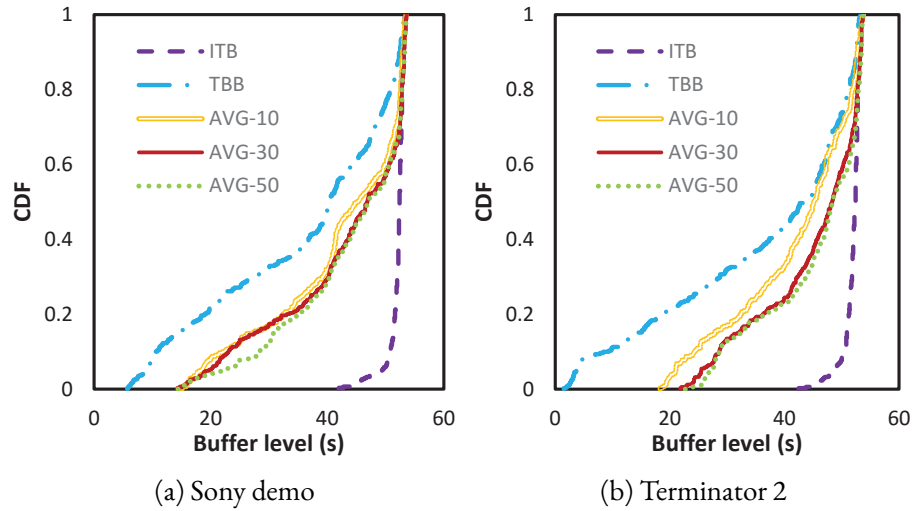


Figure 3.6: Cumulative distribution functions (CDF) of buffer level in the complex bandwidth scenario.

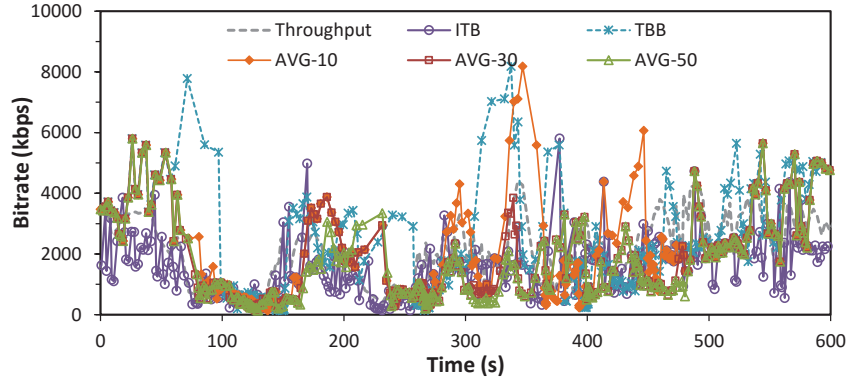
Table 3.4: Statistics of the reference methods and the proposed method with three different options for “Terminator 2”. Here, STD is the standard deviation.

Statistics	ITB	TBB	AVG-10	AVG-30	AVG-50
Average bitrate (kbps)	1544.3	1926.2	1962.9	1779.4	1691.3
Average version	4.51	4.34	4.61	4.55	4.46
Maximum version	6	6	6	6	6
Minimum version	1	1	2	2	2
Number of switches	128	17	18	12	12
Maximum switch degree	3	3	1	1	1
STD of switch degrees	0.63	0.34	0.24	0.20	0.20
Minimum buffer level (s)	42.5	1.5	18.3	21.8	22.7
STD of buffer levels (s)	1.64	16.44	10.43	9.22	8.98

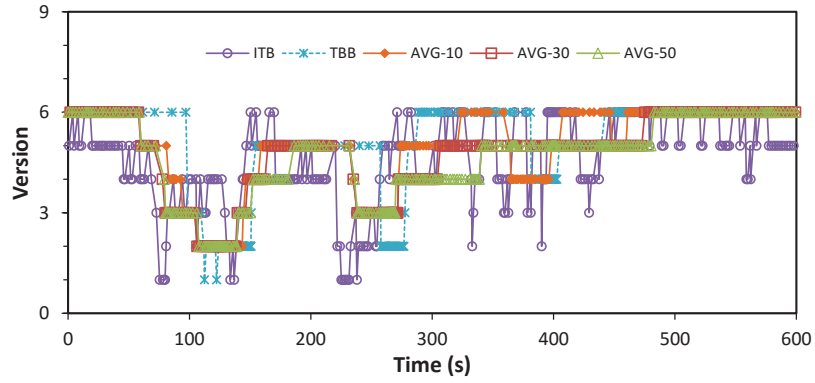
Figure 3.7 shows the experimental results for Terminator 2 video, where similar behaviors of the methods can be found. Our method again provides a smoother version curve and a more stable buffer level curve than the TBB method. The aggressiveness of AVG-10 can also be seen in Fig. 3.7b. For example, during 330s ~ 370s, the AVG-10 option shortly increases the quality to version 6 and then reduces the quality to version 5 and version 4. Meanwhile, the other two options still request version 5 during the same interval.

The statistics of the adaptation results for Terminator 2 video is provided in Table 3.4. With this video, the average version values of our method are all higher than that of the TBB method. Especially, the average bitrate of the AVG-10 option is higher than that of the TBB method. This is because version bitrates of Terminator 2 are smaller than those of Sony Demo, so the AVG-10 option can easily reach the highest possible quality. In addition, the other parameters also show that our method provides a smoother quality and a more stable buffer than the TBB method.

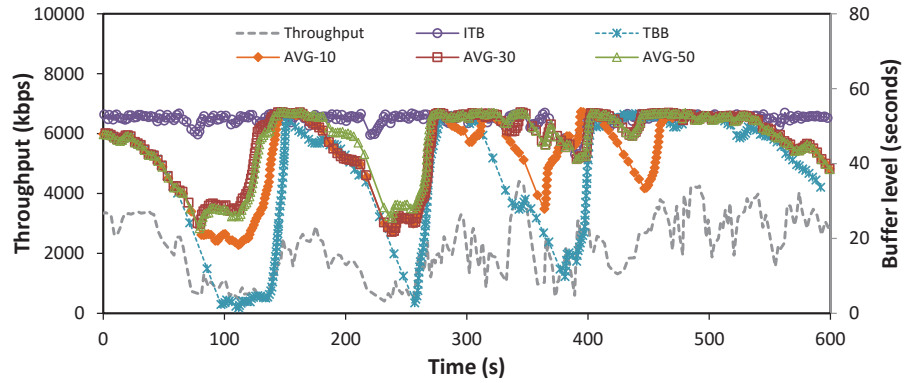
Besides, the CDFs of buffer levels (Fig. 3.6b) confirm the stability of buffer level of our method. Fig. 3.6b also shows that the buffer levels of the AVG-30 and AVG-50 options are a little better than that of the AVG-10 option. This is because the AVG-10 option is more aggressive than the other options. Especially, compared to the case of Sony Demo video (Fig. 3.6a), the buffer of the TBB method is worse while the buffer of our method is better. This is actually due to the characteristics of the Terminator 2 video as discussed in the next part.



(a) Segment bitrate



(b) Requested version



(c) Resulting buffer level

Figure 3.7: Adaptation results of the three methods in the complex bandwidth scenario with “Terminator 2” video.

3.3.4 Discussions

From the above experimental results, we can see some interesting points. First, in some cases the average bitrate of the TBB method is higher than that of our proposed method, while

its average version is lower than that of our method. This can be explained by the fact that sometimes the instant bitrate is very high. So, having some more segments of high bitrates, especially those belonging to the top version, will significantly increase the average bitrate. In case of Terminator 2 video, which has lower version bitrates than Sony Demo video, the average bitrate of TBB method is not better than that of our method.

Meanwhile, in low-throughput periods, the selected versions of the TBB method are lower than those of our method, resulting in a lower value of average version. So, as the bitrate of VBR video is highly fluctuating, the parameter of average bitrate should not be the main indicator to evaluate the performance of VBR video streaming (as in [110]); rather, the parameter of average version should be considered. This is different from CBR video streaming, where the average bitrate is always of high interest.

The results of our adaptation method show that a normal buffer size (i.e. similar to that in CBR video streaming) is already enough to cope with strong bitrate variations of VBR videos. Also, a representative bitrate for each version is important for VBR videos. Even the ITB method benefits a lot from learning the instant bitrate. Though having a highly fluctuating version curve, this method can be used for live streaming thanks to its very stable buffer. On the other hand, the TBB method does not consider bitrate values, and thus poorly handles the strong variations of VBR video. For example, although Terminator 2 video has lower version bitrates than Sony Demo video, the buffer stability of the TBB method in the case of Terminator 2 video is worse than in the case of Sony Demo video. This is just because the version bitrates of Terminator 2 video are extremely varying (with repeated and quick changes from a high value to a low value). Whereas, the performance of our method is not degraded in the case of Terminator 2 video.

The three options of our method are similar in terms of average version values. Yet, among the three options, the AVG-10 option seems to be more aggressive in switching up and maintaining a high quality. This is because the representative bitrate of this option is more “local” and can quickly detect low-bitrate segment intervals, where the client can switch to a higher version when the throughput allows. Such property can increase the average version; however,

that also causes more switches as seen in the statistics of the AVG-10 option. The above statistics show that AVG-30 and AVG-50 options have nearly the same performance with good quality stability and good buffer stability.

In general, it is shown that our method is more effective than the reference methods in providing stable streaming quality, with smooth version transitions even when the available bandwidth dramatically drops. If a user prefers streaming with the highest possible quality, more “local” representative bitrates should be used. On the other hand, if the user prefers a stable streaming session with fewer version switches, more “global” representative bitrates should be used.

It should be noted that, actually, the quality of a CBR video is “variable” and the quality of a VBR video is “constant”. Meanwhile, as shown in [108], it is totally feasible to provide a quasi-CBR streaming service from VBR video data sets. The above results also show that, with the simple method of representative bitrate estimation, we do not need to know exactly bitrates of all VBR video segments even though the instant bitrate is extremely varying. This means that VBR videos and the client-based adaptation method proposed in this chapter can be easily to be deployed in DASH-compliant streaming systems without the need to modify the current standard specification.

3.4 Summary

In this chapter, we have presented an adaptation method for VBR videos and evaluated the method in the mobile streaming context. Our detailed contributions of this VBR-related work are as follows:

1. To cope with strong variations of video bitrate, we employed a local average bitrate as the representative bitrate of a version.
2. A buffer-based algorithm was proposed to conservatively adapt video quality by taking into account a smoothed throughput estimate, the representative bitrate, and even the instant bitrate.

3. The experimental results show that our method can provide smooth video quality and stable buffer level, even under very strong variations of bandwidth and video bitrates. Hence, this study helps enable VBR video streaming with high video quality over mobile networks.

4

Adaptive Streaming over HTTP/2



As mentioned earlier, the target delivery protocol of most existing HAS systems is HTTP/1.1, where every segment is delivered using a request-response pair [10]. Usually, all segments have the same duration, which is typically from 2 to 10 seconds [108]. If the segment duration is short, the client can react quickly to network variations and select the bitrate for a high video quality. However, the use of short segment durations results in a large number of HTTP requests, which in turn causes high overheads (e.g., in terms of energy, processing, bandwidth) for clients, servers, and network nodes [140, 141].

The recently ratified HTTP version 2 (HTTP/2) provides a new feature called “server push”, which enables the client to send one request and then receive multiple objects [142]. Hence, short segment durations can be used in HAS systems without increasing the number

of HTTP requests (by asking the server to respond to a request with multiple consecutive segments). Recently, MPEG has started working on an extension of its HAS standard, where an HTTP/2-based request can specify the number of requested segments [143]. In the literature, the server push feature with a fixed number of pushed segments per request has been investigated (e.g., in [140, 141]). Among existing studies, our previous studies in [144, 145] are the first ones that raise the need to adaptively decide the number of pushed/requested segments in each request to balance the requirements of low request-related overhead and buffer stability. To the best of our knowledge, no previous work has taken into consideration bitrate variations. It should be noted that in mobile networks, as the throughput is fluctuating, the above methods may result in large bitrate reductions and such rate-switching events definitely reduce the quality of service [96].

In this work, we extend our preliminary work in [145] and additionally consider bitrate variations. Instead of selecting an *adaptation pair* (R, N) of bitrate R and number of pushed segments N for only one next request, we decide adaptation pairs for some future requests. Since the proposed method considers the bitrate trend in the future, it can provide users with gradual bitrate transitions. More specifically, to find the adaptation pairs for the next requests, we examine all possible changes of bitrate R and number N . The optimal option, which balances the requirements of buffer stability, gradual down-switching and low request-related overhead, is then selected. Experiments using variable throughput conditions of mobile networks show that our method can yield gradual bitrate transitions and a low number of requests while keeping the buffer level stable.

The rest of the chapter is organized as follows. We provide related work in Section 4.1. Section 4.2 presents the adaptation problem of video streaming over HTTP/2. After that, our proposed method is given in Section 4.3. We present the experimental results and discussions in Section 4.4. Finally, Section 4.5 summarizes the chapter.

4.1 Related work

4.1.1 HTTP/2 and Server push feature

In 2015, the new HTTP/2 standard was published [14] and it is now used by 15.1% of all the websites [146]. In an HTTP/2-based system, an HTTP connection between a client and a server contains multiple *streams*, each is further divided into *frames*. It should be noted that both terms *stream* and *frame* are specific to the HTTP/2 standard [14] and they are distinguished from “video stream” and “video frame”. To avoid confusions in this chapter, we use two terms stream and frame to indicate HTTP/2 stream and HTTP/2 frame, respectively. Essentially, a stream conveys an object (e.g., one video segment), which could be broken into one or more DATA frames. Besides DATA frames, the stream may include other frames such as HEADERS frame that contains the information of the current stream. More details about HTTP/2 can be found in [14].

By using frames and streams in communication, HTTP/2 offers some prominent features, which may be useful for video streaming [20]. Among HTTP/2 features introduced in [14], we focus on *server push* feature. By means of the server push feature, the server can speculatively push a segment to the client prior to receiving the request based on the client’s previous request. To make the client aware of the coming response, the server first sends the so-called PUSH_PROMISE frame in a client-initiated stream, which contains the information of the new stream that carries the segment. Then, the server initiates the stream and sends the associated segment without the client’s acknowledgment of the PUSH_PROMISE frame to reduce the content delivery time.

4.1.2 HTTP/2-based adaptation methods

In the literature, the work in [140] is the first one to exploit HTTP/2 features in context of live streaming. The authors introduce the Push-N adaptation method, where N segments are pushed per request. The Push-N method has been then investigated for low request-related overhead [141], power efficient mobile streaming [147], and SVC (scalable video encoding) video streaming [148].

In [144, 145], we for the first time propose a HTTP/2 push-based method that adaptively decides the number of pushed/requested segments in each request to balance the requirements of low request-related overhead and buffer stability. In [149], the number of pushed segments is adapted to avoid the “over-push” problem, in which network resources are wasted when a user decides to stop watching a video after checking the first few seconds. However, the main drawback of [149] is that it only considers stable network conditions.

Recently, the full-push approach is proposed in [20], where the server periodically pushes new segments. If the client wants to change the video bitrate, it will send a bitrate update request to the server. Otherwise, the server pushes the next segment with the same video bitrate as the previous segment. Because the server always pushes a segment right after this segment is fully generated, a large number of pushed segments may be queued in the network if the throughput drastically drops. To this problem, in the later work [99, 150], the authors present the full-push adaptation method with the client’s acknowledgments, which limits the number of concurrently pushed segments. It should be noted that the full-push approaches presented in [20, 99, 150] require server modifications to handle the pushed video bitrate.

4.2 Problem description

In this section, we present the adaptation problem of adaptive streaming over HTTP/2. Some notations and their definitions are shown in Table 4.1. Again, for clear explanation purpose, we use a specific set of notations in each chapter.

4.2.1 Adaptation space

Suppose that a video is subdivided into small video segments with the same duration of τ seconds. Also, the providers offer the client a set \mathfrak{R} of K bitrate versions $\mathfrak{R} = \{R^k | k = 1, 2, \dots, K\}$. The bitrate becomes lower when the version index k decreases.

At certain time t_i right after fully receiving all segments of the current (or last) request i , the

Table 4.1: Notations and definitions used in this chapter.

Notation	Unit	Definition
τ	second	the media segment duration.
t_i	second	the time instance right after the client completely receives all segments of request i .
T_i	second	the instant throughput obtained at time t_i .
T_i^s	second	the smoothed throughput obtained at the time t_i .
T_i^e	second	the estimated throughput at time t_i for the future segments.
K		the number of available video versions.
\mathfrak{R}		the set of available bitrates $\mathfrak{R} = \{R^1, R^2, \dots, R^K\}$.
\mathcal{M}		the number of push strategies.
\mathfrak{N}		the set of push strategies $\mathfrak{N} = \{1, 2, \dots, \mathcal{M}\}$.
μ		the safety margin, used to decide the bitrate in (4.8) and (4.16).
(R_j, N_j)		the adaptation pair decided for request j , it contains bitrate R_j and number of pushed segments N_j .
B_i	second	the measured buffer level at time t_i .
B_j^e	second	the estimated buffer level after the client fully receives all segment of request j ($j > i$).
B_{tar}	second	the buffer target level, it is also the buffer size.
B_{min}	second	the minimum buffer threshold in the rate decrease case.
\mathbb{S}		the sequence of adaptation pairs for some future segments.
L		the number of adaptation pairs in sequence \mathbb{S} .
\mathfrak{I}		the set of sequence candidates in the rate decrease case.

client will decide a sequence \mathbb{S} of L adaptation pairs for the next L requests:

$$\mathbb{S} = \left\{ (R_j, N_j) \mid j = i + 1, i + 2, \dots, i + L \right\}, \quad (4.1)$$

where $R_j \in \mathfrak{R}$ and $N_j \geq 1$. Note that the adaptation pair (R_i, N_i) that has already been decided for the last request i is considered as the origin of sequence \mathbb{S} .

We assume that the client has a set \mathfrak{N} of \mathcal{M} push strategies $\mathfrak{N} = \{1, 2, \dots, \mathcal{M}\}$. Hence, there are totally $\mathcal{M} \times K$ options for each adaptation pair (R_j, N_j) .

4.2.2 Buffer level estimation and Adaptation problem

An important objective of adaptation methods is to provide seamless streaming. Therefore, sequence \mathbb{S} should be decided so that the buffer level is not depleted during the session. For this purpose, we estimate the buffer level in the near future, given the current buffer level B_i and estimated throughput T_i^e . Note that based on the context, the estimated throughput T_i^e could be either instant throughput T_i or smoothed throughput T_i^s , where the smoothed throughput is computed as follows:

$$T_i^s = \begin{cases} (1 - \delta) \times T_{i-1}^s + \delta \times T_i & \text{if } i > 1, \\ T_1 & \text{if } i = 1, \end{cases} \quad (4.2)$$

where δ is a weight in the range $[0,1]$.

Let us denote B_j^e the buffer level measured right after the client fully receives all segments of request j ($i < j \leq i + L$). We compute buffer level B_j^e from the previous buffer level B_{j-1}^e as follows. With new N_j segments, the buffer level increases $N_j \times \tau$ seconds. However, the client also spends $N_j \times (\tau \times R_j) / T_i^e$ seconds on downloading these segments. Therefore, the estimated buffer level B_j^e is computed by

$$B_j^e = \begin{cases} B_{j-1}^e + \Delta_j & \text{if } j > i + 1, \\ B_i + \Delta_j & \text{if } j = i + 1, \end{cases} \quad (4.3)$$

where Δ_j is the change of buffer level given request j :

$$\Delta_j = N_j \times \tau \times \left(1 - \frac{R_j}{T_i^e}\right). \quad (4.4)$$

So, to avoid buffer underflows, we have a buffer constraint for selecting sequence \mathbb{S} as follows:

$$B_j^e > B_{min}, \quad i < j \leq i + L, \quad (4.5)$$

where B_{min} is a predefined buffer threshold.

All sequence options of sequence \mathbb{S} that satisfy condition (4.5) are gathered into a set \mathfrak{I} of sequence candidates. In the following section, we present our solution to select the optimal sequence from set \mathfrak{I} .

4.3 Adaptation solution

This part presents our method in details. The proposed method includes the following distinguishing features:

- Since the method estimates the trend of buffer level in the near future and selects a sequence from set \mathfrak{I} , it can avoid buffer underflows even under the strong variations of connection throughput.
- When the throughput drops, the method considers the requirement of quality smoothness and balances it with the requirements of buffer stability and low request-related overhead.
- When the throughput is increased, as the quality and the buffer level are not negatively affected, the method fast increases the bitrate to quickly improve video quality. Additionally, the highest possible number of pushed segments is decided to reduce request-related overheads.

4.3.1 General adaptation process

Based on the relation between the current bitrate R_i and the instant throughput T_i , we divide our method into two cases: Rate decrease (when $R_i > T_i$) and Rate increase (when $R_i \leq T_i$). In both cases, our basic idea to decide sequence \mathbb{S} is to balance the requirements of buffer stability, gradual down-switching and low request-related overhead. The detailed algorithms for the two sub-cases are presented in Subsections 4.3.2 and 4.3.3.

Assume that the request sequence is decided at time t_i . In an ideal condition, the client simply sends the requests based on the current (decided) sequence \mathbb{S} until the final request $i +$

Algorithm 4.1: Request selection at time t_j

Input : Current sequence \mathbb{S} decided at time $t_i (i < j)$, current buffer level B_j .
Output: Adaptation pair (R_{j+1}, N_{j+1}) for request $j + 1$.

```

1 begin
2   if  $B_j > B_{min}$  then
3     if the client has requested the final adaptation pair of sequence  $\mathbb{S}$  or
        $|B_j^e - B_j| > \tau$  then
4        $i \leftarrow j$ ;
5       Select new sequence  $\mathbb{S}$ ;
6     else
7       Continue using current sequence  $\mathbb{S}$ ;
8     Decide pair  $(R_{j+1}, N_{j+1})$  of  $\mathbb{S}$  for request  $j + 1$ ;
9   else
10    Abort current sequence  $\mathbb{S}$ ;
11    Decide pair  $(R^1, \mathcal{M})$  for request  $j + 1$ ;

```

L. However, at certain time, if the buffer level is reduced below threshold B_{min} , the current sequence \mathbb{S} is aborted immediately, the client then selects pair (R^1, \mathcal{M}) for the next request to quickly improve the buffer level.

In addition, during the interval of the current sequence \mathbb{S} , if the throughput changes, leading to a significant buffer variation, it is also reasonable to redetermine sequence \mathbb{S} . In our method, we decide a new sequence at time t_{j^*} ($j^* > i$) if the mismatch between the estimated buffer $B_{j^*}^e$ and the actually-measured buffer level B_{j^*} at that time exceeds one segment duration, i.e.

$$|B_{j^*}^e - B_{j^*}| > \tau. \quad (4.6)$$

Note that the sequence could be redetermined by either the rate decrease case or the rate increase case, depending on the relation between the bitrate and the throughput at time t_{j^*} .

In summary, the general process for pair selection at time t_j , given the current sequence \mathbb{S} (decided at the previous time t_i), is provided in Algorithm 4.1. Note that this process is invoked at every request by the client.

4.3.2 Rate decrease case ($R_i > T_i$)

In order to quickly capture throughput behavior in the rate decrease case, we use the instant throughput T_i in estimating the throughput:

$$T_i^e = T_i. \quad (4.7)$$

As the throughput is higher than the current bitrate, the final bitrate R_{i+L} should be lower than the estimated throughput T_i^e . Usually, a safety margin μ in the range $[0, 1]$ is used to compute the final bitrate R_{i+L} :

$$R_{i+L} = \max \left\{ R \mid R \in \mathfrak{R} \wedge R < (1 - \mu) \times T_i^e \right\}. \quad (4.8)$$

To find an optimal sequence \mathbb{S} , we introduce a cost function, which is the tradeoff among request-related cost C_{req} , bitrate smoothness-related cost C_{smt} , and buffer-related cost C_{buf} :

$$C = \alpha \times C_{req} + \beta \times C_{smt} + \gamma \times C_{buf}, \quad (4.9)$$

where α , β and γ are tradeoff parameters.

Since the request-related overhead is inversely proportional to the number of pushed segments, we propose a request cost function based on the average value of the requested segments throughout a sequence:

$$C_{req} = \frac{1}{\frac{1}{L} \times \sum_{j=i+1}^{i+L} N_j}. \quad (4.10)$$

The smoothness cost is a function with regards to quality changes along the sequence, it is defined by:

$$C_{smt} = \max \left\{ V(R_{j-1}) - V(R_j) \mid i < j \leq i + L \right\}, \quad (4.11)$$

where $V(x)$ indicates the version index of bitrate x . It is noted that C_{smt} is the minimum when the changes of bitrate along the sequence are equal.

The buffer cost should depend on the mismatch between the target buffer level B_{tar} and the estimated buffer level B_{i+L}^e (i.e., the buffer level after using sequence \mathbb{S}). We propose an exponential function to compute the buffer cost C_{buf} :

$$C_{buf} = \exp(B_{tar} - B_{i+L}^e), \quad (4.12)$$

where B_{i+L}^e is computed following (4.3).

So, the client computes the overall cost C for each sequence candidate \mathbb{S} in set \mathfrak{J} and then selects the optimal sequence that has the lowest cost. Due to strong fluctuations of mobile networks, it is reasonable to set the sequence length L small. In practical implementation in Java, we found that with typical parameters (e.g., $L = 3$, $M = 4$, and a set of 17 bitrate versions as in [69]), the execution time for deciding sequence \mathbb{S} is just a few milliseconds and thus does not negatively affect the processing time for adaptation.

4.3.3 Rate increase case ($R_i \leq T_i$)

In this case, the smoothness of quality is also not vital. Therefore, it is reasonable to decide an adaptation pair only for the next request $i + 1$ (i.e., the sequence length $L = 1$). Our idea is to maintain the current bitrate until the buffer level reaches the target buffer level B_{tar} . The detailed adaptation logic is as follows.

To avoid short-term throughput fluctuations, we additionally use the smoothed throughput T_i^s in estimating the throughput:

$$T_i^e = \min \{T_i^s, T_i\}. \quad (4.13)$$

Here, T_i^s is computed by (4.2) with $\delta = 0.125$, which is recommended for smoothing computation [151].

We now divide the rate increase case into two sub-cases depending on the current buffer level B_i . If $B_i < B_{tar}$ (i.e., the first sub-case), the current bitrate R_i is maintained until the buffer

level reaches the target buffer level B_{tar} :

$$R_{i+1} = R_i. \quad (4.14)$$

The number of pushed segments N_{i+1} is computed by:

$$N_{i+1} = \min \left\{ \mathcal{M}, \min \left\{ N \mid B_{i+1}^e \geq B_{tar} \right\} \right\}, \quad (4.15)$$

where B_{i+1}^e is obtained from (4.3).

In the second sub-case ($B_i \geq B_{tar}$), the client immediately switches up to the highest bitrate, which is lower than throughput estimate T_i^e :

$$R_{i+1} = \max \left\{ R \mid R \in \mathfrak{R} \wedge R < (1 - \mu) \times T_i^e \right\}. \quad (4.16)$$

The number of pushed segments N_{i+1} is set to \mathcal{M} in order to minimize request-related overheads:

$$N_{i+1} = \mathcal{M}. \quad (4.17)$$

It should be noted that the existing methods strictly decide the number of pushed segment in advance (e.g., [140]) or based on buffer level only (e.g., [145]). Meanwhile, our method flexibly makes decisions based on both buffer level and connection throughput. Our general procedure in both rate decrease and rate increase cases is summarized in Algorithm 4.2.

4.4 Experiments and Discussions

4.4.1 Experiment settings

In this section, we evaluate our method using a simple bandwidth trace and a complex bandwidth trace obtained from a mobile network. For reference methods, we use the Push-N method used in [140, 141] and the request-buffer-based (RBB) method proposed in [145]. In

Algorithm 4.2: Sequence selection at time t_i

Input : Instant throughput T_i , smoothed throughput T_i^s , current buffer level B_i .
Output: Sequence \mathbb{S} .

```

1 if  $R_i > T_i$  then
2   // Rate decrease case
3    $T_i^e \leftarrow T_i$ ;
4   for each sequence candidate  $\mathbb{S}$  in set  $\mathfrak{I}$  do
5     Compute a set of corresponding buffer levels  $\{B_j^e | i < j \leq i + L\}$ ;
6     Compute request cost  $C$  following (4.9);
7   Select the optimal sequence  $\mathbb{S}$  which has the lowest cost;
8 else
9   // Rate increase case
10   $T_i^e \leftarrow \min \{T_i^s, T_i\}$ ;
11  if  $B_i < B_{tar}$  then
12    Compute  $(R_{i+1}, N_{i+1})$  following (4.14) and (4.15);
13  else
14    Compute  $(R_{i+1}, N_{i+1})$  following (4.16) and (4.17);
15   $\mathbb{S} \leftarrow \{R_{i+1}, N_{i+1}\}$ ;

```

the former method, N segments are sent for each request. Meanwhile, in the latter method, the number of pushed segments is adaptively decided to balance buffer stability and low request-related overhead. Note that these two methods decide the bitrate based on the estimated throughput only. Here, our focus is on the adaptation behavior in the steady stage. In the initial buffering stage, the simple Push- N method with $N = 1$ is employed.

The testbed of our experiments is similar to that of [145], which includes an HTTP/2 web server, an HTTP/2 client and an IP network. The IP network includes a router and wired connections connecting the server and the client. On the server side, an HTTP/2-enable server¹ is installed in Ubuntu 14.04 LTS. A test video is encoded at 17 bitrate versions based a DASH dataset in [69], i.e. $\mathfrak{R} = \{100, 150, 200, 250, 300, 400, 500, 700, 900, 1200, 1500, 2000, 2500, 3000, 4000, 5000, 6000\}$ (kbps). All segments have the same segment duration, which is provided later in each experiment. The client is implemented in Java and runs on a Windows 7 notebook with 2.4 GHz core i5 CPU. Dummynet tool [138], a network emulator, is installed

¹In our experiments, we use a nghttp2 library to setup an HTTP/2 server as well as an HTTP connection. This library can be found at <http://nghttp2.org>.

at the client to emulate the characteristics of mobile networks. The round trip time (RTT) is set to 100ms. The packet loss rate is set to 0%, assuming that the packet loss are already included in the fluctuations of employed bandwidth traces.

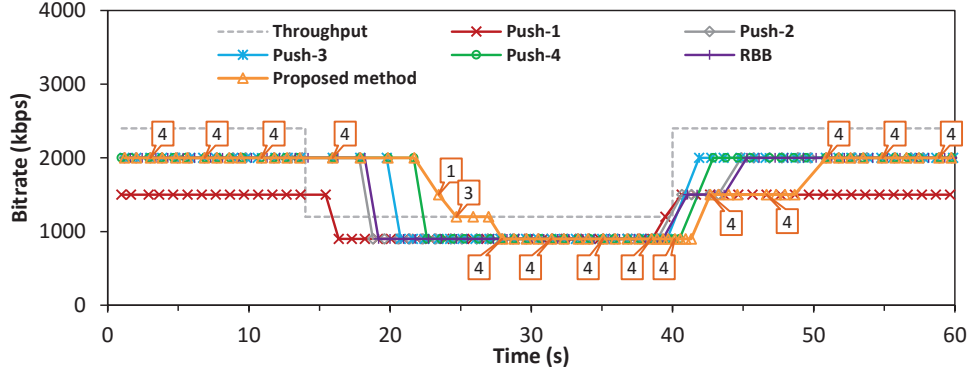
The three evaluated methods are employed with the same target buffer level (or the buffer size) $B_{tar} = 15s$ and the same safety margin $\mu = 0.05$. We implement the Push-N method with 4 options $N = 1, 2, 3$ and 4. The set of push strategies of the two other methods is set to $\mathcal{N} = \{1, 2, 3, 4\}$. The RBB method is employed with tradeoff parameter $\alpha = 0.2$. Our method is implemented with $B_{min} = 3s$, i.e. 3 segment durations with option $\tau = 1s$.

In our method, a streaming provider can adjust the balance between the requirements for low request-related overhead, gradual transitions, and buffer stability by changing the tradeoff parameters α , β and γ . Basically, we fix α and select two others. Given $\alpha = 10$, the contribution of the request-related component to the overall cost C is at most 10 (i.e. when the client decides only one segment per request). Since we want to prioritize the requirement of gradual transitions, β is selected so that the contribution of the smoothness-related cost is higher than that of the request-related cost. With parameter γ , because the buffer-related cost C_{buf} varies in a wide range from 2^0 to $2^{B_{tar}}$, parameter γ should be small to reduce the contribution of the buffer-related cost to the overall cost C . Based on our experience, good empirical values of the parameters α , β , and γ are 10, 13.5, and 0.08, respectively. In our future work, the tradeoff of these factors and other additional factors (e.g. minimum bitrate, buffer level variation, etc.) will be investigated.

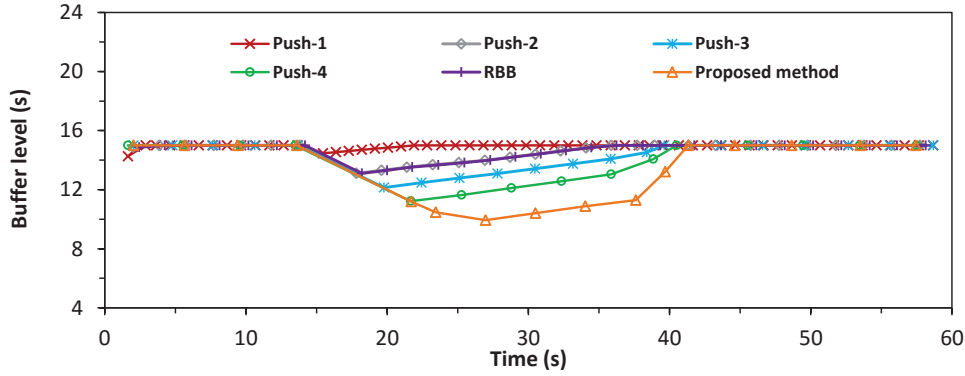
4.4.2 Simple bandwidth scenario

As mentioned earlier, one of the main challenges in HAS is to provide gradual down-switching when the throughput suddenly drops. Thus, we compare the methods when the connection throughput suddenly declines from 2400kbps to 1200kbps as in Fig. 4.1a. In this experiment, the segment duration is set to 1 second, which is lower than that of popular HAS systems (i.e. typically from 2 to 10 seconds [108]). Fig. 4.1 shows the adaptation results of the three methods.

The Push-N and the RBB methods aggressively switch the bitrate according to throughput



(a) Adapted bitrate



(b) Resulting buffer level

Figure 4.1: Adaptation results of the three methods in simple bandwidth scenario.

variations. As a consequence, they all result in a large bitrate change. Especially, the Push-1 option has the smallest bitrate curve since this option does not leverage the push feature of HTTP/2. From Fig. 4.1a, we also observe that the time to change the bitrate of the RBB and Push-N methods are dependent on the number of pushed segments, which is decided around time $t = 13$ s (before the throughput drops). As seen in Fig. 4.1, the lower number of pushed segments, the earlier the bitrate is changed and so, the more stable the buffer level becomes.

As for our proposed method, besides the number of pushed segments and buffer level, it also considers bitrate variations. Fig. 4.1a shows that the proposed method gradually changes the bitrate from 2000kbps to 900kbps. Specifically, at time $t = 22$ s after detecting a throughput drop, the method decides the sequence $\mathbb{S} = \{(R_j, N_j)\} = \{(1500\text{kbps}, 1); (1200\text{kbps}, 3); (900\text{kbps}, 4)\}$ for the next 3 requests. It is seen that in our method, both bitrate and number of pushed segments are adaptively decided. The results show that our method requires 3

requests for downloading the next 8 segments (i.e., around 3 segments per request), provides gradual down-switching, and has good buffer levels (i.e. higher than 9s). That means, when the throughput is reduced, our method can balance between gradual down-switching, buffer stability and number of requests.

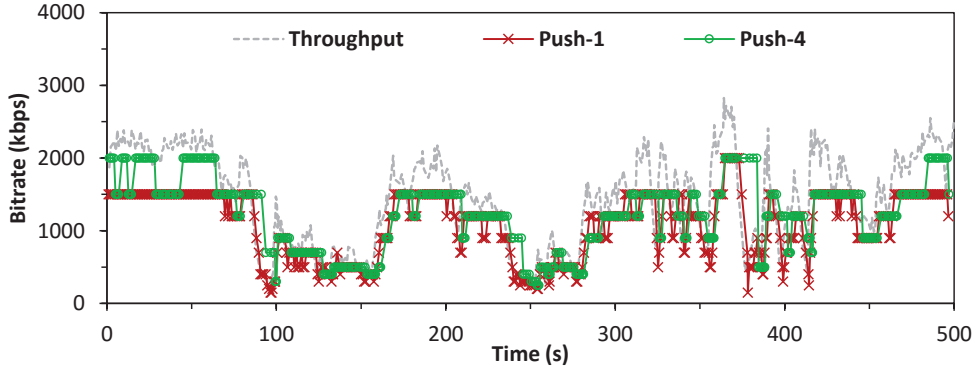
4.4.3 Complex bandwidth scenario

This part investigates the performance of the three methods using a time-varying bandwidth trace obtained from a mobile network [133]. As shown in Fig. 4.2a, the connection throughput widely varies from 300kbps to 2800kbps. All other settings are the same as before.

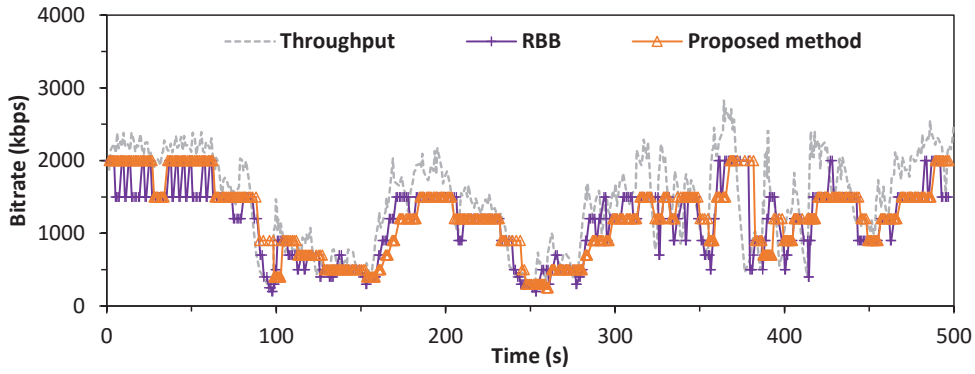
Fig. 4.2 shows the adaptation results of the three methods. Because the Push-2 and Push-3 options have similar performances to that of the Push-4 option, they are not included in Fig. 4.2 for the sake of clarity, but they are still shown in our later statistics. As seen in Figs. 4.2a and 4.2b, the bitrate curves provided by both Push-N and RBB methods widely vary according to throughput fluctuations. Meanwhile, our proposed method provides bitrate stability and gradual down-switching. For example, during 60s ~ 90s, our method maintains a bitrate of 1500kbps even the throughput is fluctuating. At time $t = 375s$, when the throughput suddenly drops, it changes the bitrate gradually from 2000kbps to 700kbps while other methods directly jump to a bitrate around 400kbps. Fig. 4.2c shows that the Push-N method with option $N = 1$ has the most stable buffer level curve while our method's buffer level curve is also good. The minimum buffer level of our method (3.2s) is still higher than the predefined threshold B_{min} .

Some statistics of the adaptation results are provided in Table 4.2. The statistics are related to adapted bitrate, buffer level, number of requests and version decreases. Note that version decreases are used instead of bitrate decreases because a change of 1000kbps at a high bitrate level may be not as severe as a change of 500kbps at a low bitrate level.

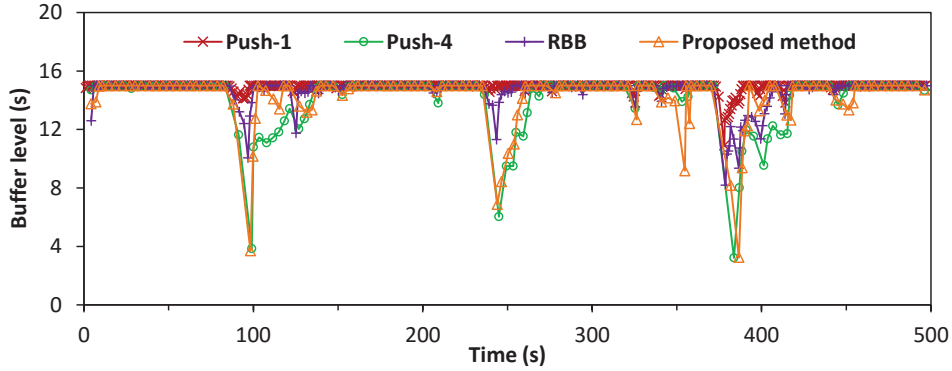
As expected, the Push-1 option has the lowest average bitrate (1081kbps) as the client has to wait one RTT before starting receiving every segment. The statistics of buffer level demonstrate that all adaptation methods can avoid buffer underflows. Although giving the highest minimum buffer level (10.9s), the Push-1 option results in the highest number of requests (500) and



(a) Adapted bitrate of the push-N method with $N = 1$ and 4



(b) Adapted bitrate of the RBB and proposed methods



(c) Resulting buffer level

Figure 4.2: Adaptation results of the three methods in complex bandwidth scenario.

thus a high request-related overhead. Meanwhile, the RBB method dynamically decides the number of pushed segments based on the buffer level. The adaptation results show that the RBB method can balance between buffer stability and low request-related overhead. From the third row of Table 4.2, we see that our method requires only 131 requests during a whole session, which is comparable to 125 requests of the Push-4 option. This good result is because our

Table 4.2: Statistics of adaptation results in complex bandwidth scenario.

Statistics	Push-N				RBB	Proposed method
	N=1	N=2	N=3	N=4		
Average bitrate (kbps)	1081	1148	1162	1184	1152	1180
Minimum buffer level (seconds)	10.9	8.9	6.2	3.2	8.2	3.2
Number of requests	500	250	167	125	252	131
Number of version decreases	74	62	38	32	60	21
Average version decrease	1.3	1.3	1.5	1.4	1.4	1.3
Maximum version decrease	6	5	5	5	5	3

method decides the number of pushed segments based on not only the buffer level but also the throughput trend (increase or decrease).

In terms of version decreases, the Push-1 option has the highest number of version decreases (up to 74) while that of our method is the lowest (21). Our method's result is even 30 percent lower than that of the second (32). Moreover, our proposed method provides a small average value (1.3) and the smallest maximum value (3) of version decreases. That means, our method reasonably switches the bitrate down to avoid large version decreases. The above statistics confirm that our proposed method can provide a high average bitrate, buffer stability, a low number of requests as well as gradual transitions when the throughput is reduced.

As already mentioned, HTTP/2-based streaming supports small segment durations. Therefore, we investigate the performances of the three methods with a smaller segment duration. Table 4.3 shows the statistics for the case that the segment duration is reduced to 500ms. It is seen that all the methods have better buffer behaviors but higher numbers of requests in comparison with those for the case of 1-second segment duration. Among the three methods, our method has the highest average bitrate (1218kbps). In addition, the proposed method also provides the smoothest down-switching (with only 48 version decreases and at most 2 version decreases per switch) while still providing a very low number of requests (264).

Table 4.3: Statistics of adaptation results in complex bandwidth scenario with a segment duration of 500ms.

Statistics	Push-N				RBB	Proposed method
	N=1	N=2	N=3	N=4		
Average bitrate (kbps)	1039	1119	1138	1164	1132	1218
Minimum buffer level (s)	12.3	10.9	9.9	8.6	10.7	8.1
Number of requests	1000	500	334	250	371	264
Number of version decreases	155	87	68	54	73	48
Average version decrease	1.1	1.2	1.3	1.3	1.3	1.0
Maximum version decrease	6	6	5	5	6	2

4.4.4 Discussions

From the above experiments, it can be seen that although the Push-N method can yield buffer stability (when $N = 1$) or low request-related overhead (when $N = 4$), setting the value of N in the varying conditions of connection throughput is not easy. The RBB method can decide the number of pushed segment for each request to balance the two mentioned factors. However, the main disadvantage of these methods is that they decide the bitrate regardless of bitrate switching and thus leading to quality instability and sudden quality decreases.

To overcome this problem, the proposed method further considers the requirements of gradual down-switching in making decisions. For gradual transitions when switching down the bitrate, our method decides the adaptation pair (R, N) not only for one next request but also for some future requests. The adaptation rule to provide the gradual down-switching, together with the policy of immediate up-switching when the buffer is full, helps our method to have a high average bitrate during a whole session.

Interestingly, although deciding the sequence \mathbb{S} of adaptation pairs for some future segments, our method is not sacrificed by the buffer stability since it considers the buffer trend in the future. During a session, the buffer level is measured after receiving every response; if the estimated buffer levels become inaccurate, our method can redetermine sequence \mathbb{S} to adapt to new conditions of the client/networks. This feature enables our method to take advantage of the buffer to better cope with throughput fluctuations.

In our method, the number of request over a streaming session is reduced since the method decides the number of requests depending on both buffer level and throughput trend. If the throughput is reduced, the number of pushed segments for each request is adaptively decided. Otherwise, the client can select the highest option since the buffer and the video quality are not negatively affected. It is different from the other methods where the number of request is set in advance [140, 141] or is based on buffer level only [145].

4.5 Summary

The recently ratified HTTP/2 protocol provides server push feature that helps reduce request related overheads (e.g., in terms of energy, processing, bandwidth) compared to traditional HTTP/1.1-based systems. This chapter has presented a novel adaptation method that not only takes advantage of HTTP/2 but also improves video quality by providing gradual down-switching. In our method, the client considers adaptation for some future segments rather than just the next segment. If the bitrate should be reduced, the client selects the optimal selection to balance the requirements of buffer stability, low request-related overhead and gradual transitions. If the bitrate is to be increased, the method rapidly increases the bitrate to quickly improve video quality. Experimental results show that our proposed method can provide users with high average bitrates, buffer stability and low numbers of requests while outperforming the reference methods in terms of quality switching.

This page intentionally left blank.

5

HTTP Live Streaming with Perceptual Quality Control



IN HAS, a larger buffer provides the client a better ability to deal with throughput fluctuations. Therefore, a good strategy for on-demand streaming is to use a very large buffer size (e.g., up to 30 seconds as in [10, 133, 134]). Nevertheless, for live streaming, even if the client tries to request the lowest bitrate version, the amount of buffered media is still limited because 1) the initial delay should be small and 2) the client can request new segments only if they have been generated on the server side (from a live video source). Due to a small buffer size, the client faces interruptions when the throughput is drastically reduced. In fact, there are a few heuristic methods supporting live streaming services,

where the buffer size is 10 seconds or less.

To avoid interruptions, the client may aggressively change the bitrate according to throughput fluctuations and thus, it cannot guarantee gradual bitrate transitions with a step of one representation. In bitrate adaptation, a bitrate change (large or small) has specific impact on video quality. As the client may largely change the representation, Just Noticeable Difference (JND) [68] that can indicate the perceptual quality change can be used to help the client select which representation (or video bitrate) should be requested. Taking advantage of JND metric, the client may decrease the bitrate by 40% without significantly affecting user experience [24].

Motivated by the above finding, we develop an adaptation method for HTTP live streaming that can provide smooth viewing experience while avoid buffer underflows. For the purpose of buffer stability, we adopt the *future-buffer based approach* introduced in [117] that allows the client to examine all possible changes of the bitrate and corresponding buffer levels in the near future. Then, the client uses JND metric to select a sequence of video bitrates for some next segments. The selected sequence should have the smoothest viewing experience and does not result in any buffer underflows. In our experiments, we evaluate our method in mobile streaming context with a small buffer size of 10 seconds. The experimental results show that our method can jointly provide buffer stability and smooth user experience.

This chapter is organized as follows. In Section 5.1, we first provide related work, including prominent adaptation methods for HTTP live streaming, future buffer based approach, and JND metric for adaptive streaming. We describe our quality-driven method in Section 5.2. Section 5.3 presents experiment results and discussions. Finally, we summarize this chapter in Section 5.4.

5.1 Related work

5.1.1 Adaptation methods for HTTP live streaming

As discussed in Section 3.1, recent bitrate adaptation methods can be roughly divided into two main groups: throughput-based and buffer-based groups. In this section, we remark some

prominent adaptation methods, which effectively support HTTP live streaming.

The throughput-based methods decide the bitrate based on the throughput estimate. The key differences among such methods are the ways to estimate and use the throughput. In the simplest way, the throughput of the last segment i is used as the estimated throughput T_i^e of the next segment. Because the estimated throughput obtained in this way often fluctuates in short-term network variations, our previous work [10] proposed a better throughput-based method that can provide a smooth throughput estimate in this context. Given a throughput estimate, the next requested bitrate could be computed by $T_i^e \times (1 - \mu)$ where μ is a safety margin in the range (0,1). In [67], we show that the instant throughput based method is very effective to maintain buffer stability in live streaming, since it quickly reacts to throughput fluctuations.

The buffer-based methods decide the bitrate mainly based on buffer characteristics. Usually, they take advantage of the throughput estimate as well. In [133, 134], the buffer is divided into multiple ranges with thresholds B_1, B_2, B_3 and B_{max} ($0 < B_1 < B_2 < B_3 < B_{max}$). Depending on the buffer level at a given time, the client will make appropriate actions. Compared to throughput-based methods, buffer-based methods provide smoother video bitrate curves in on-demand streaming. However, they may result in a unacceptably large number of buffer underflows if the buffer size is small [67].

In HAS, because connection bandwidth is highly fluctuating, video bitrate (and so video quality) should be adjusted quickly. Therefore, Just Noticeable Difference (JND) is used to describe how the changes of bitrate (and video quality) impact on end users' perception [24]. In [152], Thakolsri et al. apply JND concept to help build the utility functions in their QoE-driven resource allocation optimization for the case of multiple streams. In our previous work [67], the adapted videos provided by some typical adaptation methods have been evaluated in terms of perceptual distortion.

5.1.2 Future buffer based approach

The adaptation problem of HAS could be depicted in Fig. 5.1. We assume that the time when the client begins receiving video content is t_0' and the time when the client initiates to play

is t_0^p . From these two points, we have two curves (called *arrival curve* and *playout curve*).

- Arrival curve describes the accumulated data size received by the client at a given time instant.
- Playout curve describes the accumulated data size consumed by the client at a given time instant.

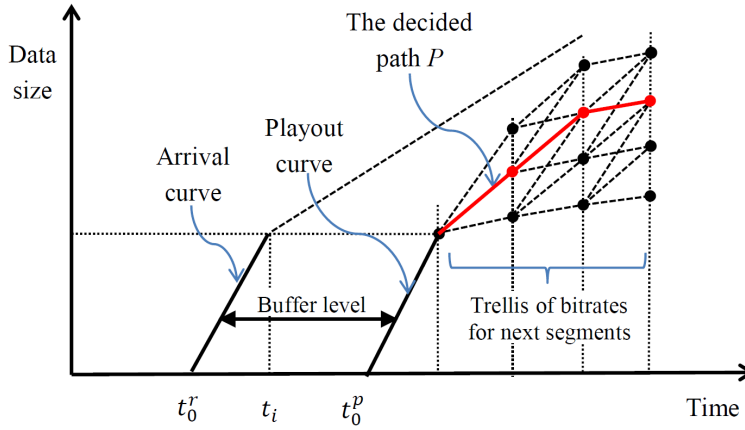


Figure 5.1: Illustration of arrival curve, playout curve and trellis representation.

As the video content is received and then consumed in blocks, the arrival and playout curves actually increase step by step. However, for simplicity, we just consider the points right after completely receiving (or consuming) media segments. Hence, the arrival curve can be represented by linear sections, of which the slopes represent average arrival rates. Also, the playout curve is composed of linear sections, of which the slopes represent playout rates.

Basically, the horizontal distance between the two mentioned curves is the buffer level in time of the client. When the arrival rate (also average throughput) and playout rate (also bitrate of media segments) are equal, the horizontal distance between these curves is maintained and the buffer is stable.

Suppose that after completely downloading the current segment i at time t_i , the client will select a sequence of bitrates for the next N segments. For each future segment $i+j$ ($1 \leq j \leq N$), we assume that the client has K bitrate options. If bitrate R_{jk} ($1 \leq k \leq K$) is selected, the arrival curve would be extended by a line section of which the slope is the average throughput during

the downloading interval of that segment. Also, the playout curve would be extended by a line section of which the slope represents bitrate R_{jk} . The client obviously has many options in selecting an appropriate sequence of bitrates. The decision problem is therefore represented as a *trellis* in which each possible sequence of bitrates is shown as a path. With this representation of trellis, the client can predict the trend of buffer level in near future, according to the segments' bitrate and throughput estimate. Hence, a good path could be selected to meet certain specific criteria (e.g. minimum buffer level, smallest number of bitrate changes, etc.).

5.1.3 JND metric for Adaptive Streaming

Theoretically, the concept of JND is used to indicate the minimum perceptual difference between two stimuli that can be detected by a human being. This is different from an other popular quality metric, Mean Opinion Score (MOS) [50]. Using MOS, we can measure the video quality of the whole session (good or bad), but this metric does not reflect the impact of quality variations on user experience as JND.

Suppose that x_i and x_j are the physical intensities of stimulus i and stimulus j . When a person perceives stimulus i , the perceptual intensities $\Psi(x_i)$ is actual intensity judged by that person. When intensities x_i and x_j are sufficiently far apart, a user can observe a “*perceptual difference*” between stimulus i and j , which is:

$$J = \Psi(x_i) - \Psi(x_j). \quad (5.1)$$

Denote p the probability that a perceptual difference between the two stimuli is detected. The probability p is used to convert perceptual difference J into an interval scale. In particular, we obtain probability p by forced-choice paired comparison tests, in which each observer is shown a pair of stimuli and asked to select the one with higher (or lower) quality. Then, based on Thurstone's “Law of Comparative judgment” [68], we can obtain perceptual difference J using the relationship $p - J$ below:

$$p = C \left[\frac{J}{\sqrt{2}} \right] \quad (5.2)$$

where $C[\cdot]$ is the accumulative normal probability density function. Obviously, when J becomes larger, probability p increases toward the maximum value of 1. According to the definition in [153], J is equal to one JND unit if probability p is equal to 75% (i.e. when 75% of selections are for the better stimulus and 25% for the other one in a forced-choice test). As JND is an experimental threshold, a difference of 1 JND in practice is “essentially undistinguishable”, and even a difference of 3 JND units is still “not obviously different” [154].

In [24], we have evaluated the perceptual distortion (in JND units) of down-scaled video versions through forced-choice subjective tests. For this study, the same process has been repeated for a higher resolution (i.e. 4CIF). Fig. 5.2 depicts the perceptual distortion w.r.t. the normalized bitrate for five popular video sequences. For each sequence, the normalized bitrate of a version is obtained by dividing its bitrate by the bitrate of the highest quality version. This figure indicates that the relationship between the perceptual distortion and the normalized bitrate is not linear. Specifically, the perceptual distortion increases mostly in low bitrate ranges. Meanwhile, in higher ranges, the original video bitrate could be reduced by 40% without worrying about quality reduction (the degradation is less than 1 JND unit). Based on this finding, we use the JND metric to help the client select the bitrate.

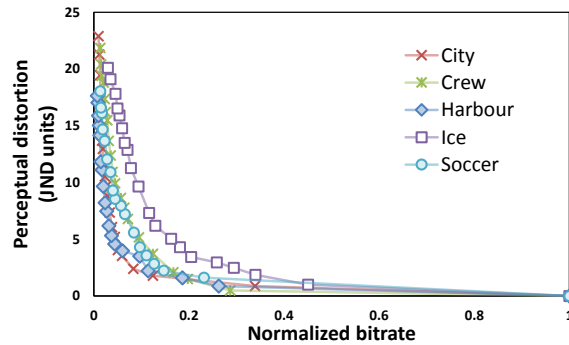


Figure 5.2: The relationship between perceptual distortion and normalized bitrate.

Using Fig. 5.2, the client can determine the perceptual distortion PD from the given video bitrate b and vice versa by lookup table:

$$PD = F_{distortion}[b], \quad (5.3)$$

$$b = F_{bitrate} [PD]. \quad (5.4)$$

In our experiments, City video (with slow motion) and Crew video (with fast motion), are used as our test videos.

5.2 Adaptation solution

5.2.1 Overview of proposed algorithm

Suppose that at time t_i right after successfully receiving the current segment i , the client makes decision on what video bitrates are requested for the next N segments. Some notations along with their definitions used in the study are provided in Table 5.1.

Table 5.1: Notations and definitions.

Notation	Definition
τ	the segment duration
T_i	the instant throughput obtained at time t_i
b_i	the video bitrate requested for the current segment i
β_i	the current buffer level measured at time t_i . Note that $0 \leq \beta_i \leq \beta_{max}$ where β_{max} is the buffer size
(i, j)	the next segment with its index equals to $i + j$ ($1 \leq j \leq N$)
T_i^e	the estimated throughput of next segments at time t_i
$b_{(i,j)}^e$	the expected video bitrate that could be requested for segment (i, j) . Note that this expected bitrate is decided at the time t_i
$\beta_{(i,j)}^e$	the estimated buffer level right after receiving segment (i, j) . This is estimated at time t_i as well
$q_{(i,j)}^e$	the perceptual distortion in JND units corresponding to bitrate $b_{(i,j)}^e$

Path definition

Before delving into the details, we use the term *path* P to denote the sequence of bitrates that could be requested for the next N segments:

$$P = \{b_{(i,1)}^e, b_{(i,2)}^e, \dots, b_{(i,N)}^e\}. \quad (5.5)$$

Here, video bitrate b_i that has been requested for the current segment i can be considered as the starting point $b_{(i,0)}^e$ of path P . Additionally, in order to maintain a stable buffer level, the final expected bitrate $b_{(i,N)}^e$ should be lower than the estimated throughput T_i^e . In this study, based on the knowledge of JND concept, we compute bitrate $b_{(i,N)}^e$ by using a predefined safety margin μ_{jnd} in JND units as follows:

$$b_{(i,N)}^e = F_{bitrate}[F_{distortion}[T_i^e] + \mu_{jnd}] \quad (5.6)$$

Buffer estimation

At each step j ($1 \leq j \leq N$) of path P , we estimate buffer level $\beta_{(i,j)}^e$ based on the previous estimated buffer level $\beta_{(i,j-1)}^e$. With the assumption of a stable throughput in near future, the interval to download segment (i, j) with requested bitrate $b_{(i,j)}^e$ is $\frac{\tau \times b_{(i,j)}^e}{T_i^e}$. Because the contribution of this segment to the buffer is τ , buffer level $\beta_{(i,j)}^e$ can be approximated by the equation below:

$$\beta_{(i,j)}^e = \begin{cases} \beta_{(i,j-1)}^e + \tau * (1 - \frac{b_{(i,j)}^e}{T_i^e}) & , j > 0 \\ \beta_{(i,0)}^e & , j = 0 \end{cases} \quad (5.7)$$

where $\beta_{(i,0)}^e$ is the current buffer level ($\beta_{(i,0)}^e = \beta_i$).

It should be noted that all buffer levels $\{\beta_{(i,j)}^e\}$ have to be higher than a predefined minimum threshold (denoted by β_{min}) in order to avoid buffer underflows:

$$\beta_{(i,j)}^e > \beta_{min} \quad , \forall j = 1..N. \quad (5.8)$$

Path rebuilding

From Eq. 5.7, the client is able to estimate buffer levels in the near future corresponding to its selected bitrate path. At the next segments, if the difference between the actual throughput and estimated throughput T_i^e is small enough, then the existing buffer estimation is acceptable. For this case, the client can simply request the bitrate following the path P .

On the contrary, if the throughput significantly changes (e.g. at segment (i, j^*)), the esti-

mated throughput and estimated buffer level become inaccurate. For this case, a simple way is to rebuild path P with new values of estimated throughput and buffer level at segment (i, j^*) . However, this results in high complexity in making decision. Additionally, the actual throughput in future may fluctuate around existing throughput T_i^e . So, for efficiency, our method only rebuilds path P if the difference between the estimated buffer level $\beta_{(i,j^*)}^e$ and the actual buffer level β_{i+j^*} is higher than 1 second:

$$\left| \beta_{i+j^*} - \beta_{(i,j^*)}^e \right| > 1s. \quad (5.9)$$

The pseudo code of bitrate requesting and path rebuilding is given in Algorithm 5.1.

Algorithm 5.1: Bitrate selection for segment $seg=(i, j + 1)$.	
Input: The current path: P , the current point (i, j) : $\beta_{(i,j)}^e, T_{(i,j)}^e$, the current segment $i+j$: β_{i+j}, b_{i+j} . Output: b_{seg}	
1	if $(P \neq \emptyset)$ and $(\left \beta_{i+j} - \beta_{(i,j)}^e \right \leq 1s)$ then
2	// move to the next point
3	if $j < N$ then
4	$j := j + 1$;
5	else
6	// store the segment that is the new starting point
7	$i := i + j$; // compute the ending point of P following Eq. 5.6
8	$b_{(i,N)}^e = F_{bitrate}[F_{distortion}[T_i^e] + \mu_{jnd}]$
9	// rebuild path P following subsection 5.2.2
10	$P = \{b_{(i,0)}^e, b_{(i,1)}^e, \dots, b_{(i,N)}^e\}$;
11	// restart at the second point
12	$j := 1$;
13	// decide video bitrate for segment seg ;
14	$b_{seg} = b_{(i,j)}^e$;

5.2.2 JND-driven criteria for Path finding

This part presents the way to use JND concept in deciding which appropriate bitrates should be requested in near future. We have the sequence of expected perceptual distortion values cor-

responding to the expected bitrates of path P

$$\mathcal{Q} = \{q_{(i,0)}^e, q_{(i,1)}^e, \dots, q_{(i,N)}^e\} \quad (5.10)$$

where $q_{(i,j)}^e = F_{distortion}[b_{(i,j)}^e]$. It should be noted that $q_{(i,0)}^e$ and $q_{(i,N)}^e$ are known before building path P .

Obviously, the selection of path P should depend on the current buffer level as well as the perceptual distortion of path candidates. In the following part, we subdivide our method into two cases: 1) throughput-decrease, and 2) throughput-increase.

For the throughput-increase case, the client can change the bitrate without causing buffer underflows. So, our method decides to switch the bitrate up with a step of an empirical value (i.e. equivalent to 1.5 JND units). In other words, the video quality is changed gradually.

For the throughput-decrease case, the problem is that requesting a high bitrate (with a gradual quality change) may lead to a long download time while requesting a low bitrate (with a sudden quality change) definitely has negative impact on users. Therefore, our method focuses on balancing the requirements of smooth perceptual distortion and high buffer level. To achieve this objective, we introduce two notions: *distortion cost* and *buffer level cost* for each path candidate.

The *distortion cost*, denoted by C_{dist} , for a given path P is computed as follows:

$$C_{dist}(P) = \frac{\max\{q_{(i,j)}^e - q_{(i,j-1)}^e\}}{q_{(i,0)}^e - q_{(i,N)}^e}, \quad \forall j = 1..N. \quad (5.11)$$

Intuitively, the cost is the lowest and equal to $1/N$ when $q_{(i,0)}^e - q_{(i,1)}^e = q_{(i,1)}^e - q_{(i,2)}^e = \dots = q_{(i,N-1)}^e - q_{(i,N)}^e$. The reason behind the use of maximization criterion is to find the maximum video quality switch between two consecutive segments of path P .

The *buffer level cost*, denoted by C_{buf} , is a function of consumed buffer levels when the client follows the path P . Because the maximum consumed buffer level is $\max\{\beta_i - \beta_{(i,j)}^e\}$, we

compute the buffer level cost as follows:

$$C_{buf}(P) = \frac{\max\{\beta_i - \beta_{(i,j)}^e\}}{\beta_i - \beta_{min}}, \quad \forall j = 1..N. \quad (5.12)$$

The high buffer level cost means, the client need more amount of buffer level to request video bitrate according to path P .

The combined cost is then simply the weighted sum of C_{dist} and C_{buf} :

$$C_{total}(P) = \alpha \times C_{buf}(P) + (1 - \alpha) \times C_{dist}(P). \quad (5.13)$$

The client computes the combined cost for each path P and picks a path that has the lowest combined cost. The factor α here is an empirical threshold in the range $[0, 1]$ that controls the tradeoff between the requirements of high perceptual quality and high buffer level. Currently, we set the path length N to 6. As the trellis in this case is small, we can find the best path by a full-search algorithm.

5.3 Experiments and Discussions

In this section, we evaluate our proposed method using a real bandwidth trace obtained from a mobile network in [133]. Especially, we compare our method to two other adaptation methods: the throughput-based method proposed in [10], and the buffer-based method proposed in [134]. For simplicity, we use the terms “Aggressive method” and “Threshold-based method” respectively to call these reference methods.

5.3.1 Experiment settings

The organization of our test-bed is similar to that of Section 3.3, including three components: a HTTP server (i.e. a Web server), IP networks and an evaluation client. On the server side, an Apache HTTP server of version 2.2.22 is installed in Ubuntu 12.04 LTS (with default TCP CUBIC). For persistent connections, we set the server’s Timeout parameter to 1000s and

MaxRequest parameter to 0. That means, the client can request video segments unlimitedly. For video content, a video clip (i.e. “City” video in the perceptual evaluation of Section ??) is selected in our experiments. Because this clip is very short (about 10s), it is looped until the end of each session. The video content is subdivided into segments with the same duration of 2 seconds. Furthermore, the representation set has 15 available bitrate versions from 200 to 3000kbps with a step of 200kbps.

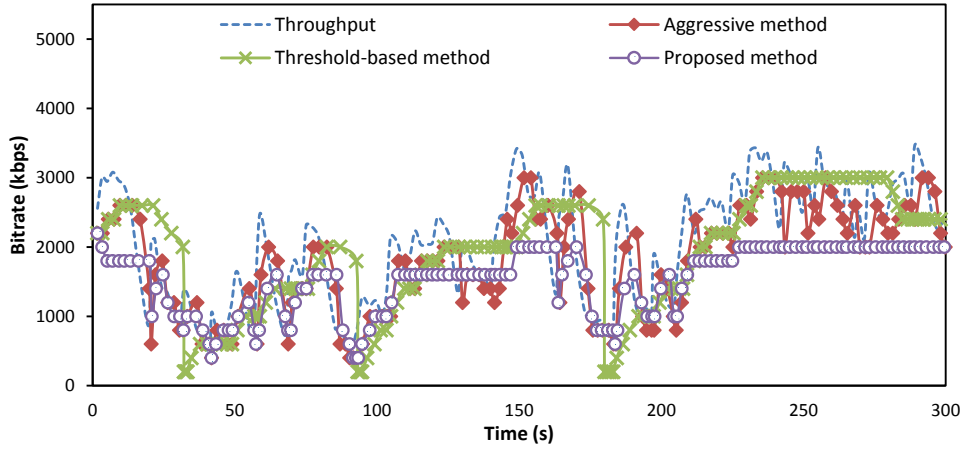
The client is implemented in Java language. The initial buffering delay (also the target buffer level) is set to 5 segment durations (i.e. 10s). In all experiments, we implement Threshold-based method with buffer thresholds $(B_1, B_2, B_3) = (4s, 7s, 10s)$. The safety margin μ is set to 0.1 for both reference methods. Meanwhile, our method is deployed with buffer thresholds $(\beta_{min}, \beta_{max}) = (4s, 10s)$, safety margin $\mu_{jnd} = 0.5$ JND units and $\alpha = 0.8$. For throughput estimation, we use the method of [10].

For network emulation, DummyNet tool [138] is installed on the client side to simulate network characteristics. RTT is set to 40ms. The packet-loss rate is set to 0%, assuming that it is already included into the fluctuations of the bandwidth trace.

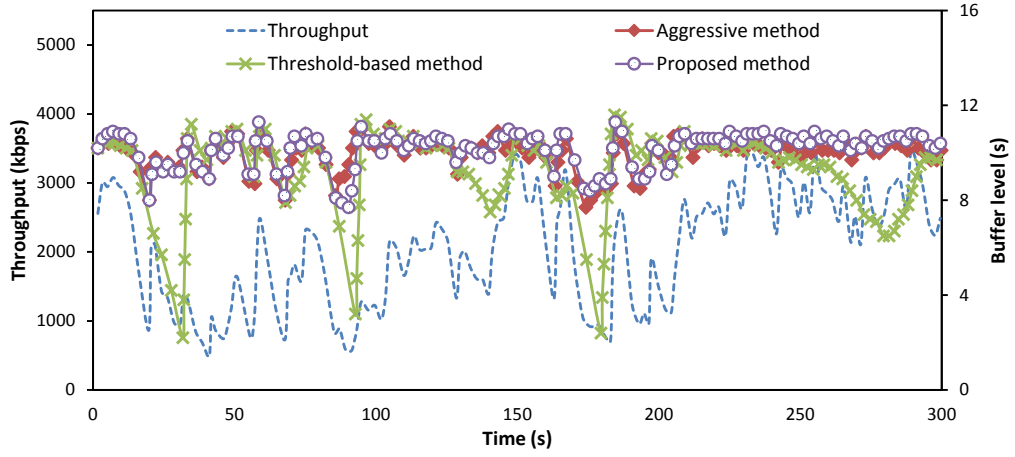
5.3.2 Obtained results

Fig. 5.3 shows the adapted bitrates and corresponding buffer levels of the three methods. Meanwhile, Fig. 5.4 shows the results of perceptual distortion values and relative distortion changes provided by evaluated methods. It should be noted that each distortion change is computed by the perceptual distortion difference between the two consecutive segments.

In general, the video bitrates requested by our method are less than that of other methods. It is because, our proposed method uses the safety margin in JND units. Although the difference is sometimes high (e.g. up to 800kbps at 270s), the perceptual distortion increase is very small (i.e. only 0.5 JND units as seen in Fig. 5.4a). Furthermore, the use of safety margin μ_{jnd} provides some advantages. Firstly, when the throughput fluctuates in high value ranges, the requested bitrates and resulting buffer levels are stable. For instance, during 250s ~ 300s, instead of following throughput variations as other methods, our method maintains a low bitrate



(a) Adapted bitrate

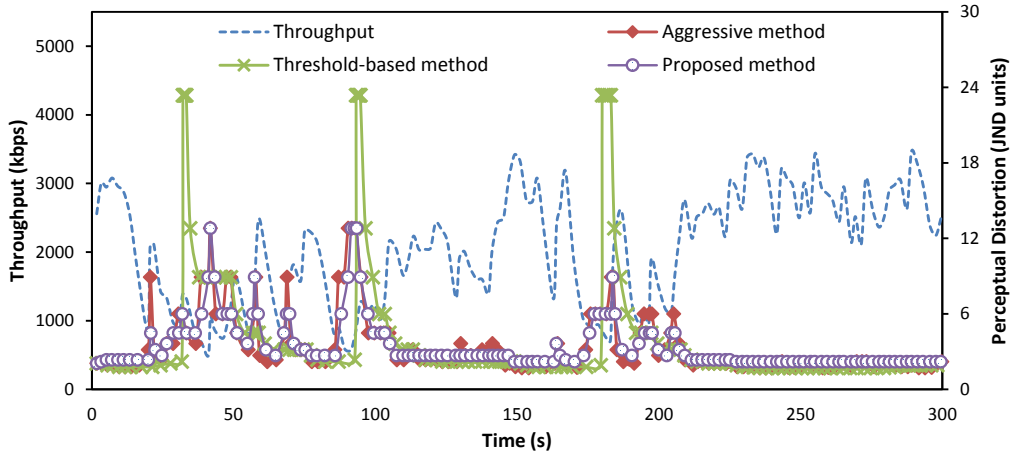


(b) Resulting buffer level

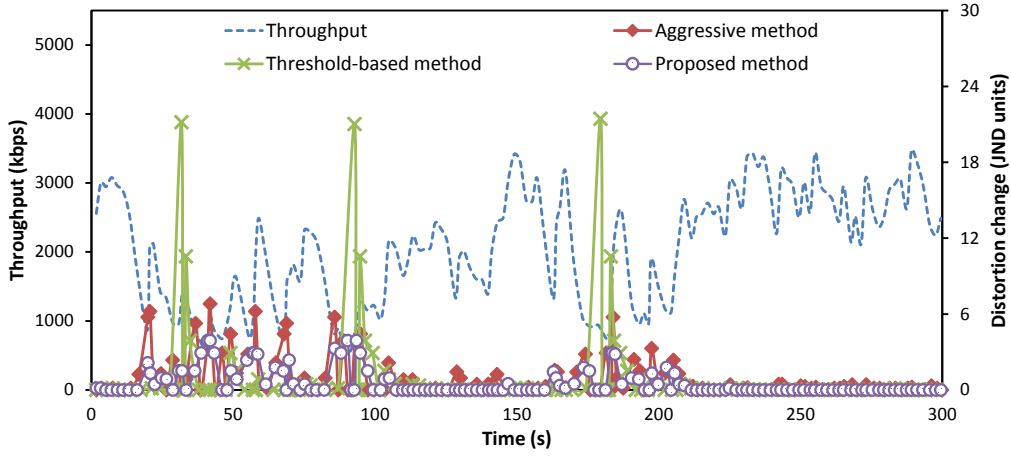
Figure 5.3: Adaptation results of three adaptation methods in the complex scenario.

(2200kbps), leading to a more stable buffer level and a smaller number of bitrate switches. Secondly, when the throughput suddenly drops, requesting the lower bitrate helps reducing the values of bitrate changes (or quality changes). For example, during $170s \sim 180s$, the bitrate change of our method are at most 600kbps (from 1600kbps to 1000kbps at $t = 173s$) while that of other methods are 1600kbps or more.

Thanks to buffer estimation, our method can provide video bitrate with gradual quality changes while still considering buffer situation. The distortion changes of our method are at most 3 JND units as seen in Fig. 5.4b. Meanwhile, the distortion changes of Aggressive method are sometimes higher than 5 JND units (especially in the first 100s) and those of Threshold-based method are even higher (e.g. up to more than 14 JND units at $t = 30s, 90s$, or $180s$).



(a) Resulting Perceptual Distortion



(b) Resulting Distortion change

Figure 5.4: Results of perceptual distortion and distortion change of three adaptation methods in the complex scenario.

Note that a perceptual distortion change of 5 JND units means, the difference is “readily apparent” [154]. Moreover, the switching amount of the proposed method is adjusted adaptively based on the buffer level at the given time to avoid buffer underflows. The smaller buffer level is, the higher the bitrate change (or distortion change) is. As seen in Fig. 5.3b, the resulting buffer level of our method is rarely under 7s and is comparable to that of Aggressive method.

Some statistics of the three methods’ behaviors are provided in Table 5.2. The first two data rows show the statistics of the buffer level, while the next two rows show the statistics of the perceptual distortion. The last three rows describe those of perceptual distortion changes.

We observe that the statistics reflect well the behaviors of the methods. The first row of Ta-

Table 5.2: Statistics of different adaptation methods. Except the number of switches and statistics of buffer level, the unit of other parameters is JND unit.

Statistics	Aggressive Method	Threshold-based Method	Proposed Method
The lowest buffer level (s)	7.7	2.2	7.7
STD of buffer level (s)	0.68	1.85	0.75
Average of distortion	3.57	5.29	3.62
STD of distortion	2.56	6.64	2.24
Maximum distortion change	6.81	21.44	3.89
Average of distortion changes	0.92	0.87	0.59
Number of non-zero distortion changes	97	48	55

Table 5.2 implies that our method and Aggressive method can provide high resulting buffer levels. In particular, the lowest buffer levels of these methods are equal to 7.7s. Note that the buffer size is only 10s. Besides, compared to Threshold-based method, our proposed method has the smaller standard deviation (STD) of buffer level (0.75). This value is comparable to that of Aggressive method (0.68). In terms of perceptual distortion, average distortion of our method (3.62) is similar to that of Aggressive method (3.57) and much higher than Threshold-based method (6.64). Moreover, our method has the smallest STD of distortion (2.24). This implies that our method provides the most stable perceptual distortion. In terms of distortion change, our proposed method always tries to request video segments with smooth distortion transitions. As shown in Table 5.2, our method results in the lowest maximum distortion change (3.89) and the lowest average of distortion changes (i.e. 0.59; reductions are at less 30% in comparison with others). Besides, its number of non-zero distortion changes is slightly higher than the lowest value (48). So, it is seen that the video provided by our method is much smoother than those of two reference methods.

5.4 Summary

In this chapter, we have presented a novel quality-based method for the case of live streaming and evaluated the proposed method with a buffer size of 10 seconds. Our detailed contri-

butions of this live-streaming-related work are as follows:

1. We for the first time used JND metric in bitrate adaptation over HTTP. In context of live streaming, a client cannot change the video quality step-by-step as in on-demand streaming. So, JND metric, which is used to indicate the perceptual quality change, can efficiently help the client maintain smooth quality transitions.
2. Based on JND metric and the future buffer based approach, we proposed an adaptation method for live streaming that can improve video quality by enabling smooth viewing transactions but also provides buffer stability.
3. The experimental results with a small buffer size of 10 seconds show that our proposed method can effectively support live streaming services.

6

Low-delay Live Streaming over HTTP



THE main challenge of HTTP low-delay live streaming is that the buffer size is limited to a few seconds or less. For example, given a small segment duration of 2 seconds, to maintain a buffer size of 6 seconds in low-delay streaming, a client only buffers at most 3 segments, leading to a high probability of buffer underflow, especially in mobile networks. Although deciding the bitrate in context of low-delay live streaming is not easy, the buffer size should be as small as possible, since low delay is a crucial requirement for live streaming services [47]. As analysed in [98], the smallest buffer size for HTTP live streaming is expected to be 2 segments.

In the literature, there are few studies focusing on adaptation problem in low-delay live streaming. In [132], the buffer size for live streaming scenario is set to 8s (or 4 segments), which

is much lower than the typical buffer size of 30s (or 15 segments) for on-demand scenario. In [135], a heuristic adaptation method for live streaming is proposed, but this method is only designed for buffers of three or more segments. Moreover, the method has some thresholds which are not easy to set. A learning-based adaptation method for low-delay streaming is proposed in [131]; however, the lowest supported buffer size is still 3 segments. In [67], through an extensive evaluation, it is found that throughput-based methods are more effective to support small buffer sizes than buffer-based methods. However, when the buffer size is reduced to 2 segments, no adaptation methods evaluated in [67] could guarantee a continuous session under strong variations of a mobile connection. Note that, besides the average video bitrate, interruptions strongly affect the quality of experience [22].

In this study, we propose a probabilistic adaptation method for HTTP low-delay live streaming, where the buffer size could be just two segments. We first formulate the adaptation problem for HTTP live streaming, taking into account the instant buffer level. Then, a solution to that problem formulation is presented that decides the bitrate for each segment. Through experiments, we show that the proposed method can effectively cope with throughput fluctuations and significantly reduce the chance of interruptions.

This chapter is organized as follows. Section 6.1 presents adaptation problem formulation for low-delay live streaming. We describe our adaptation method in Section 6.2. Section 6.3 presents experiment results and discussions. Finally, we summarize this chapter in Section 6.4.

6.1 Problem description

Suppose that, at the server, a video is provided at a number of bitrate versions, each is chopped into segments of τ seconds. For each segment n ($n = 1, 2, \dots$) during a streaming session, the client selects bitrate B_n from the available bitrate options.

In the initial buffering stage, the client requests the lowest bitrate for L ($L \geq 2$) segments before playing out. Because the client can download only the segments that have already been generated in real time, the client has to spend L segment durations for initial buffering. Note that the total duration of segments downloaded in initial buffering is equal to the buffer size

[67]. After initial buffering, the client switches to the steady stage, where it receives and plays video segments simultaneously. During this period, if the download rate and the playout rate are equal, the buffer level will be stable at the value of buffer size (i.e., $L \times \tau$ seconds); this value is also referred to as the target buffer level β^{tar} in this chapter.

Denote T_{n-1} the measured throughput of the last segment $n-1$ ($n > L$), which is used as the throughput estimate for the next segment n . In this study, the client uses a safety margin $\gamma_n \in [0, 1)$ to compute bitrate B_n :

$$B_n = T_{n-1} \times (1 - \gamma_n). \quad (6.1)$$

We denote β_n^r the buffer level measured at t_n^r , which is right after segment n is fully received. Because the buffer size in live streaming is small, the client should select the highest possible bitrate while the buffer is still not affected. For that purpose, we need to find the minimum margin γ_n so that the probability that the resulting buffer level β_n^r is lower than the target buffer level β^{tar} is smaller than a desired constraint.

Denote $\Pr(\beta_n^r < \beta^{tar} | \gamma_n)$ the probability that β_n^r is smaller than β^{tar} given γ_n . The adaptation problem is defined as:

$$\text{minimize} \quad \gamma_n \quad (6.2)$$

$$\text{subject to} \quad \Pr(\beta_n^r < \beta^{tar} | \gamma_n) < \varepsilon, \quad (6.3)$$

where ε is a desired probability constraint. Intuitively, the smaller ε is, the lower the risk of buffer instability will become, and also the lower the bitrate will be.

6.2 Adaptation solution

In this section, we present our proposed method in detail. Fig. 6.1 shows an illustration of the request times and buffer behavior of the client at segments $n-1$, n , and $n+1$. Let t_n^s and β_n^s be the time and the buffer level when the client sends the request of segment n , respectively.

In live streaming, the client should periodically send requests with a distance of τ seconds. So, in an ideal condition, segment n is requested at time t'_n (i.e., $t_n^s = t'_n$), which is computed by

$$t'_n = \vartheta + (n - 1) \times \tau, \quad (6.4)$$

where ϑ is the request time of the first segment. The time t'_n is considered as the earliest time, at which the client can send the request of segment n .

At time t'_{L+1} (i.e. when the play-out is started), β_{L+1}^s is β^{tar} . Therefore, in the steady stage, if the last segment $n - 1$ has been fully received before t'_n , the buffer level β_n^s at time $t_n^s = t'_n$ is also β^{tar} (Fig. 6.1).

Denote t_{n-1}^r the time right after the client has fully received the last segment $n - 1$. If $t_{n-1}^r < t'_n$, the client has to wait for an interval, which is $\Delta t_{n-1}^r = t'_n - t_{n-1}^r$, before sending the next request at t'_n (as illustrated in Fig. 6.1). However, because of throughput variations, the client may complete receiving the last segment $n - 1$ after time t'_n (i.e., $t_{n-1}^r \geq t'_n$). In the case where $t'_n \leq t_{n-1}^r \leq t'_n + \beta^{tar}$, the buffer level is reduced below the target level β^{tar} and so, the client should send the request immediately at time t_{n-1}^r to quickly increase the buffer level. Finally, if $t_{n-1}^r > t'_n + \beta^{tar}$ (i.e., the buffer has been depleted and the recently received segment $n - 1$ has missed its deadline), the client ignores segment $n - 1$ and switches back to the initial buffering stage.

So, if no buffer underflow is observed, the actual request time for segment n is

$$t_n^s = \max\{t'_n, t_{n-1}^r\}, \quad (6.5)$$

and the buffer level at that time is

$$\beta_n^s = \begin{cases} \beta^{tar} & \text{if } t_{n-1}^r < t'_n, \\ \beta_{n-1}^r & \text{if } t'_n \leq t_{n-1}^r \leq t'_n + \beta^{tar}. \end{cases} \quad (6.6)$$

The next segment n , which contains τ seconds of media, will be requested with video bitrate

where x_n , called the *throughput ratio constraint* for segment n , is calculated by

$$x_n = \frac{\beta_n^s + \tau - \beta^{tar}}{\tau \times (1 - \gamma_n)}. \quad (6.11)$$

Let X be a random variable that represents the ratio $\frac{T_{n-1}}{T_n}$, and $F_X(\cdot)$ be the cumulative distribution function (CDF) of X . So, the condition (6.10) is converted into

$$F_X(x_n) > 1 - \varepsilon. \quad (6.12)$$

In our method, the CDF of random variable X is obtained by using the throughput history of the client. Specifically, we use an independent process, called *observation process*, in which the CDF of random variable X is updated every 2s. Fig. 6.2b provides an illustration of the CDF of random variable X corresponding to a bandwidth trace (Fig. 6.2a).

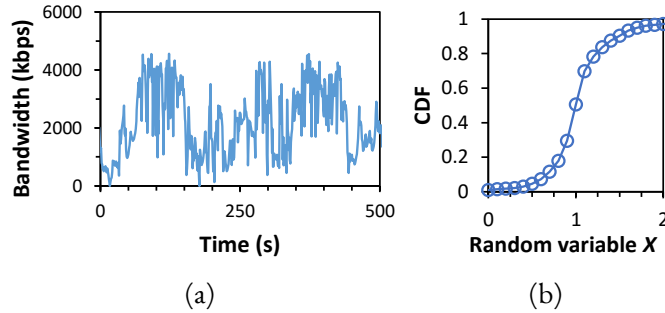


Figure 6.2: Illustration of (a) the bandwidth trace and (b) the CDF of variable X , obtained by the observation process.

Based on the obtained CDF, the client selects the minimum ratio x_n^* that meets condition (6.12), and then decides margin γ_n based on (6.11) as follows:

$$\gamma_n = 1 - \frac{\beta_n^s + \tau - \beta^{tar}}{\tau \times x_n^*}. \quad (6.13)$$

The general procedure to select the bitrate for segment n in the steady stage can be summarized as follows:

- 1) Measure the current buffer level. If the buffer is depleted, switch back to the initial buffer-

ing stage; otherwise, compute the throughput T_{n-1} of the last segment $n - 1$ as in [10].

- 2) Determine the time t'_n to send the next request using (6.5).
- 3) Given the CDF of random variable X , select the minimum x_n^* following condition (6.12); then compute the margin γ_n following (6.13).
- 4) Based on γ_n and T_{n-1} , decide the bitrate for the next segment following (6.1). If no bitrate is found, the minimum bitrate option is used.
- 5) Send the request to the server at time t'_n .
- 6) Repeat step 1 until the end of the session.

6.3 Experiments and Discussions

6.3.1 Experiment settings

Our test-bed organization used for the experiments is similar to that of Section 3.3, which consists of an HTTP Web server, a streaming client and an IP network as in Fig. 6.3. The IP network includes a router and wired connections connecting the server and the client. The server is an Apache HTTP server of version 2.2.22 running on Ubuntu 14.04LTS. Our test-bed uses DummyNet tool [138] installed at the client side to emulate network characteristics. The packet loss rate is set to 0%, assuming that the bandwidth trace used in the experiments already takes into account the fluctuations caused by packet loss. RTT value of DummyNet is set to 40ms. The client is implemented in Java and runs on a Windows 7 notebook with Core i5 2.6GHz CPU and 4GB RAM.

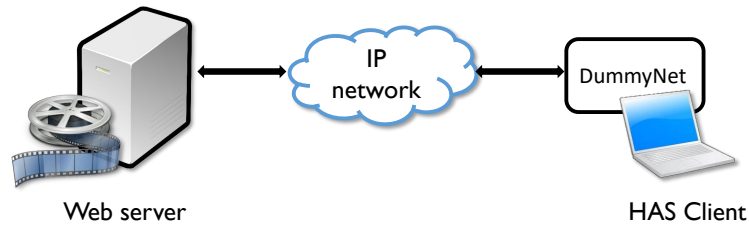


Figure 6.3: Test-bed organization for experiments.

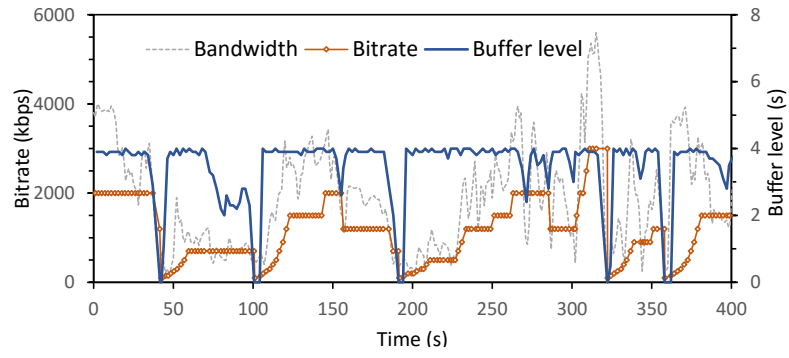
At the server, we employ a popular DASH dataset, where the video content is provided in constant bitrate (CBR) mode at 17 bitrate versions, from 100kbps to 6000kbps [69]. All segments have the same duration of $\tau = 2s$. At the client side, we implement our proposed method (called probability based (PB) method) with $\epsilon = 0.25$. The target buffer level is set to $4s$ (i.e., $L = 2$). For comparison, two reference methods, the instant throughput based (ITB) method [67] and the conservative throughput based (CTB) method [116], are employed. The former method uses a fixed margin of 0.2 while the latter method uses a mechanism like TCP congestion control, rather than a fixed margin.

6.3.2 Obtained results

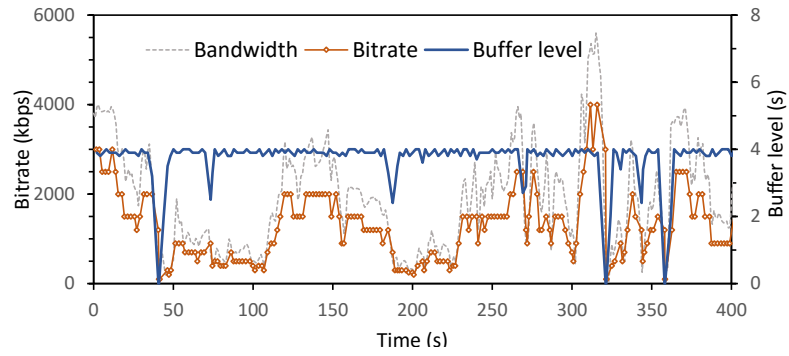
In our first experiment, we use two bandwidth traces. The first trace (Fig. 6.2a) is used as the bandwidth history data of a past session to compute the initial CDF of random variable X . The second trace (Fig. 6.4a) that is obtained from the same mobile network as the first one, is used to evaluate the proposed method. It should be noted that, at time 325s, the bandwidth drops to nearly zero, so any experiment run will mostly have one buffer underflow at this point. Fig. 6.4 shows the bitrate curves and buffer level curves of the three methods. We observe that the CTB method is conservative (aggressive) in increasing (decreasing) the bitrate. Despite this behavior, the CTB method results in as many as five buffer underflows (or interruptions). The bitrate curves of the ITB and PB methods are quite similar. However, the ITB method has three interruptions, whereas the PB method has only one, which is unavoidable at time 325s.

For adaptation statistics, the three adaptation methods are tested with multiple experimental runs, using a full bandwidth trace in Fig. 6.5 (obtained in [133]). For each adaptation method, 15 experimental runs are recorded, each of which is 400s long and has a random starting point. In this experiment, the PB method is employed with ϵ being 0.35, 0.25, and 0.15; these three options are respectively referred to as PB-35, PB-25 and PB-15. All other settings are the same as before.

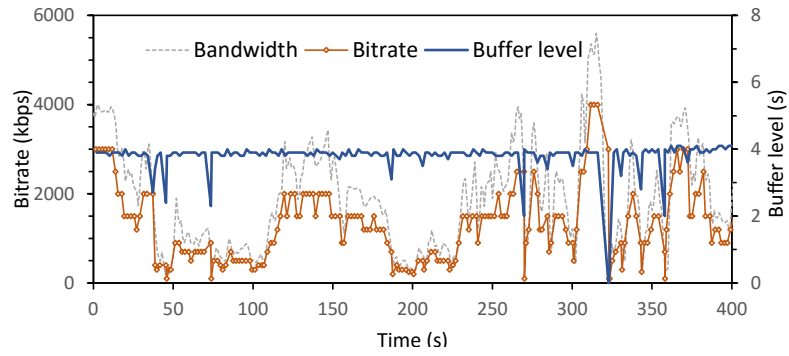
Table 6.1 provides statistics of adaptation results, where each item is the average value from the 15 experimental runs. The statistics show that the CTB method has a low average bitrate



(a)



(b)



(c)

Figure 6.4: The bandwidth trace and adaptation results of (a) the CTB method, (b) the ITB method, and (c) the PB method.

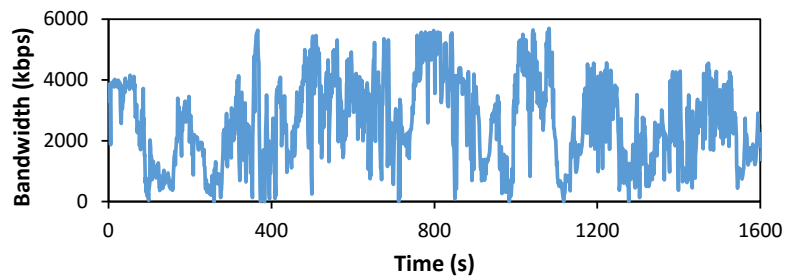


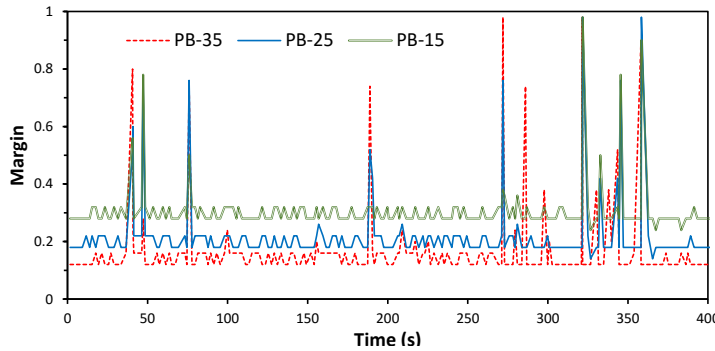
Figure 6.5: The bandwidth trace of a mobile network.

Table 6.1: Average statistics of the adaptation methods.

Statistics	CBT	ITB	PB-35	PB-25	PB-15
Average bitrate (kbps)	1678	1865	2043	1895	1661
Number of interruptions	3.73	1.80	1.47	1.20	0.87
Total duration of interruptions (s)	15.1	7.6	6.3	5.1	3.7

and the highest number of interruptions. Compared to the ITB method, the PB-35 and PB-25 options have fewer interruptions and better average bitrates. With the PB-15 option, although providing a somewhat lower average bitrate than that of the ITB method, it significantly reduces the number of interruptions (to 0.87) and the total duration of interruptions (to 3.7s).

In the proposed method, parameter ε , together with the current buffer status, actually affects the margin value. Fig. 6.6 shows the instant value of the margin when the bandwidth trace in Fig. 6.4a is used. It is seen that the margin value is initially decided to be 0.18 when $\varepsilon = 0.25$. During the session, the margin value varies and so the bitrate is adapted accordingly. For example, at time 40s, when the buffer level is drastically reduced due to a throughput drop, the margin is increased to 0.56. When the requirement for buffer stability is increased (i.e., ε is reduced), the margin is accordingly increased in an automatic manner. For example, when $\varepsilon = 0.15$, the starting value of the margin is decided to be 0.28. This means, parameter ε can be used to control the tradeoff between video bitrate and buffer stability in a live streaming session.

**Figure 6.6:** Instant value of the margin for $\varepsilon = 0.35, 0.25, 0.15$.

It can be seen that the effectiveness of the proposed method is provided by its two aspects. First, based on the recent throughput history and the probabilistic buffer constraint (ε), we can

decide the starting value of the margin and then adjust it during a session. In the ITB method, there is no way to decide the (fixed) margin value in advance. That means we cannot know, given some bandwidth trace, which margin will be a good one. Second, our method also considers the current buffer level. When the buffer level is dropped, the margin will be increased to avoid buffer underflows. As we try to keep the buffer level as stable as possible, there are not many times the buffer level is dropped (by unexpected throughput fluctuations). This is the reason why the margin value is mostly stable and is increased only at times of buffer level drop.

6.4 Summary

In this chapter, we have considered the adaptation problem of HTTP low-delay live streaming over mobile networks. Our detailed contributions of this work are as follows:

1. Since existing deterministic adaptation methods difficultly decide the bitrate to maintain a small buffer size in low-delay live streaming, we proposed a probability based method that helps reduce the chance of buffer underflow.
2. The experimental results in context of mobile streaming show that the proposed method significantly reduces the chance of buffer underflow by at least 30% while providing similar average video bitrate, compared to reference methods.
3. To the best of our method, our method is the first one that effectively supports a small buffer size of 2 segment durations.

This page intentionally left blank.

7

Conclusions and Future Work



IN this thesis, we have approached the problem of maximizing the video quality of HTTP adaptive streaming by smartly adapting the video quality to the available network conditions. We have developed adaptation methods for three important and challenging streaming scenarios: HTTP adaptive streaming of VBR videos, Adaptive streaming over HTTP/2, and HTTP live streaming. Although the problems are at the server, the client, or the transport protocol, we restrict to the client-based approach to maintain the advantages of HAS. Besides, all the proposed adaptation methods have been implemented and evaluated using streaming testbeds in the laboratory environment. Some insight gained while performing the various implementations could be used to help streaming developers design a flexible and efficient streaming client architecture. We summary our contributions and

our findings as follows.

Contributions

Chapter 3 presented our adaptation solution to adaptive streaming of VBR videos. To cope with bitrate variations, we proposed to use a local average bitrate as the representative bitrate of a version. A buffer-based algorithm was presented that uses the representative bitrate to make decisions. The experimental results show that with our approach, a normal buffer size (i.e., similar to that in CBR video streaming) is already enough to enable high video quality.

Chapter 4 focused on the adaptive streaming over HTTP/2. We developed an adaptation method that not only takes advantage of the HTTP/2 Server Push feature but also provides users with high video quality. Specifically, the proposed method considers some future segments rather than just one next segment. The decisions on bitrate selections for the future segments are based on request-related overhead, buffer stability, and gradual quality transitions. Through experiments in mobile network conditions, we showed that our method particularly outperforms reference methods in terms of video quality.

In Chapter 5, we proposed an adaptation method for HTTP live streaming, where the buffer size is about 10 seconds or less. The development of the adaptation method was motivated by the fact that video bitrate may be reduced even up to 40% without significantly degrading the viewing experience. To maintain seamless streaming with high video quality, the client estimates future buffer levels and selects the bitrate to meet the tradeoff between smooth quality transition and buffer stability. The experimental results show that our proposed method can change the video quality gracefully without buffer underflows.

Finally, we presented a probability adaptation method for HTTP low-delay live streaming in the Chapter 6. In context of low-delay live streaming, a critically small buffer is easily depleted due to strong throughput variations. Our approach takes advantage of the client buffer and considers the throughput history to select the video quality for each segment. The experimental results with a buffer size of only two segment durations showed that the proposed method can significantly reduce the chance of buffer underflows while providing high video bitrate.

Future Work

It is seen that we considered the three important contexts in HTTP adaptive streaming. Other combinations of these above contexts, e.g. HTTP/2-based adaptive streaming of VBR videos, HTTP live streaming of VBR videos, etc., will be reserved for our future work.

Besides, we envision further potential in using HTTP/2 to support adaptive streaming. The new version of HTTP offers many other prominent features such as stream termination, stream prioritization, etc., which are very promising to improve video quality.

Another hot topic of research covers the extension of adaptive streaming to multi-view videos (i.e., virtual reality, 360-degree videos). The optimal method that allows to effectively transfer such multi-view videos over the Internet is very challenging since these videos typically require huge bandwidth.

Last but not least, quality model for video streaming has not been addressed yet. A fast, low-complex, and effective quality model definitely helps the adaptation method improve user experience.

This page intentionally left blank.

Acknowledgment

This thesis would not have been possible without the help of many people. First of all, I am greatly thankful to Prof. Truong for his support, encouragement, and guidance to me to achieve this project. He also has set an excellent example as a researcher as well as an instructor. Being a passionate researcher, he loves to get to the bottom of things and masters the art of asking the right questions.

My sincere gratitude to Prof. Anh Pham, the head of Computer Communications Laboratory, for his help and support over the past five years. I would like to thank Prof. Paik and Prof. Li for spending time to work as review committee members for this thesis. I am deeply indebted to Prof. Nam Pham, my former professor at Hanoi University of Science and Technology for sharing his magnificent experience in life as well as in work.

I am enormously grateful to my friends at University of Aizu for their remarkable supports. Special thanks goes to Dr. Vuong Viet Mai of Korea Advanced Institute of Science and Technology for sharing his knowledge and his understanding during my studies. We had a great time working together.

This thesis document is partly belong to an Adaptive Streaming project, in Computer Communications Laboratory, University of Aizu. I inherited the basic client program and the Java tutorials from the senior students' works. I would like to thank all of them for the solid platform that I can work with.

I would like to express my infinite gratitude to my family in Vietnam for their love and their

encouragement. They have been inspiring and pushing me to obtain my goal and I am happy to have them in my life.

Last but not least, I would like to acknowledge financial support from Japanese government for my 3-year graduate study. Very special thanks to Aizu Area Foundation for the Promotion of Education and Science, Telecommunication Advancement Foundation, and Computer Communications Laboratory for awarding me travel grants to attend international conferences.

References

- [1] “Advanced Research Projects Agency Network,” [Online] Available: <https://en.wikipedia.org/wiki/ARPANET>. Accessed: Apr. 16, 2017.
- [2] “QuickTime,” [Online] Available: <https://en.wikipedia.org/wiki/QuickTime>. Accessed: Apr. 16, 2017.
- [3] “ActiveMovie,” [Online] Available: <https://en.wikipedia.org/wiki/ActiveMovie>. Accessed: Apr. 16, 2017.
- [4] “RealPlayer,” [Online] Available: <http://en.wikipedia.org/wiki/RealPlayer>. Accessed: Apr. 16, 2017.
- [5] Sandvine, “2016 Global Internet Phenomena,” [Online] Available: <https://www.sandvine.com/trends/global-internet-phenomena>. Accessed: Apr. 16, 2017.
- [6] “YouTube statistics,” [Online] Available: <http://www.youtube.com/yt/press/statistics.html>. Accessed: Apr. 6, 2017.
- [7] T. Verge, “League-of-Legends eSports finals watched by 32 million people,” [Online] Available: <http://www.theverge.com/2013/11/19/5123724/league-of-legends-world-championship-32-million-viewers>. Accessed: Apr. 16, 2017.
- [8] H. Riiser, *Adaptive Bitrate Video Streaming over HTTP in Mobile Wireless Networks*. PhD thesis, Faculty of Mathematics and Natural Sciences, University of Oslo, Jun. 2013.
- [9] A. C. Begen, T. Akgul, and M. Baugher, “Watching Video over the Web: Part I: Streaming Protocols,” *IEEE Internet Computing*, vol. 15, pp. 54–63, Mar. 2011.
- [10] T. C. Thang, Q.-D. Ho, J. W. Kang, and A. T. Pham, “Adaptive streaming of audiovisual content using MPEG DASH,” *IEEE Transactions on Consumer Electronics*, vol. 58, pp. 78–85, Feb. 2012.
- [11] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson, “RTP: A Transport Protocol for Real-Time Applications,” [Online] Available: <https://tools.ietf.org/html/rfc3550>, Jul. 2003.
- [12] H. Schulzrinne, A. Rao, and R. Lanphier, “Real Time Streaming Protocol (RTSP),” [Online] Available: <https://tools.ietf.org/html/rfc2326>, Apr. 1998.

- [13] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, and T. Berner-Lee, "Hypertext Transfer Protocol (RFC 2068)," [Online] Available: <https://www.ietf.org/rfc/rfc2068.txt>, Jan. 1997.
- [14] M. Belshe, R. Peon, and M. Thomson, "Hypertext Transfer Protocol Version 2 (RFC 7540)," [Online] Available: <http://datatracker.ietf.org/doc/rfc7540/>, May 2015.
- [15] T. Stockhammer, "Dynamic Adaptive Streaming over HTTP: Standards and Design Principles," in *Proc. Second Annual ACM Conference on Multimedia Systems (MMSys '11)*, pp. 133–144, 2011.
- [16] ISO/IEC 23009-1:2014, "Information technology - Dynamic adaptive streaming over HTTP (DASH) – Part 1: Media presentation description and segment formats," 2014.
- [17] A. C. Begen, T. Akgul, and M. Baugher, "Watching Video over the Web: Part II: Applications, Standardization, and Open Issues," *IEEE Internet Computing*, vol. 15, pp. 59–63, May 2011.
- [18] C. Systems, "Cisco Visual Network Index: Forecast and Methodology, 2015 - 2020 While Paper," Jun. 2016.
- [19] T. Lakshman, A. Ortega, and A. Reibman, "VBR video: tradeoffs and potentials," *Proc. of the IEEE*, vol. 86, pp. 952–973, May 1998.
- [20] R. Huyssegems, J. van der Hooft, T. Bostoen, P. Rondao Alface, S. Petrangeli, T. Wauters, and F. De Turck, "HTTP/2-Based Methods to Improve the Live Experience of Adaptive Streaming," in *Proc. 23rd ACM International Conference on Multimedia (MM '15)*, pp. 541–550, 2015.
- [21] H. T. Le, "Buffer-based Bitrate Adaptation for Adaptive HTTP Streaming," Mar. 2014.
- [22] M. Seufert, S. Egger, M. Slanina, T. Zinner, T. Hoßfeld, and P. Tran-Gia, "A Survey on Quality of Experience of HTTP Adaptive Streaming," *IEEE Communications Surveys Tutorials*, vol. 17, no. 1, pp. 469–492, 2015.
- [23] T. Hoßfeld, S. Egger, R. Schatz, M. Fiedler, K. Masuch, and C. Lorentzen, "Initial delay vs. interruptions: Between the devil and the deep blue sea," in *Proc. 4th International Workshop on Quality of Multimedia Experience (QoMEX 2012)*, pp. 1–6, Jul. 2012.
- [24] T. C. Thang, H. X. Nguyen, A. T. Pham, and N. P. Ngoc, "Perceptual difference evaluation of video alternatives in adaptive streaming," in *2012 Fourth International Conference on Communications and Electronics (ICCE)*, pp. 322–326, Aug 2012.
- [25] H. T. Le, H. N. Nguyen, N. P. Ngoc, A. T. Pham, and T. C. Thang, "A Novel Adaptation Method for HTTP Streaming of VBR Videos over Mobile Networks," *Mobile Information Systems*, vol. 2016, Article ID 2920850, 2016.
- [26] H. N. Nguyen, T. Vu, H. T. Le, N. P. Ngoc, and T. C. Thang, "Smooth Quality Adaptation Method for VBR Video Streaming over HTTP," in *International Conference on Computing, Management and Telecommunications (ComManTel 2015)*, Dec. 2015.

- [27] H. T. Le, T. Vu, N. P. Ngoc, A. T. Pham, and T. C. Thang, "Seamless mobile video streaming over HTTP/2 with gradual quality transitions," *IEICE Transactions on Communications*, vol. E100.B, pp. 901–909, May 2017.
- [28] T. Vu, H. T. Le, P. N. Nam, and T. C. Thang, "Adaptive Mobile Streaming Over HTTP/2 with Gradual Quality Transitions," in *Proc. 5th IEEE Global Conference on Consumer Electronics (GCCE 2016)*, Oct. 2016.
- [29] H. T. Le, H. N. Nguyen, N. P. Ngoc, A. T. Pham, H. L. Minh, and T. C. Thang, "Quality-driven bitrate adaptation method for HTTP live-streaming," in *Proc. IEEE International Conference on Communication Workshop (ICCW 2015)*, pp. 1771–1776, Jun. 2015.
- [30] H. T. Le, N. P. Ngoc, A. T. Pham, and T. C. Thang, "A probabilistic adaptation method for HTTP low-delay live streaming over mobile networks," *IEICE Transactions on Information & Systems*, vol. E100.D, pp. 379–383, Feb. 2017.
- [31] R. Braden, L. Zhang, S. Berson, A. Herzog, and S. Jamin, "Resource ReSerVation Protocol (RSVP)," [Online] Available: <https://tools.ietf.org/html/rfc2205>, Sep. 1997.
- [32] Y. Bernet, P. Ford, R. Yavatkar, F. Baker, L. Zhang, M. Speer, R. Braden, B. Davie, J. Wroclawski, and E. Felstaine, "A Framework for Integrated Services Operation over Diffserv Networks," [Online] Available: <https://tools.ietf.org/html/rfc2998>, Nov. 2000.
- [33] T. Friedman, R. Caceres, and A. Clark, "RTP Control Protocol Extended Reports (RTCP XR)," [Online] Available: <https://tools.ietf.org/html/rfc3611>, Nov. 2003.
- [34] C. Holmberg, S. Hakansson, and G. Eriksson, "Web Real-Time Communication Use Cases and Requirements," [Online] Available: <https://tools.ietf.org/html/rfc7478>, Mar. 2015.
- [35] C. Perkins and V. Singh, "Multimedia Congestion Control: Circuit Breakers for Unicast RTP Sessions," [Online] Available: <https://tools.ietf.org/html/rfc8083>, Mar. 2017.
- [36] V. Singh, S. Ahsan, and J. Ott, "MPRTP: Multipath Considerations for Real-time Media," in *Proc. 4th ACM Multimedia Systems Conference (MMSys '13)*, pp. 190–201, 2013.
- [37] J. Postel, "User Datagram Protocol (RFC 768)," [Online] Available: <https://tools.ietf.org/html/rfc768>, Aug. 1980.
- [38] S. McCanne, V. Jacobson, and M. Vetterli, "Receiver-driven Layered Multicast," *SIGCOMM Computer Communication Review*, vol. 26, pp. 117–130, Aug. 1996.
- [39] K. A. Noghani and M. O. Sunay, "Streaming Multicast Video over Software-Defined Networks," in *Proc. 11th International Conference on Mobile Ad Hoc and Sensor Systems*, pp. 551–556, Oct 2014.
- [40] H. Eriksson, "MBONE: The Multicast Backbone," *Communications of the ACM*, vol. 37, pp. 54–60, Aug. 1994.

- [41] C. Diot, B. N. Levine, B. Lyles, H. Kassem, and D. Balensiefen, "Deployment Issues for the IP Multicast Service and Architecture," *Netw. Mag. of Global Internetworkg.*, vol. 14, pp. 78–88, Jan. 2000.
- [42] B. Li and J. Liu, "Multirate video multicast over the Internet: an overview," *IEEE Network*, vol. 17, pp. 24–29, Jan 2003.
- [43] Y. Cui, B. Li, and K. Nahrstedt, "oStream: asynchronous streaming multicast in application-layer overlay networks," *IEEE Journal on Selected Areas in Communications*, vol. 22, pp. 91–106, Jan 2004.
- [44] Y. Liu, Y. Guo, and C. Liang, "A survey on peer-to-peer video streaming systems," *Peer-to-Peer Networking and Applications*, vol. 1, no. 1, pp. 18–28, 2008.
- [45] A. Arefin, Z. Huang, K. Nahrstedt, and P. Agarwal, "4D TeleCast: Towards Large Scale Multi-site and Multi-view Dissemination of 3DTI Contents," in *Proc. 32nd International Conference on Distributed Computing Systems*, pp. 82–91, June 2012.
- [46] J. F. Kurose and K. W. Ross, *Computer Networking: A Top-Down Approach*. Pearson, 6th ed., 2012.
- [47] R. Kooji, K. Ahmed, and K. Brunnstrom, "Perceived Quality of Channel Zapping," in *Proc. 5th International Conference Communication Systems and Networks (CSN'06)*, pp. 155–158, Aug. 2006.
- [48] Youtube, "Live encoder settings, bitrates, and resolutions," [Online] Available: <https://support.google.com/youtube/answer/2853702?hl=en>. Accessed: Apr. 16, 2017.
- [49] Facebook, "Live Videos from Publishing Tools," [Online] Available: <https://www.facebook.com/facebookmedia/get-started/live>. Accessed: Apr. 16, 2017.
- [50] "Methods for subjective determination of transmission quality (P.800)," *International Telecommunication Union*, Aug. 1996.
- [51] K. Miller, *Adaptation Algorithms for HTTP-Based Video Streaming*. PhD thesis, Telecommunication Networks Group Department, Technische Universität Berlin, Nov. 2016.
- [52] E. Kohler, M. Handley, and S. Floyd, "Datagram Congestion Control Protocol (DCCP)," [Online] Available: <https://tools.ietf.org/html/rfc4340>, Mar. 2006.
- [53] C. Perkins, "RTP and the Datagram Congestion Control Protocol," [Online] Available: <http://tools.ietf.org/html/rfc5762>, Apr. 2010.
- [54] R. Stewart, "Stream Control Transmission Protocol," [Online] Available: <https://tools.ietf.org/html/rfc4960>, Sep. 2007.

- [55] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, and E. Schooler, "SIP: Session Initiation Protocol," [Online] Available: <https://www.ietf.org/rfc/rfc3261>, Jun. 2002.
- [56] K. J. Ma, R. Bartos, S. Bhatia, and R. Nair, "Mobile video delivery with HTTP," *IEEE Communications Magazine*, vol. 49, pp. 166–175, April 2011.
- [57] B. Wang, J. Kurose, P. Shenoy, and D. Towsley, "Multimedia Streaming via TCP: An Analytic Performance Study," in *Proc. 12th Annual ACM International Conference on Multimedia (MULTIMEDIA'04)*, pp. 908–915, 2004.
- [58] B. Wang, J. Kurose, P. Shenoy, and D. Towsley, "Multimedia Streaming via TCP: An Analytic Performance Study," *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)*, vol. 4, pp. 16:1–16:22, May 2008.
- [59] A. C. Auge and J. P. Aspas, "TCP/IP over wireless links: performance evaluation," in *Vehicular Technology Conference, 1998. VTC 98. 48th IEEE*, vol. 3, pp. 1755–1759 vol.3, May 1998.
- [60] J. Yang, N. Tin, and A. K. Khandani, "Adaptive modulation and coding in 3G wireless systems," in *Proceedings IEEE 56th Vehicular Technology Conference*, vol. 1, pp. 544–548 vol.1, 2002.
- [61] D. F. Brueck and M. B. Hurst, "Apparatus, system, method for multi-bitrate content streaming," U.S. Patent 7 818 444 B2, Oct. 2010.
- [62] Z. Li, X. Zhu, J. Gahm, R. Pan, H. Hu, A. C. Begen, and D. Oran, "Probe and Adapt: Rate Adaptation for HTTP Video Streaming At Scale," *IEEE Journal on Selected Areas in Communications*, vol. 32, pp. 719–733, April 2014.
- [63] Microsoft Corporation, "Microsoft Silverlight Smooth Streaming," [Online] Available: <https://www.microsoft.com/silverlight/smoothstreaming>. Accessed: Aug. 14, 2017.
- [64] Apple Inc., "Apple HTTP Live Streaming," [Online] Available: <https://developer.apple.com/streaming/>. Accessed: Aug. 14, 2017.
- [65] Adobe Inc., "Adobe HTTP Dynamic Streaming (HDS) Technology Center," [Online] Available: <https://www.adobe.com/devnet/hds.html>. Accessed: Aug. 14, 2017.
- [66] J. Ozer, "What's the right keyframe interval?," [Online] Available: <http://streaminglearningcenter.com/blogs/whats-the-right-keyframe-interval.html>. Accessed: Apr. 16, 2017.
- [67] T. C. Thang, H. T. Le, A. T. Pham, and Y. M. Ro, "An Evaluation of Bitrate Adaptation Methods for HTTP Live Streaming," *IEEE Journal on Selected Areas in Communications*, vol. 32, pp. 693–705, Apr. 2014.
- [68] A. B. Watson and L. Kreslake, "Measurement of visual impairment scales for digital video," in *Proc. SPIE–Human Vision, Visual Processing, and Digital Display*, pp. 79–89, 2001.

- [69] S. Lederer, C. Müller, and C. Timmerer, “Dynamic Adaptive Streaming over HTTP Dataset,” in *Proc. 3rd Multimedia Systems Conference (MMSys ’12)*, pp. 89–94, Feb. 2012.
- [70] C. Kreuzberger, B. Rainer, H. Hellwagner, L. Toni, and P. Frossard, “A Comparative Study of DASH Representation Sets Using Real User Characteristics,” in *Proc. 26th International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV ’16)*, pp. 4:1–4:6, 2016.
- [71] L. Toni, R. Aparicio-Pardo, K. Pires, G. Simon, A. Blanc, and P. Frossard, “Optimal Selection of Adaptive Streaming Representations,” *ACM Trans. Multimedia Comput. Commun. Appl.*, vol. 11, pp. 43:1–43:26, Feb. 2015.
- [72] J. Kua, G. Armitage, and P. Branch, “A Survey of Rate Adaptation Techniques for Dynamic Adaptive Streaming over HTTP,” *IEEE Communications Surveys Tutorials*, 2017.
- [73] Y. Sani, A. Mauthe, and C. Edwards, “Adaptive Bitrate Selection: A Survey,” *IEEE Communications Surveys Tutorials*, 2017.
- [74] ISO/IEC 13818-1:2000, “Information Technology—Generic Coding of Moving Pictures and Associated Audio Information: Systems,” 2000.
- [75] ISO/IEC 14496-12:2005, “Information Technology—Coding of Audio-visual Objects—Part 12: ISO Base Media File Format,” 2005.
- [76] C. Mueller, S. Lederer, C. Timmerer, and H. Hellwagner, “Dynamic Adaptive Streaming over HTTP/2.0,” in *Proc. IEEE International Conference on Multimedia and Expo (ICME ’13)*, pp. 1–6, Jul. 2013.
- [77] M. Jain and C. Dovrolis, “Ten Fallacies and Pitfalls on End-to-end Available Bandwidth Estimation,” in *Proc. 4th ACM SIGCOMM Conference on Internet Measurement (IMC ’04)*, pp. 272–277, 2004.
- [78] V. Ramamurthi and O. Oyman, “Link aware HTTP Adaptive Streaming for enhanced quality of experience,” in *Proc. IEEE Global Communications Conference (GLOBECOM’13)*, pp. 1675–1680, Dec. 2013.
- [79] V. Ramamurthi, O. Oyman, and J. Foerster, “Using link awareness for HTTP Adaptive Streaming over changing wireless conditions,” in *Proc. International Conference on Computing, Networking and Communications (ICNC 2015)*, pp. 727–731, Feb. 2015.
- [80] A. Balachandran, V. Sekar, A. Akella, S. Seshan, I. Stoica, and H. Zhang, “A Quest for an Internet Video Quality-of-experience Metric,” in *Proc. 11th ACM Workshop on Hot Topics in Networks*, pp. 97–102, 2012.
- [81] W. Song and D. W. Tjondronegoro, “Acceptability-Based QoE Models for Mobile Video,” *IEEE Transactions on Multimedia*, vol. 16, pp. 738–750, Apr. 2014.

- [82] U. Reiter, K. Brunnström, K. De Moor, M.-C. Larabi, M. Pereira, A. Pinheiro, J. You, and A. Zgank, *Factors Influencing Quality of Experience*, pp. 55–72. Springer International Publishing, 2014.
- [83] T. De Pessemier, K. De Moor, W. Joseph, L. De Marez, and L. Martens, “Quantifying the Influence of Rebuffering Interruptions on the User’s Quality of Experience During Mobile Video Watching,” *IEEE Transactions on Broadcasting*, vol. 59, pp. 47–61, Mar. 2013.
- [84] A. Raake and S. Egger, *Quality of Experience Advanced Concepts, Applications and Methods*. Springer, 2014.
- [85] Y. Qi and M. Dai, “The Effect of Frame Freezing and Frame Skipping on Video Quality,” in *2006 International Conference on Intelligent Information Hiding and Multimedia*, pp. 423–426, Dec 2006.
- [86] T. Hoßfeld, D. Strohmeier, A. Raake, and R. Schatz, “Pippi Longstocking Calculus for Temporal Stimuli Pattern on YouTube QoE: $1+1=3$ and $1\cdot4\neq4\cdot1$,” in *Proc. 5th Workshop on Mobile Video (MoVid ’13)*, pp. 37–42, Feb. 2013.
- [87] Q. Huynh-Thu and M. Ghanbari, “Temporal Aspect of Perceived Quality in Mobile Video Broadcasting,” *IEEE Transactions on Broadcasting*, vol. 54, pp. 641–651, Sept 2008.
- [88] T. Hoßfeld, M. Seufert, M. Hirth, T. Zinner, P. Tran-Gia, and R. Schatz, “Quantification of YouTube QoE via Crowdsourcing,” in *Proc. IEEE International Symposium on Multimedia (ISM ’11)*, pp. 494–499, Dec. 2011.
- [89] M. Zink, J. Schmitt, and R. Steinmetz, “Layer-encoded video in scalable adaptive streaming,” *IEEE Transactions on Multimedia*, vol. 7, pp. 75–84, Feb. 2005.
- [90] T. Hoßfeld, M. Seufert, C. Sieber, and T. Zinner, “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)*, pp. 111–116, Sep. 2014.
- [91] R. K. Mok, E. W. Chan, X. Luo, and R. K. Chang, “Inferring the QoE of HTTP Video Streaming from User-viewing Activities,” in *Proc. First ACM SIGCOMM Workshop on Measurements Up the Stack (W-MUST ’11)*, pp. 31–36, 2011.
- [92] O. Oyman and S. Singh, “Quality of experience for HTTP adaptive streaming services,” *IEEE Communications Magazine*, vol. 50, pp. 20–27, Apr. 2012.
- [93] C. Alberti, D. Renzi, C. Timmerer, C. Mueller, S. Lederer, S. Battista, and M. Matavelli, “Automated QoE evaluation of Dynamic Adaptive Streaming over HTTP,” in *2013 Fifth International Workshop on Quality of Multimedia Experience (QoMEX)*, pp. 58–63, Jul. 2013.
- [94] B. Lewcio, B. Belmudez, A. Mehmood, M. Wältermann, and S. Möller, “Video quality in next generation mobile networks – Perception of time-varying transmission,” in *Proc. IEEE International Workshop Technical Committee on Communications Quality and Reliability (CQR ’11)*, pp. 1–6, May 2011.

- [95] L. Yitong, S. Yun, M. Yinian, L. Jing, L. Qi, and Y. Dacheng, "A study on Quality of Experience for adaptive streaming service," in *Proc. IEEE International Conference on Communications Workshops (ICC '13)*, pp. 682–686, June 2013.
- [96] R. K. P. Mok, X. Luo, E. W. W. Chan, and R. K. C. Chang, "QDASH: A QoE-aware DASH System," in *Proc. Third Multimedia Systems Conference (MMSys '12)*, pp. 11–22, Feb. 2012.
- [97] M. Grafl and C. Timmerer, "Representation Switch Smoothing for Adaptive HTTP Streaming," in *Proc. 4th International Workshop on Perceptual Quality of Systems (PQS 2013)*, pp. 178–183, Sep. 2013.
- [98] T. Lohmar, T. Einarsson, P. Frojdh, F. Gabin, and M. Kampmann, "Dynamic adaptive HTTP streaming of live content," in *Proc. IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM)*, pp. 1–8, Jun. 2011.
- [99] J. van der Hooft, S. Petrangeli, T. Wauters, R. Huysegems, T. Bostoen, and F. De Turck, "An HTTP/2 Push-Based Approach for Low-Latency Live Streaming with Super-Short Segments," *Journal of Network and Systems Management*, Mar. 2017.
- [100] H. T. T. Tran, N. P. Ngoc, Y. J. Jung, A. T. Pham, and T. C. Thang, "A novel quality model for HTTP adaptive streaming," *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, vol. E100-A, no. 2, pp. 555–564, 2017.
- [101] Conviva, "Viewer Experience Report," tech. rep., 2014.
- [102] K. D. Singh, Y. Hadjadj-Aoul, and G. Rubino, "Quality of experience estimation for adaptive HTTP/TCP video streaming using H.264/AVC," in *Proc. IEEE Consumer Communications and Networking Conference (CCNC '12)*, pp. 127–131, Jan. 2012.
- [103] T. Hoßfeld, M. Seufert, C. Sieber, T. Zinner, and P. Tran-Gia, "Identifying QoE optimal adaptation of HTTP adaptive streaming based on subjective studies," *Computer Networks*, vol. 81, pp. 320 – 332, 2015.
- [104] J. D. Vriendt, D. D. Vleeschauwer, and D. Robinson, "Model for estimating QoE of video delivered using HTTP adaptive streaming," in *Proc. IFIP/IEEE International Symposium on Integrated Network Management (IM '13)*, pp. 1288–1293, May 2013.
- [105] Z. D. Rodriguez, Z. Wang, L. Rosa, and G. Bressan, "The impact of video-quality-level switching on user quality of experience in dynamic adaptive streaming over HTTP," *EURASIP Journal on Wireless Communications and Networking*, vol. 2014, pp. 1687–1499, Dec. 2014.
- [106] Z. Guo, Y. Wang, and X. Zhu, "Assessing the visual effect of non-periodic temporal variation of quantization stepsize in compressed video," in *Proc. IEEE International Conference on Image Processing (ICIP '15)*, pp. 3121–3125, Sept 2015.
- [107] Y. Shen, Y. Liu, Q. Liu, and D. Yang, "A method of QoE evaluation for adaptive streaming based on bitrate distribution," in *Proc. IEEE International Conference on Communications Workshops (ICC '14)*, pp. 551–556, June 2014.

- [108] T. C. Thang, H. T. Le, H. X. Nguyen, A. T. Pham, J. W. Kang, and Y. M. Ro, "Adaptive video streaming over HTTP with dynamic resource estimation," *Journal of Communications and Networks*, vol. 15, pp. 635–644, Dec. 2013.
- [109] D. V. Nguyen, D. M. Nguyen, H. T. Tran, N. P. Ngoc, A. T. Pham, and T. C. Thang, "Quality-delay tradeoff optimization in multi-bitrate adaptive streaming," in *Proc. IEEE International Conference on Consumer Electronics (ICCE)*, pp. 66–67, Jan. 2015.
- [110] Y. Zhou, Y. Duan, J. Sun, and Z. Guo, "Towards simple and smooth rate adaption for VBR video in DASH," in *Proc. IEEE Visual Communications and Image Processing Conference*, pp. 9–12, Dec. 2014.
- [111] H. T. Le, N. P. Ngoc, T. A. Vu, A. T. Pham, and T. C. Thang, "Smooth-bitrate adaptation method for HTTP streaming in vehicular environments," in *Proc. IEEE Vehicular Networking Conference (VNC 2014)*, pp. 187–188, Dec. 2014.
- [112] Y. Sun, X. Yin, J. Jiang, V. Sekar, F. Lin, N. Wang, T. Liu, and B. Sinopoli, "CS2P: Improving Video Bitrate Selection and Adaptation with Data-Driven Throughput Prediction," in *Proc. ACM SIGCOMM Conference (SIGCOMM '16)*, pp. 272–285, 2016.
- [113] K. Evensen, A. Petlund, H. Riiser, P. Vigmostad, D. Kaspar, C. Griwodz, and P. Halvorsen, "Mobile Video Streaming Using Location-based Network Prediction and Transparent Handover," in *Proc. 21st International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV '11)*, pp. 21–26, Jun. 2011.
- [114] J. Jiang, V. Sekar, and H. Zhang, "Improving Fairness, Efficiency, and Stability in HTTP-based Adaptive Video Streaming with FESTIVE," in *Proc. 8th International Conference on Emerging Networking Experiments and Technologies (CoNEXT '12)*, pp. 97–108, 2012.
- [115] S. Garcia, J. Cabrera, and N. Garcia, "Quality-Control Algorithm for Adaptive Streaming Services Over Wireless Channels," *IEEE Journal of Selected Topics in Signal Processing*, vol. 9, pp. 50–59, Feb. 2015.
- [116] C. Liu, I. Bouazizi, and M. Gabbouj, "Rate Adaptation for Adaptive HTTP Streaming," in *Proc. Second Annual ACM Conference on Multimedia Systems (MMSys '11)*, pp. 169–174, Feb. 2011.
- [117] H. T. Le, D. V. Nguyen, N. P. Ngoc, A. T. Pham, and T. C. Thang, "Buffer-based bitrate adaptation for adaptive HTTP streaming," in *International Conference on Advanced Technologies for Communications (ATC 2013)*, pp. 33–38, Oct. 2013.
- [118] Z. Li, A. C. Begen, J. Gahm, Y. Shan, B. Osler, and D. Oran, "Streaming Video over HTTP with Consistent Quality," in *Proc. 5th ACM Multimedia Systems Conference (MMSys '14)*, pp. 248–258, 2014.

- [119] T.-Y. Huang, R. Johari, N. McKeown, M. Trunnell, and M. Watson, "A Buffer-based Approach to Rate Adaptation: Evidence from a Large Video Streaming Service," in *Proc. ACM Conference on SIGCOMM (SIGCOMM '14)*, pp. 187–198, 2014.
- [120] A. Bokani, M. Hassan, and S. Kanhere, "HTTP-Based Adaptive Streaming for Mobile Clients using Markov Decision Process," in *Proc. 20th International Packet Video Workshop (PV 2013)*, pp. 1–8, Dec. 2013.
- [121] L. De Cicco, S. Mascolo, and V. Palmisano, "Feedback Control for Adaptive Live Video Streaming," in *Proc. Second Annual ACM Conference on Multimedia Systems (MMSys '11)*, pp. 145–156, 2011.
- [122] G. Tian and Y. Liu, "Towards Agile and Smooth Video Adaptation in Dynamic HTTP Streaming," in *Proc. 8th International Conference on Emerging Networking Experiments and Technologies (CoNEXT '12)*, pp. 109–120, 2012.
- [123] X. Yin, A. Jindal, V. Sekar, and B. Sinopoli, "A Control-Theoretic Approach for Dynamic Adaptive Video Streaming over HTTP," *SIGCOMM Comput. Commun. Rev.*, vol. 45, pp. 325–338, Oct. 2015.
- [124] K. Miller, D. Bethanabhotla, G. Caire, and A. Wolisz, "A Control-Theoretic Approach to Adaptive Video Streaming in Dense Wireless Networks," *IEEE Transactions on Multimedia*, vol. 17, pp. 1309–1322, Aug 2015.
- [125] P. Xiong, J. Shen, Q. Wang, D. Jayasinghe, J. Li, and C. Pu, "NBS: A Network-Bandwidth-Aware Streaming Version Switcher for Mobile Streaming Applications under Fuzzy Logic Control," in *Proc. First International Conference on Mobile Services*, pp. 48–55, June 2012.
- [126] D. Vergados, A. Michalas, A. Sgora, and D. Vergados, "A control-based algorithm for rate adaption in MPEG-DASH," in *Proc. 5th International Conference on Information, Intelligence, Systems and Applications (IISA 2014)*, pp. 438–442, Jul. 2014.
- [127] A. Sobhani, A. Yassine, and S. Shirmohammadi, "A Fuzzy-based Rate Adaptation Controller for DASH," in *Proc. 25th ACM Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV '15)*, pp. 31–36, 2015.
- [128] M. Claeys, S. Latré, J. Famaey, and F. De Turck, "Design and Evaluation of a Self-Learning HTTP Adaptive Video Streaming Client," *IEEE Communications Letters*, vol. 18, pp. 716–719, Apr. 2014.
- [129] Y. L. Chien, K. C. J. Lin, and M. S. Chen, "Machine learning based rate adaptation with elastic feature selection for HTTP-based streaming," in *Proc. IEEE International Conference on Multimedia and Expo (ICME '15)*, pp. 1–6, June 2015.
- [130] V. Menkovski and A. Liotta, "Intelligent control for adaptive video streaming," in *Proc. IEEE International Conference on Consumer Electronics (ICCE '13)*, pp. 127–128, Jan 2013.

- [131] J. van der Hooft, S. Petrangeli, M. Claeys, J. Famaey, and F. D. Turck, "A learning-based algorithm for improved bandwidth-awareness of adaptive streaming clients," in *IFIP/IEEE International Symposium on Integrated Network Management (IM '15)*, pp. 131–138, May 2015.
- [132] S. Akhshabi, S. Narayanaswamy, A. C. Begen, and C. Dovrolisa, "An experimental evaluation of rate-adaptive video players over http," *Signal Processing: Image Communication*, vol. 27, pp. 271–287, Apr. 2012.
- [133] C. Müller, S. Lederer, and C. Timmerer, "An Evaluation of Dynamic Adaptive Streaming over HTTP in Vehicular Environments," in *Proc. 4th Workshop on Mobile Video (MoVid '12)*, pp. 37–42, 2012.
- [134] K. Miller, E. Quacchio, G. Gennari, and A. Wolisz, "Adaptation algorithm for adaptive streaming over HTTP," in *Proc. 19th International Packet Video Workshop (PV)*, pp. 173–178, May 2012.
- [135] S. Benno, A. Beck, J. Esteban, L. Wu, and R. Miller, "WiLo: A Rate Determination Algorithm for HAS video in wireless networks and low-delay applications," in *IEEE Globecom Workshops 2013*, pp. 512–518, Dec. 2013.
- [136] C. Zhou, C.-W. Lin, X. Zhang, and Z. Guo, "A Control-Theoretic Approach to Rate Adaption for DASH Over Multiple Content Distribution Servers," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 24, pp. 681–694, Apr. 2014.
- [137] "H264/AVC Video Trace Library," [Online] Available: <http://trace.eas.asu.edu/h264/>. Accessed: 2015-10-30.
- [138] L. Rizzo, "Dummynet: A Simple Approach to the Evaluation of Network Protocols," *SIGCOMM Comput. Commun. Rev.*, vol. 27, pp. 31–41, Jan. 1997.
- [139] T. Wiegand, G. Sullivan, G. Bjontegaard, and A. Luthra, "Overview of the H.264/AVC video coding standard," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 13, pp. 560–576, Jul. 2003.
- [140] S. Wei and V. Swaminathan, "Low Latency Live Video Streaming over HTTP 2.0," in *Proc. 24th International Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV '14)*, pp. 37–42, Mar. 2014.
- [141] S. Wei and V. Swaminathan, "Cost effective video streaming using server push over HTTP 2.0," in *Proc. 16th IEEE International Workshop on Multimedia Signal Processing (MMSP2014)*, pp. 1–5, Sep. 2014.
- [142] M. Belshe, R. Peon, and M. Thomson, "Hyper transfer protocol version 2 (HTTP/2)," in *RFC 7540*, May 2015.
- [143] Working draft for ISO/IEC 23009-6, "DASH over Full Duplex HTTP-compatible Protocols (FDH)," Oct. 2015.

- [144] D. V. Nguyen, H. T. Le, P. N. Nam, A. T. Pham, and T. C. Thang, "Request adaptation for adaptive streaming over HTTP/2," in *Proc. IEEE International Conference on Consumer Electronics (ICCE '16)*, pp. 189–191, Jan 2016.
- [145] D. V. Nguyen, H. T. Le, P. N. Nam, A. T. Pham, and T. C. Thang, "Adaptation method for video streaming over HTTP/2," *IEICE Communications Express*, vol. 5, no. 3, pp. 69–73, 2016.
- [146] "Usage of HTTP/2 for websites," [Online] Available: <https://w3techs.com/technologies/details/ce-http2/all/all>. Accessed: Jul. 17, 2017.
- [147] S. Wei, V. Swaminathan, and M. Xiao, "Power efficient mobile video streaming using HTTP/2 server push," in *Proc. 17th IEEE International Workshop on Multimedia Signal Processing (MMSP2015)*, pp. 1–6, Oct 2015.
- [148] J. van der Hooft, S. Petrangeli, N. Bouten, T. Wauters, R. Huysegems, T. Bostoen, and F. D. Turck, "An HTTP/2 push-based approach for SVC adaptive streaming," in *IEEE/IFIP Network Operations and Management Symposium*, pp. 104–111, Apr. 2016.
- [149] M. Xiao, V. Swaminathan, S. Wei, and S. Chen, "Evaluating and Improving Push Based Video Streaming with HTTP/2," in *Proc. 26th International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV '16)*, pp. 3:1–3:6, May 2016.
- [150] J. van der Hooft, S. Petrangeli, T. Wauters, R. Huysegems, P. R. Alface, T. Bostoen, and F. D. Turck, "HTTP/2-Based Adaptive Streaming of HEVC Video Over 4G/LTE Networks," *IEEE Communications Letters*, vol. 20, pp. 2177–2180, Nov. 2016.
- [151] V. Paxson, M. Allman, H. J. Chu, and M. Sargent, "Computing TCP's Retransmission Timer," 2011.
- [152] S. Thakolsri, W. Kellerer, and E. Steinbach, "QoE-Based Cross-Layer Optimization of Wireless Video with Unperceivable Temporal Video Quality Fluctuation," in *Proc. IEEE International Conference on Communications (ICC '11)*, pp. 1–6, June 2011.
- [153] B. Keelan and H. Urabe, "ISO 20462, A psychophysical image quality measurement standard," *Proc. SPIE*, vol. 5294, pp. 181–189, 2004.
- [154] J. Janssen, T. Coppens, and D. D. Vleeschauwer, "Quality assessment of video streaming in the broadband era," in *Workshop on Advanced Concepts for Intelligent Vision Systems (ACIVS'2002)*, pp. 38–45, Sep. 2002.