TOWARDS A QOE-AWARE VIDEO STREAMING SYSTEM

KA-PUI MOK

Ph.D

The Hong Kong Polytechnic University

2016

The Hong Kong Polytechnic University
Department of Computing

Towards a QoE-aware Video Streaming System

Ka-Pui Mok

A thesis submitted in partial fulfilment of the requirements for the degree of Doctor of Philosophy

March 2016

CERTIFICATE OF ORIGINALITY

I hereby declare that this thesis is my own work and that, to the best of my knowledge and belief, it reproduces no material previously published or written, nor material that has been accepted for the award of any other degree or diploma, except where due acknowledgement has been made in the text.

_____ (Signed)

Ka-Pui Mok

_____ (Name of student)

To my wife and my family

Abstract

Online video streaming is one of the most popular web application nowadays. Billions of users watch videos on YouTube, while Netflix has 60 millions of subscribers around the world. These video service providers use HTTP streaming to transfer video data to the users over the existing web architecture. HTTP streaming gains its popularity rapidly, because it can help video data traverse the firewall and the Network Address Translation (NAT). The Content Delivery Network (CDN) can also be leveraged to largely increase the scalability. HTTP Adaptive Streaming (HAS), a newer version of HTTP streaming, can support video bitrate adaptation, which can adjust the video bitrate according to the network condition. Although video service providers can easily obtain the network Quality of Service (QoS) data, ordinary Internet users do not have sufficient knowledge about the network QoS. They concern more about their Quality of Experience (QoE), which represents their overall perceived quality. However, it is very challenging for service providers to evaluate the QoE because of its the subjective nature.

In this research, we contribute to three aspects of the QoE of HTTP video streaming system—QoE measurement, QoE improvement, and QoE assessment. QoE measurement is to understand the impact from the network quality on the QoE of HTTP video streaming. We investigate the relationships among the network QoS, application layer events, and the QoE. A set of Application Performance Metrics (APM) is proposed to quantify the application layer events. The APM is further correlated with the QoE

collected from the subjective assessments. Our results show that the rebuffering events can adversely affect the QoE. Our follow-up studies further show that both the initial and the abrupt change of video bitrate in HTTP adaptive streaming can affect the QoE. The MOS can be reduced by 17% if a sub-optimal initial bitrate is chosen.

One of the key issues leading to sub-optimal QoE in HTTP adaptive streaming is that the streaming system lacks accurate network measurement data to support the selection of video bitrate. This can result in rebuffering events or unnecessary bitrate switching. However, it is challenging to perform lightweight and accurate network measurement on the clients' browsers. To improve the QoE, we propose a server-side measurement paradigm, which executes the main measurement logic in a middlebox installed in front of the streaming server. With this framework, we design and implement two systems, `IRate` and `QDASH`, to support measurement before and during the video stream (i.e., *pre-stream* and *mid-stream* stages), respectively. `IRate` exploits the pre-stream time window to probe the network, so that a quick estimation of the network condition can be performed, and the best initial video bitrate can be estimated at the onset of the streaming. Our results show that `IRate` can achieve an accuracy of 80% by performing 10s of measurement. After the video streaming starts, `QDASH` hijacks the video flow to carry out *inline measurement*. By carefully designing the packet sending order and rate, we can conduct packet train-based available bandwidth measurement and estimate the video bitrate the network can support. We compare the amount of time required for obtaining the correct throughput between `QDASH` and throughput averaging method commonly used in video players. Our testbed experiment shows that `QDASH` can respond quicker than the harmonic mean of throughput data for at least 5s.

The final part of this research is on enhancing the scalability and reliability of QoE assessments by employing user behavior analytics. Traditional subjective assessment in a controlled environment using Mean Opinion Score (MOS) does not scale. Dis-

tributing a set of customized video can increase the participation, but the subjective assessment alone has its limitations. Crowdsourcing is aimed at further scaling the QoE measurement. However, screening out low-quality workers is an inherent and unsolved challenge for this approach. Subjective assessment is also hard to be conducted in real-world environment, because users are not responsive to the assessment. We propose user-behavior analytics to improve the video QoE assessments. User behaviors, such as pause events, mouse click events and cursor trajectory, contain rich information reflecting users' cognitive processes. We record and analyze these user behaviors while they review videos in customized video players or crowdwsourcing platform. We found that these user-behavior data can significantly improve the explanatory power of QoE model for the MOS by 8%. Our worker behavior based approach can detect around 80% of low-quality users in crowdsourcing platforms.

Acknowledgements

Foremost, I would like to express my sincere gratitude to my advisor, Prof. Rocky Chang for his tremendous and continuous support and guidance of my Ph.D study. Rocky gives me much freedom on choosing and experimenting the topic that I am interested in. This is a very good training for me to develop as an independent researcher. He also spends his best effort to seek suitable resources to support me, so that I can focus on the work in this thesis.

Apart from my advisor, I would like to express my gratitude to the rest of the Board of Examiners: Prof. Dah Ming Chiu of The Chinese University of Hong Kong, Prof. Yong Liu of New York University, and Prof. Jia You of The Hong Kong Polytechnic University. Their comments and suggestions are useful for improving this work.

I particularly thank Edmond Chan and Daniel Xiapu Luo for their mentorship, especially at the early stage of my study. Edmond inspired me to investigate the problem of the QoE and guided me a lot on network measurement research. Daniel was very supportive and gave insightful comments to my work. I would also like to thank the past and current fellow members and student helpers in the Internet Infrastructure and Security Research Laboratory: Weichao Li, Star Poon, Steven Chien, Anson Kwan, Peter Membrey, Toby Lam, Steven Lee, Waiting Fok, Curtis Yung, Jack Chan, Daoyuan Wu, Brent Zhou, Monica Leung, Xue Lei, Wei Yu, Ang Chen, Peixin Chen, Qingjie Xu, Yujing Liu, Mckeith Kwok, Piotr Dylis, Mike Cumming, and Sherry Lin.

I am grateful to external collaborators: Dr. Zhigang Sun and his team of National

University of Defense Technology for their support and recommendation in implementing measurement tools with the NetMagic; Prof. kc claffy and the CAIDA of University of California, San Diego for their warm reception during my visit in February 2016.

I thank the anonymous reviewers in reviewing my paper submissions to conferences/journals and giving insightful comments. Moreover, I want to thank the Innovation and Technology Fund (ITF) and the Joint University Computer Centre (JUCC) for their generous support. This work is partially supported by three ITF projects ("Uncooperative Measurement and Monitoring of Internet Path Quality with Applications", "Reliable and Accurate Bandwidth Measurement of Asymmetric Network Paths", and "Design and Implementation of a Unified Box for Offering Network Path Measurement as a Service") and the JUCC consultancy project ("Performance Monitoring and Measurement of HARNET").

Last but not least, I am highly indebted to my family. I particularly thank my wife, Cass Shum, for her love and care. Her emotional support gives me energy to finish this long journey. I also want to mention our pet bunny, Mr. Black. He gives I and Cass unlimited joy and fun in the last year. I am thankful to my parents, who give me the support and opportunity to pursue my research career. I do not think I can ever repay the debt I owe them all.

Contents

Abstract **v**

Acknowledgements **viii**

List of Figures **xiv**

List of Tables **xvii**

List of Abbreviations **xix**

1 Introduction **1**
1.1 QoE Measurement 4
1.2 QoE Improvement 5
1.2.1 Pre-stream network measurement 6
1.2.2 Mid-stream network measurement 8
1.3 QoE Assessment 9
1.3.1 Inferring the QoE from user-viewing activities 9
1.3.2 Detecting low-quality workers in QoE crowdtesting by worker behavior 10
1.4 Contributions 11
1.5 Organization 13

2 Background and Related Work **16**
2.1 HTTP streaming system 17
2.2 Evaluating the performance of HTTP streaming 18
2.2.1 Testbed evaluations 18
2.2.2 Internet experiment 20
2.3 Measuring the QoE of HTTP streaming 21
2.3.1 Subjective assessment based approach 21
2.3.2 Pseudo subjective metrics based approach 23
2.3.3 Model based approach 24
2.4 Improving the QoE of video streaming 25
2.4.1 Quality adaptation algorithms 25
2.4.2 Server-side and middleboxes approach 32

2.4.3 Initial video bitrate selection . . . 34
2.5 User-behavior based QoE assessment . . . 36
2.6 Detecting low-quality workers in QoE crowdtesting . . . 37

3 Understanding the Influencing Factors to the QoE 39
3.1 Network QoS and application QoS . . . 41
3.1.1 Application performance metrics . . . 42
3.1.2 Modeling the APMs . . . 43
3.1.3 Measuring the APMs from clients . . . 46
3.1.4 Testbed experiments . . . 46
3.2 The QoE measurement . . . 50
3.3 Correlating QoE with network QoS . . . 53
3.3.1 Visualizing the correlation . . . 53
3.3.2 Applications . . . 55
3.4 Impact of network path asymmetry . . . 56
3.5 Measuring the influence of bitrate adaptation to the QoE . . . 59
3.5.1 Evaluating the QoE impact on video quality down-switching . 59
3.5.2 The impact of initial bitrate setting to the QoE . . . 66
3.6 Summary . . . 70

4 Improving the Initial Video Bitrate 72
4.1 Background . . . 73
4.1.1 Default settings for initial video quality . . . 73
4.1.2 Impacts of the default settings . . . 75
4.2 User-behavior driven design . . . 79
4.2.1 Design goals . . . 79
4.2.2 Pre-stream measurement window . . . 80
4.2.3 Measuring network path-quality metrics . . . 81
4.2.4 Methods for estimating the BIBR . . . 85
4.3 An IRate implementation . . . 87
4.3.1 Probe kit . . . 89
4.3.2 Quality oracle . . . 91
4.4 Evaluation . . . 93
4.4.1 Testbed experiments . . . 93
4.5 User QoE experiments . . . 102
4.6 Discussion . . . 104
4.6.1 Accessing videos from third-party video sites . . . 104
4.6.2 Scalability and security issues of IRate . . . 105
4.6.3 Short vs. long video clips . . . 105
4.7 Summary . . . 106

5 Towards an Accurate Bandwidth Estimate for DASH Video Streams 107
5.1 Overview of QDASH . . . 109
5.2 Measuring available bandwidth for DASH . . . 111
5.2.1 Assumptions . . . 111
5.2.2 QDASH-abw probing methodology . . . 112
5.2.3 Determining the video quality levels . . . 115
5.2.4 Evaluating QDASH-abw . . . 116
5.3 A QoE-aware switching algorithm . . . 122
5.3.1 Buffer-aware strategies . . . 122
5.3.2 Designing an QoE-aware switching algorithm . . . 124
5.4 Summary . . . 125

6 User-behavior Analytics for QoE Assessment 128
6.1 User-viewing activities . . . 129
6.1.1 The overall methodology . . . 131
6.2 Evaluation . . . 133
6.2.1 Experiment setup . . . 134
6.2.2 Client software . . . 135
6.2.3 Video materials . . . 135
6.2.4 Subjective assessment . . . 136
6.3 Results and analysis . . . 137
6.3.1 User-viewing activities and network path quality . . . 137
6.3.2 Hypothesis testing for user-viewing activities . . . 139
6.3.3 Correlating user-viewing activities with QoE . . . 140
6.4 Summary . . . 142

7 Improving the Reliability of QoE Crowdtesting 144
7.1 Overview . . . 146
7.2 Assessing the workers' quality . . . 149
7.2.1 Complexity in text input response . . . 150
7.2.2 Violation of soft rules . . . 152
7.2.3 Contradictory responses . . . 153
7.3 Quantifying workers' behavior . . . 154
7.3.1 Observation . . . 154
7.3.2 Worker behavior metrics . . . 155
7.4 Evaluation and Results . . . 160
7.4.1 Implementation of the QoE crowdtesting task . . . 160
7.4.2 Results . . . 163
7.5 Discussion . . . 171
7.5.1 The influence of rating methods . . . 171
7.5.2 Limitation . . . 171
7.6 Summary . . . 172

8 Conclusions and Future Work **174**
8.1 Future work 176

Bibliography **179**

List of Figures

1.1 Relationships among QoE Measurement, QoE Improvement, and QoE Assessment. 4

2.1 A typical HTTP adaptive streaming system. 18

3.1 Three levels of QoS considered in this thesis. 40
3.2 Time series of the video playhead time and the amount of video data buffered at the player. 43
3.3 A testbed setup for evaluating the correlation between network QoS and application QoS. 47
3.4 The three APMs under different network path quality. 48
3.5 The CDF of relative errors between the APM estimates and actual APMs. 49
3.6 The rebuffering frequency estimates for a 87-second video clip under different buffer sizes and goodput-to-bitrate ratios. 50
3.7 MOS against levels of f_{rebuf} for four types of video clips. 53
3.8 The radar chart mapping network QoS with QoE. 54
3.9 Time series of the HTTP video streaming's MOS, and the RTT and packet loss rate of the network path. 55
3.10 The APM differences under different asymmetric loss rates, bandwidth of 5 Mbits/s, and RTTs of $\{25, 100\}$ ms. 57
3.11 The APM differences for two TCP variants under different round-trip loss rates, bandwidth of 5 Mbits/s, and RTTs of $\{25, 100\}$ ms. 58
3.12 The quality transition of $\mathbb{R}_{1,4}$, $\mathbb{R}_{3,7}$, and $\mathbb{R}_{3,8}$. 62
3.13 The overall distribution of overall QoE rating. 64

4.1 Video playhead times and buffer statuses for default initial bitrate and BIBR in classic HTTP streaming under a lossy network condition. . . 77
4.2 Quality levels vs. video playhead times for default initial bitrate and BIBR under DASH streaming in a good network condition. 79
4.3 Typical user actions and background actions before starting a streaming video. The dark-bordered boxes are the actions for typical video streaming, whereas the light-bordered boxes are added for IRate-enabled video streaming. 81

4.4 The percentage difference of average throughput against different web object sizes. 83
4.5 The number of Bytes transferred in 10-second IRate measurement against RTTs. 85
4.6 An execution-time comparison of the equation-based and decision tree methods for estimating the BIBR. 87
4.7 The main steps in IRate-enabled video streaming. 88
4.8 Deploying IRate in a large-scale video site. 89
4.9 Probe kit's measurement flow between a user's browser and the measurement middlebox. 91
4.10 The testbed topology. 94
4.11 The accuracy of IRate's estimation of the BIBRs using different pre-stream time windows. 97
4.12 The CDFs of application performance metrics. 100
4.13 The CDFs of DASH efficiency and stability metrics. 102

5.1 The overall QDASH architecture. 111
5.2 Two probe rounds of sending rates. 114
5.3 CDF of sampled RTTs under available bandwidth of 1Mbps and 2Mbps. 118
5.4 The mean and the standard deviation of sampled RTTs with different sampling rate and different available bandwidth. 119
5.5 Evaluations with four network profiles. 126
5.6 An example of buffer-aware quality transition. 127

6.1 The enhanced QoE assessment model. 129
6.2 A timeline for a video viewing session with impairment events and user-viewing activities. 133
6.3 Experiment setup for the subjective experiment. 134
6.4 The overall distribution of MOS. 137
6.5 Time series of application-level events and user-viewing activities. . . 139
6.6 Time series of the RTT and packet loss rate. 139

7.1 The overview of the workflow. 147
7.2 Mouse cursor trajectories of an honest worker *R* and a low-quality worker *C* 154
7.3 Cursor velocity of the data in Figure 7.2 155
7.4 A timeline of events. 156
7.5 A submovement example. 157
7.6 The layout of the question page. 161
7.7 The architecture of our assessment platform. 162
7.8 The CDF of completion time of each assessment (x-axis in log scale). 164
7.9 The CDFs of the quality scores. 164
7.10 The result of a principal component analysis of the seven measures. . 166
7.11 The average error rate of different models. 167

7.12 Recall and precision of CrowdMOS approach against different correlation coefficient, r_{th}. 170
7.13 Cursor trace of different rating methods from a typical worker. 173

List of Tables

2.1 Summary of different video streaming events investigated in the literatures. 23
2.2 A list of quality adaptation algorithms studied in this literature. 26
2.3 Notation used in this section. 27

3.1 The network QoS parameters emulated by the Click router. 47
3.2 A specification of the video used in the testbed experiments and QoE assessment. 48
3.3 Three levels of application performance based on the APMs. 51
3.4 The maximum number of segments supported at intermediate levels, n_{frag}. 61
3.5 Summary of experiment rule sets defining different quality transition schemes. 62
3.6 Profiles of the video quality levels [63]. 63
3.7 The MOS difference, ΔMOS(column,row). 67
3.8 Summary and the MOS of different quality transition schemes. 69

4.1 Streaming video services provided by major video service providers. . 74
4.2 A comparison of the default initial bitrates and the BIBRs for sample videos. 78
4.3 Input attributes and class variable for decision tree building. 92
4.4 Network path parameters used for generating the training data. 95
4.5 Distribution of network types. 103

6.1 A list of possible user-viewing activities with the effect of technical impact, mean rating difference, explicit and implicit meaning. 132
6.2 Path quality settings used in the Click modular router. 134
6.3 Profiles of the video clips. 136
6.4 Average time distance for different user-viewing activities and impairment events. 141
6.5 Regression results for the APM model and modified model. 142

7.1 The rating schemes and examples for rating the responses. 151
7.2 A list of collected user behavior. 156
7.3 The sample display of the four rating methods used in our task. 161

7.4 Correlation analysis among metrics of quality score. 165
7.5 Worker behavior metrics used in various models. 167
7.6 The percentage of workers showing differences between the actual and predicted worker's category. 168
7.7 A comparison of results of CrowdMOS and our method. 171

List of Abbreviations

APM Application Performance Metrics.

AVC Advanced Video Coding.

BIBR Best Initial Bitrate.

CDF Cumulative Distribution Function.

CDN Content Delivery Network.

DASH Dynamic Adaptative Streaming over HTTP.

DDoS Distributed Denial-of-Service.

EWMA Exponential Weighted Moving Average.

HAS HTTP Adaptive Streaming.

HD High Definition.

HTTP HyperText Transfer Protocol.

IDT Inter-Departure Time.

ISP Internet Service Provider.

MDP Markov Decision Process.

MOS Mean Opinion Score.

MPC Model Predictive Control.

MPD Media Presentation Description.

MSS Maximum Segment Size.

MTurk Amazon Mechanical Turk.

NAT Network Address Translation.

OS Operating System.

OSMF Open Source Media Framework.

PID Proportional-Integral-Derivative.

PSNR Peak-Signal-to-Noise-Ratio.

QoE Quality of Experience.

QoS Quality of Service.

RNN Random Neural Network.

RTT Round-Trip Time.

SDN Software Defined Network.

SMP Strobe Media Playback.

SQAD Shared Question Answer Dictionary.

SSCQE Single Stimulus Continuous Quality Evaluation.

SSIM Structural Similarity Index.

SVC Scalable Video Coding.

TCP Transport Control Protocol.

TCP ACK Pure TCP Acknowledgement Packet.

TVSQ Time-Varying Subjective Quality.

UDP User Datagram Protocol.

VoD Video-on-Demand.

Chapter 1

Introduction

Video streaming is unarguably an extremely popular application in the Internet nowadays. For example, YouTube has billions of views every day [245], while Netflix has over 65 million subscribers in over 50 countries [177]. In terms of Internet traffic volume, video traffic accounts for more than half of the downstream traffic in North America [208]. A Cisco white paper predicts that three-quarters of the world's mobile data will be video by 2019 [59].

The boom of online video streaming is partially due to the introduction of *progressive download* Adobe Flash. The Flash player running on the browser sends HTTP `GET` requests to a web server to download video data. Users can start to watch the video without completely downloading the whole video clip. Comparing to tradition UDP-based streaming, HTTP-based video streaming can easily traverse the network middleboxes, such as firewalls and Network Address Translation (NAT) boxes. However, progressive download does not support timeshifting seek, which allows users to jump to a timecode where the video data has not been downloaded. Later on, *HTTP streaming* introduces the "trick-play" technique, where the server can locate the keyframe nearest to the required time point.

A recent advance in HTTP streaming, namely *HTTP adaptive streaming* (HAS),

can switch the video bitrate according to various criteria, such as network conditions. The video clips are encoded into several (usually from 5 to 12) quality levels/representations using different resolution and bitrate settings. Each version is further divided into a number of video segments containing 2 to 10 seconds of video data. The player can download and stitch video segments of different quality levels to produce a smooth playback. Dynamic Adaptative Streaming over HTTP (DASH) is a standard published by MPEG [217][1]. Industrial implementations of HAS include Adobe HTTP Dynamic Streaming [15], Apple HTTP Live Streaming [24], and Microsoft Smooth Streaming [162].

The popularity of video streaming brings both opportunities and challenges. On the one hand, websites can easily provide better entertainment and content to attract users [104]. By using HTTP streaming, existing web architecture, such as Content Delivery Network (CDN) and web proxy, can be reused [221]. On the other hand, video streaming is more sensitive to network performance than ordinary web applications. It is challenging for the service and network providers to improve the server side and the network performance, such that the streaming system can support a huge amount of video traffic from an increasing number of users [70, 86]. However, performance degradation is still not uncommon in today's video streaming systems [105, 117, 106].

One of the reasons for the performance degradation is that the design of HTTP streaming system, especially HTTP adaptive streaming, does not incorporate a network measurement component to estimate the quality of the network path. Unlike the protocol designed for video streaming (e.g., RTP [210], RTCP, RTSP [211], and MGRP [186]), neither HTTP nor TCP can provide information on the current network status. Without the availability of accurate network measurement data, sub-optimal or even instable video bitrate, including both initial and mid-stream ones, can be chosen. This instability in bitrate selection can result in flicker and/or sudden drop in video

[1] In this thesis, HAS and DASH are used interchangeably.

bitrate, which can be annoying to users [179, 170].

Another challenge is to estimate the Quality of Experience (QoE) of users, which is defined as "the overall acceptability of an application or services, as perceived subjectively by the end-user" [111]. In addition to the network performance quantified by the Quality of Service (QoS), the QoE also depends on a number of human factors, such as past experience, expectations, and/or enjoyment level [37, 40]. For example, users may expect the video streaming supporting a High Definition (HD) or even 4K resolution rather than 360p or 480p for their large screen TVs. They may also have a higher tolerance to buffering events when they watch video through the mobile network.

The subjective nature of the QoE makes it difficult to assess in real-world environment. Measuring the QoE requires users to rate their perceived quality. Mean Opinion Score (MOS) [112] is often used as a measure of the QoE (1:Bad – 5:Excellent). However, users are often reluctant to provide feedback until they experience poor performance. Therefore, the results may not be able to reveal the QoE of unsatisfied users. User engagement [70, 27] is proposed to be a pseudo subjective measure of QoE. Although this metrics can be obtained without explicit feedback from users, other confounding factors, such as video content, can also affect the engagement level. Furthermore, some subjective factors, such as user expectation, cannot be revealed by this metrics.

This thesis presents a suite of works studying three inter-related problems–QoE measurement, QoE improvement, and QoE assessment of HTTP video streaming. Figure 1.1 shows their relationships. QoE measurement evaluates the performance and studies the correlation among network layer, application layer, and the perceived quality of HTTP streaming systems, which is essential for revealing impairments that degrade the QoE. With the measurement findings, QoE improvement enhances the HTTP streaming system by introducing better network measurement methods and consider-

ing the user perceived quality. The final step of the cycle is QoE assessment. This step focuses on the new subjective assessment methodologies, which can enhance the reliability of QoE measurement.

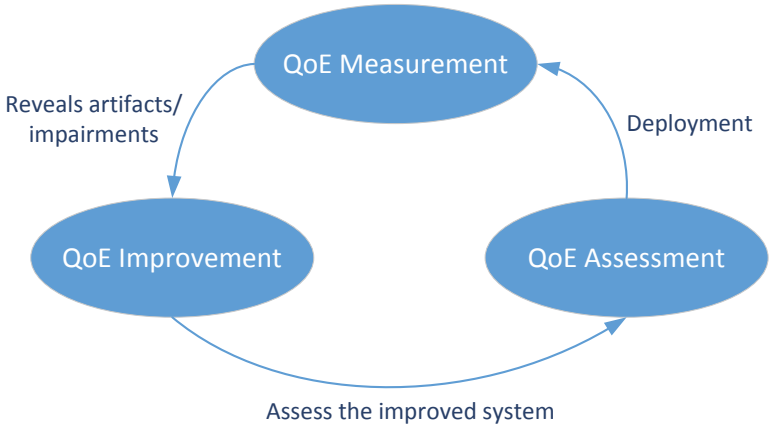


Figure 1.1: Relationships among QoE Measurement, QoE Improvement, and QoE Assessment.

In the rest of this chapter, we introduce the QoE measurement and related problems in HTTP video streaming in Section 1.1. The problems motivate us to design and implement `IRate` and `QDASH` to be introduced in Section 1.2. We then present the motivations for the QoE assessment with user-behavior analytics in Section 1.3. Finally, we summarize the contributions and organization of this thesis in Section 1.4 and Section 1.5, respectively.

1.1 QoE Measurement

Measuring the QoE of online video streaming is a challenging and important topic for both service and network providers in recent years. A number of standards and methods have been developed to measure the perceived quality of multimedia content in UDP-based streaming (e.g., [113, 114, 110]). HTTP streaming delivers video data via HTTP over TCP. The reliable features of TCP relieve the video codec from handling packet losses, and the picture quality is not affected by unrecoverable video data loss.

However, the TCP throughput can be reduced due to long network latency or packet loss. When the throughput is unable to support the video bitrate, the consumption rate of the video buffer will be faster than the data arrival rate. As a result, the playback will be paused to wait for more video data after the buffer is depleted.

In the first part of the QoE measurement, we investigate the impact of network QoS on the QoE in HTTP streaming. However, directly inferring the QoE from the network QoS is difficult in HTTP streaming because users cannot easily perceive the network quality. Therefore, a set of *application performance metrics* (APM) [165] is proposed to quantify the application level events. A testbed experiment is setup to measure the correlation between the network QoS and the APM. Furthermore, we conduct subjective experiment, which is one of the first studies in this research domain, to study the relationships between the APM and the QoE. Our results show that rebuffering events can seriously impact the QoE. We further apply a regression model to predict the QoE from the APM.

The introduction of HAS can improve the video quality or reduce the chance of rebuffering events by automatically choosing the video bitrate. However, the fluctuation of network throughput can cause abrupt changes to video quality [19]. Therefore, we design subjective experiments to investigate the impact of switching up/down of the video quality levels on the QoE in HTTP adaptive streaming. We find that inserting an *intermediate level* between the original and the target quality level can mitigate the impact on QoE in switching down the video quality. Moreover, users prefer a better initial quality rather than a stepwise switching up of the quality level.

1.2 QoE Improvement

From our subjective experiments, we find that reducing rebuffering events, a smooth change of quality levels, and better initial quality can improve the QoE of HTTP

streaming and HAS. These three directions are closely related to the accuracy of network throughput estimation. However, obtaining accurate network measurement in the browser is difficult [88, 143]. While most of the research focus on filtering noise in throughput measurement or utilizing buffer information on the client side to decide the video bitrate, we advocate a server-side paradigm which enables the HTTP streaming system to deploy lightweight and accurate measurement tools in a measurement middlebox. We design and implement two systems using this paradigm, `IRate` [169] and `QDASH` [170], to support *pre-stream* and *mid-stream* measurement.

1.2.1 Pre-stream network measurement

A major effort in improving the HAS was spent on proposing mid-stream measurement methods to update the best bitrate during the video stream. We argue that an equally, if not more, important problem is to determine the *best initial* bitrate (*BIBR*), because the default setting cannot accommodate diverse end-to-end network quality and end systems. Without any network measurement as a benchmark, a "safe" approach is to choose a conservative bitrate to start with. This approach, however, results in suboptimal QoE, especially for short video clips, because of the following two reasons.

1. Even though the quality adaptation algorithm in DASH can ramp up the video bitrate and find the best rate quickly on the client side, the video player cannot discard the low-bitrate video segments already downloaded during this process. The user therefore has to suffer from low initial video quality for a period of time. This start-up phase can be longer than 150s of video [106]. A recent study [251] shows that existing algorithms achieve only 14%-20% of optimal quality in the first 32 seconds. Our subjective tests also show that using a low initial bitrate can degrade the overall QoE as much as 17%.

2. Starting from a low video bitrate also induces unnecessary quality switching

during the adaptation process. It has been shown that a frequent switching of bitrates can hurt user engagement [70] and also the QoE [191, 61, 179].

We propose to introduce *pre-stream network measurement* into existing video streaming systems to determine the BIBR with a reasonable accuracy before the video playback starts. There are three main challenges in integrating such measurement. First, we need to identify a suitable time frame for carrying out the measurement, such that the measurement results can be ready before the onset of the video. Second, we need to mitigate the measurement inaccuracy at the browser level. Lastly, we need to consider whether the measurement can be seamlessly integrated into the existing web architecture and provide accurate path quality metrics for estimating the BIBR.

In this research, we present a practical system called `IRate` to determine the BIBR. Our user behavior driven design helps explore possible time frames for pre-stream measurements. Therefore, sufficient measurement result can be collected before the onset of the video. Moreover, we exploit a number of tactics in the TCP/IP layer and web technology to implement the measurement component. The measurement core of `IRate` is designed as a measurement box, which can be easily deployed along with the existing video caches and servers to collect accurate network path quality data at the server side. The measurement core can accommodate packet-pair based bandwidth estimation algorithms to measure the network quality at the server side. By imbedding a script in web pages, clients can measure dedicated `IRate`-enabled video caches through the browser. Based on the measurement results, `IRate` profiles clients by determining their BIBR to video caches. The web server utilizes this information and redirects users to a better video cache which could serve the highest BIBR for the best QoE. `IRate` can predict the BIBR with 80% of accuracy with 10s of pre-stream measurement.

1.2.2 Mid-stream network measurement

After the video streaming is started, the benefit from the BIBR diminishes. The quality adaptation algorithm running in the video player takes over the task of updating the most suitable video bitrate. The algorithm performs *mid-stream measurement* and obtains estimates of the network throughput by dividing the number of downloaded bytes by the download time. However, this kind of application layer-based throughput measurement requires careful calibration. For example, multiple TCP connections rather than a single one, are preferred [88] and a warm-up transfer should be performed to mitigate the effect of TCP slow start [224].

It is hard to perform calibrated throughput measurement in HTTP streaming, because the video player initiates only one TCP flow to stream the video, and the sizes of video segments can vary. Therefore, the throughput measured by the player can experience a large fluctuation [105]. Many methods are proposed to filter the noise or short-term fluctuations in the throughput measurement to avoid frequent switching of the video bitrate (e.g., [152, 229, 77, 149, 64, 118, 145, 223]). On the other hand, the buffer-based algorithm considers solely the information from the video buffer to adjust [106] the video bitrate.

In our research, instead of deciding the video bitrate based on inaccurate throughput measurement results, we propose `QDASH-abw` [170], which is a novel probing methodology to detect the highest video bitrate the current network conditions can support. Similar to `IRate`, `QDASH-abw` is designed to be deployed in a measurement box. It can manipulate the packets in a video flow to conduct *in-line* measurement and estimate the approximate available bandwidth based on the RTT variations [115]. We speed up the convergence of available bandwidth estimates by only considering a few discrete sending rates which match to the choice of the video bitrate. We also exploit the packet sending sequence to increase the number of RTT samples. We examine

the performance of our method in four different scenarios used in [19]. The results show that our method can detect available bandwidth changes much quicker than the harmonic mean of the network throughput for at least 5 seconds.

1.3 QoE Assessment

The objective measures, such as network QoS and application level events, can only partially reflect the QoE, because the subjective nature of QoE cannot easily be revealed without involving users. Therefore, a reliable QoE assessment is necessary to understand the factors influencing user perceived quality. One promising method is to conduct subjective assessments in a laboratory environment and compute the MOS for each stimulus. However, this kind of assessment cannot be applied to uncontrolled environments, such as real-world video streaming systems and crowdsourcing-based QoE assessments. In this study, we propose user-behavior analytics to alleviate QoE assessment problems in these environments. User-behavior analytics is the process of analyzing the actions or events generated by the users to uncover subjective information. It is particularly important in subjective assessments, because user behavior can potentially connect to their cognitive process [219].

1.3.1 Inferring the QoE from user-viewing activities

In the real-world environment, obtaining user feedback is challenging. Users can be non-responsive, because they lack incentive unless the service level is unacceptable. Forcing users to respond can be irritating to them. Therefore, assessing the QoE without interrupting users is not a straightforward task. We tackle this problem by investigating the movements and actions generated by users, referred to *user-viewing activities*, including pausing the video and changing the video bitrate manually in HTTP

streaming. Users are more motivated to perform these activities, because based on their past experience these actions can actually improve the streaming performance. For example, users can pause the video to buffer more video data when the network throughput is low.

We propose employing the user-viewing activities to estimate the subjective parameters in HTTP video streaming [166]. We incorporate the user-viewing activities to infer the QoE. In our implementation, the user-viewing activities and the application layer performance can be returned to the server together. Thus, the service providers can understand their users' QoE with subjective metrics without explicitly requiring their feedback.

1.3.2 Detecting low-quality workers in QoE crowdtesting by worker behavior

QoE crowdtesting [95] becomes increasingly popular among researchers to conduct QoE assessments on crowdsourcing platforms, including Amazon Mechanical Turk (MTurk) [21], Microworkers [3], and CrowdFlower [1]. The QoE of different network services, such as video streaming [125, 183, 97, 194], VoIP, and IPTV [46], can be evaluated using the crowdsourcing approach. Furthermore, the quality of multimedia materials can also be evaluated, including images [201], audio [132], or speech [36]. Experimenters can easily deploy their experiments by compiling the assessments as a website published on the crowdsourcing platform. After finishing the assessments, the workers can report to the platform to claim their payment.

The advantages of using crowdtesting over traditional laboratory experiments are lower cost and a larger, more diverse crowd of workers [95, 242, 185]. However, without any supervision, the quality of the work received from crowdtesting is questionable [236] due to various factors [124]. For example, a previous study [66] has shown that

less than 25% of Indian workers can achieve precision higher than 50%. Another study [199] also shows that the precision of Indian workers is only 70%. Some cheaters only intend to maximize their payment with minimum effort by quickly submitting the assessments. Even if workers do not intend to cheat, they can be distracted or unsuitable for the task. Both kinds of workers can lead to unreliable measurements. Therefore, identifying these workers can help improve the reliability of crowdsourcing-based assessments.

In this study, we propose a novel method to detect low-quality workers in QoE crowdtesting by analyzing the worker's behavior. Our approach is to construct a predictive model using supervised learning algorithms. A *quality score* is computed by applying existing anti-cheating techniques in the question design and human inspections to label the workers. A set of ten *worker behavior metrics* is defined as the feature vector, which quantifies different types of worker behavior, including finer-grained cursor trajectory analysis. A multiclass Naïve Bayes classifier is applied to train a model to predict the quality of workers from the metrics. We have conducted video QoE assessments on Amazon Mechanical Turk and CrowdFlower to collect the worker's behavior traces. Our results show that the error rates of the model trained from four metrics are no more than 30%. We employ four different 5-point Likert scale rating methods, and find that combining the predictions from the four rating methods can further improve the success rate in detecting low-quality workers to around 80%. Our method is 16.5% and 42.9% better in precision and recall, respectively, than CrowdMOS [202].

1.4 Contributions

The contributions of this dissertation are as follows:

Measurement studies to correlate the network QoS, Application performance, and the QoE. We have conducted a seminal study of measuring the QoE of HTTP streaming. We investigate the correlation among the network QoS, application layer events, and the QoE in HTTP streaming by performing both testbed evaluation and subjective experiments.

We have established a set of application performance metrics (APM) to quantify the performance of video streaming. We further correlate this set of metrics with the network QoS and the QoE by a model, and regression analysis, respectively. This set of metrics have been widely employed for evaluating the performance of HTTP streaming and also HAS.

We have designed and carried out subjective experiments to systematically investigate the impact of switching up/down of video bitrate on the QoE. Our findings have been cited by over 200 papers in designing quality adaptation algorithms, measuring the QoE of video streaming, and enhancing the HTTP-based video streaming system.

A novel middlebox for improving the QoE of HAS. We have proposed a novel measurement middlebox on the server side to support optimized-probing network measurement methods in HTTP video streaming system. The middlebox incorporates measurement components into the streaming system to improve both selection of initial video bitrate and the available bandwidth measurement, while the changes in both server side and client side can be kept to minimal.

We have designed and implemented `IRate` which can perform packet-pair-based (TRIO [42]) pre-stream measurement to estimate the network quality and predict the best initial bitrate (BIBR). We have identified a suitable time period (pre-stream time window) for carrying out the pre-stream measurement. We show that `IRate` can reduce the rebuffering events in HTTP streaming by 25.7% and improve the efficiency and stability of HAS by 36% and 33%, respectively.

We have presented QDASH which can manipulate video data packets in the video flows to conduct *in-line* packet-train-based (pathload [115]) mid-stream measurement. QDASH uses the round-trip time (RTT) information to determine the highest video bitrate the network can support. We show that QDASH can respond to changes in network conditions quicker than measuring the average throughput of downloading video segments.

User-behavior analytics for QoE inference and detecting low-quality QoE crowdtesting workers. We have conducted subjective assessments to analyze the *user-viewing activities* generated by users during the video playback. We show that the application events/impairments can trigger some user-viewing activities, which can be quantified and incorporated into the APM model. The estimation of the QoE can be improved by 8%.

We have proposed a user-behavior based method to detect low-quality workers in QoE crowdtesting. We have designed a set of *worker behavior metric* to systematically extract information from the raw behavior traces collected from the browser when the workers answer the questions. We show that the metrics can be used to build a machine learning based model to identify low-quality workers. Furthermore, we show that the four rating methods we tested can trigger different behavior in the cursor trace. The accuracy of the detection can reach 80% by combining the results from three of them.

1.5 Organization

The rest of this thesis consists of six chapters: Chapter 2 discusses the background and related work, Chapter 3 on QoE measurement, Chapter 4 and Chapter 5 on QoE improvement, Chapter 6 and Chapter 7 on QoE assessment, and finally Chapter 8 presents conclusions and future work.

In Chapter 2, we will first present the background of HTTP streaming and HAS.

Next, we will review some testbed and Internet performance evaluations and their findings. We will present some QoE measurement works by classifying them into three different approaches. After that, we will provide an overview of existing works on improving the QoE of this system. In addition to quality adaptation algorithms in HAS, we will summarize server-side and middlebox solutions. Lastly, we will discuss on user-behavior based QoE assessments and countermeasures against low-quality workers in crowdsourcing platforms.

In Chapter 3, we will focus on the QoE measurement. We will first present our methodology on correlating among the network QoS, application QoS, and the QoE in HTTP streaming. The APM model is proposed to quantify the application QoS. We will then measure the correlation between QoE and the APM with subjective assessment, and combine and visualize the two sets of correlation results. After that, we extend our study to investigate the impact of quality adaptation on the QoE.

Chapter 4 will introduce the `IRate` system. We discover that the existing solutions of HAS often use a "default setting" for the initial video quality. This method can easily lead to a suboptimal setting (both too high or too low), because the decision does not consider the network quality. To tackle this problem, we propose `IRate`, a practical system to determine the best initial bitrate (BIBR) based on the pre-stream measurement results collected by an IRate middlebox. Our evaluations show that IRate can provide a reasonably accurate BIBR which can improve the stability and efficiency of DASH.

Chapter 5 is devoted to the `QDASH` system. We also employ the middlebox paradigm to imbed packet-train based measurement into the video flow to estimate the available bandwidth during the mid-stream stage. Using the round-trip delay information from the measurement, we can determine the highest quality level supported by the network between the client and the middlebox. We show that our method can respond to change

of network throughput quickly. Finally, by considering the QoE measurement results in Chapter 3, we propose a QoE-aware quality adaptation algorithm which takes into account of the buffer status.

In Chapter 6, we will propose a user behavior based model for QoE assessment. In particular, we incorporate a number of user-viewing activities generated during the video playback to improve the QoE inference. We first select a set of possibly relevant user-viewing activities. After that, we will evaluate the improvement of the new framework against the APM model which only considers the application level metrics.

In Chapter 7, we will present our work on detecting low-quality workers in QoE crowdtesting. We propose a novel method to extract and analyze useful information imbedded in the worker's behavior traces. We will then describe the construction of a predictive model, which can detect low-quality workers. By removing the results submitted by these workers, the reliability of QoE crowdtesting can be improved. We will show the results obtained from both Amazon Mechanical Turk and CrowdFlower.

Chapter 2

Background and Related Work

In this chapter, we discuss existing works related to this thesis. We begin with a description of the background on the HTTP adaptive streaming system. Next, we survey the testbed and Internet measurement studies on the performance of video streaming systems and also the QoE of HTTP adaptive streaming. This is important to understand the correlation among network layer, application layer, and the QoE. We will then introduce a survey on existing methods for for improving the QoE. We classify them into three types, namely client-side, server-side, and in-network, according to the location of the measuring node. In particular, we summarize the design of different quality adaptation algorithms for client-side approaches, which is the most popular type among the three. After that, we will discuss works on assessing the QoE using user behavior, such as viewing time, and video skipping. These behavior can be useful in inferring the QoE. Finally, we will discuss the existing anti-cheating methods on QoE crowdtesting, which could be based on the distribution of ratings submitted by the workers and the worker's behavior during the task.

2.1 HTTP streaming system

A basic HTTP streaming system only requires a web server (e.g., Apache or nginx) to host the video clips. The video player running on the client side simply downloads, decodes and plays the video data for users. Later on, HTTP Adaptive Streaming (HAS) is developed as a major enhancement to the HTTP streaming. The HAS can automatically switch the video quality according to different parameters to enhance the QoE. Figure 2.1 depicts the framework of a typical HAS system. At the right of the figure, we show an example of a HAS-enabled video clip. The video clip is encoded into three quality levels or *representations* (denoted by Quality Level A, B, and C in the figure) by using different sets of parameters, such as resolution, video bitrate, audio bitrate, and so on. Each representation is further divided into video segments containing 2 to 10 seconds of video data (denoted by the numbers 1, 2, 3, ... in Figure 2.1). The video segments encoded in different quality levels contain identical video content. Therefore, the adjacent video segments in different quality levels can be seamlessly stitched and played by the client. The meta-information of the video, including the number of quality levels and their bitrates, is stored in a Media Presentation Description (MPD) file along with the video clip.

The framework of the video player is shown on the left hand side of the figure. First, after the video player runs on the client's browser, it downloads the MPD from the video server through HTTP. With the information in the MPD file, the *quality adaptation algorithm* can start to fetch the first video segment. The downloaded video segments are cached at the video buffer and further decoded by the video decoder. On the other hand, the status of the download, such as the video segment sizes and the download time, is passed to the *throughput estimator* to estimate the current network goodput for the *quality adaptation algorithm*. Some algorithms also consider the buffer status and the previous states of the player for achieving better QoE. The quality

adaptation algorithm also decides *when* to download *which* quality level of the video segments.

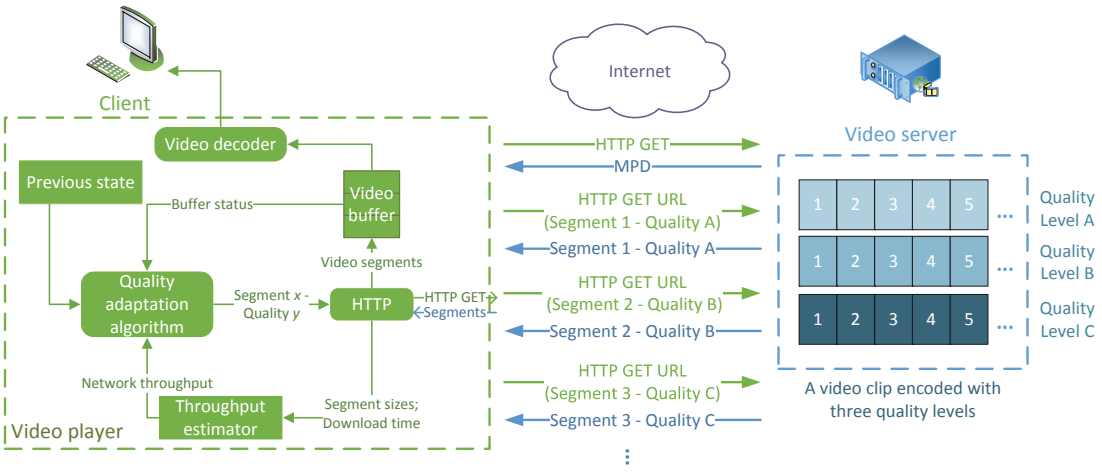


Figure 2.1: A typical HTTP adaptive streaming system.

2.2 Evaluating the performance of HTTP streaming

HTTP streaming runs on HTTP over TCP. However, HTTP is a stateless protocol. Both video players and servers cannot exchange any state information to control the video playback or detect the streaming quality. With the interaction of TCP, the streaming behavior is very different from traditional UDP based streaming. In the following, we will introduce some studies on evaluating HTTP streaming in both testbed and Internet environments.

2.2.1 Testbed evaluations

A class of work studies the behavior of HTTP streaming in different aspects under a controlled environment. Testbed experiments are carried out to evaluate industrial implementations of HTTP streaming and HAS. Biernacki and Tutschku [31] take a similar approach as our study [165], which emulates different network QoS in a testbed, to

evaluate the performance of HTTP streaming. They find a set of criteria for network QoS which can smoothly stream different quality of video. Karki et al. [123] evaluate the performance of Orb, which enables users to stream video from home networks using HTTP streaming. In [19, 30], the authors empirically evaluate commercial DASH players by emulating different bandwidth profiles. They find that some video players fail to converge to an appropriate bitrate, while some players are more aggressive in selecting a better video quality which can produce an excessive number of bitrate switching. Apart from the wired network, several studies [241, 173, 204] examine the performance of DASH players in cellular networks and find that the streaming performance is still unsatisfactory in the mobile environment which has a high variability in network throughput. Krishnamoorthi et al. [134] evaluate on the newer version of OSMF using different bandwidth traces. Besides server-client scenarios, they also evaluate the performance of HAS streaming through a proxy with different policies and conclude that the design and policy of proxies have to consider the network conditions and bottlenecks.

The aforementioned studies assume that the client is the only user of the video streaming server and the network bottleneck. In later studies [18, 231], the interaction between multiple competing HAS flows is considered. The characteristic of HAS video flows is very different from TCP bulk transfer. Because the video buffer size is limited, the video player pauses the download when the video buffer is full (*OFF* period). After some data in the video buffer is consumed, the player resumes pulling the data (*ON* period). This mechanism results in an *ON-OFF* pattern in video flows [198]. However, this pattern may not be synchronized among video flows sharing the same bottleneck. These studies find that the throughput estimator fails to accurately measure the network throughput, because the *ON-OFF* patterns can be fully, partially, or completely overlapped among video flows. Therefore, the quality adaptation algo-

rithm could fail to converge to a stable bitrate, and the bottleneck capacity cannot be fairly shared.

2.2.2 Internet experiment

A considerable amount of research effort is spent on evaluating the performance of real-world video streaming systems using the passive or active approach. Passive measurement studies include analyzing server logs [146, 148, 147, 246], traces obtained from a client-side program [117, 48, 53], and packet traces captured at the gateway of a network [86, 250, 54, 96] or in the ISPs [80, 192, 39, 215]. This kind of study can reveal the viewing behavior and diagnose the performance issues between clients and particular video streaming sites, such as YouTube, DailyMotion, PPTV, and PPLive.

On the other hand, in the active measurement approach, experimenters set up probing machines which act as clients to stream videos from video websites. Rao et al. [198] measure and models the streaming strategies of YouTube and Netflix on different OSes and browsers. The authors identify three different streaming strategies with different traffic properties. Another study [197] also shows similar findings. Liu et al. [156] conduct experiments on iOS devices and investigate redundant video streaming traffic when the video player streams videos from YouTube, DailyMotion, and Veoh. The authors propose a proxy-based approach, named *CStreamer*, to mitigate the redundancy.

Apart from directly downloading video through the Internet, it is also common to insert a middlebox between the probing machine and the Internet gateway to emulate different measurement scenarios. De Cicco and Mascolo [63] investigate the characteristics and performance of Akamai video streaming service. In [105], the authors characterize the bitrate selection methods adopted by Hulu, Netflix and Vudu, and find that obtaining accurate bandwidth estimates on the client side is hard. In [158], the

authors compare the Netflix service on an Xbox, a Roku, a Windows desktop, and an Android phone. They find that the bandwidth consumption at the steady state varies across devices.

Other than evaluating the streaming performance, the active probing method can be used to dissect the architecture of CDN employed by large-scale video websites [12, 13]. Wink and Zink [234] set up DASH servers in a cloud (e.g., Amazon EC2), and evaluate their performance by connecting them from different networks. A recent work [16] employs 16 dual-stacked probes deployed in home, office and academic networks to compare the connectivity and performance of YouTube over IPv4 and IPv6 network.

2.3 Measuring the QoE of HTTP streaming

The QoE of HTTP streaming can be measured using objective metrics or subjective assessments. However, objective metrics, such as Peak-Signal-to-Noise-Ratio (PSNR), is hard to reveal the subjective nature of the QoE. For example, video length, content, or even the time of day can also affect the QoE [171]. Therefore, in this section, we focus on three types of QoE measurement approaches—subjective-based, pseudo subjective based, and objective metrics [52]. The QoE is often expressed using Mean Opinion Score (MOS) [112]. The subjects rate their perceived quality using a 5-point Likert scale from 1 (“Bad”) to 5 (“Excellent”) after watching a video.

2.3.1 Subjective assessment based approach

A number of laboratory-based or crowdsourcing-based subjective studies investigate the impact of different video streaming events in HTTP streaming on the QoE. Table 2.1 summarizes the video streaming events these works have investigated and their

effect on the QoE. The last three rows in the table are our work to be presented in this thesis. Ni et al. [179] find that flickering video quality can degrade the QoE. In [97], the authors perform QoE crowdtesting on YouTube. Their results confirm our finding in [165] that the rebuffering events are one of the major factors can degrade the QoE. Hoßfeld et al. [94] further investigate the trade-off between initial buffering time and rebuffering events. Their results show that both impairments can hurt the QoE, but increasing the initial buffering time is less harmful to the QoE than having rebuffering events. Liu et al. [155] conduct QoE assessments for both artificial and real network video bitrate switching cases, and show that the fluency of the video, the start-up bitrate and bitrate switching can impact on the QoE.

Staelens et al. [220] use the Single Stimulus Continuous Quality Evaluation (SSCQE) [110] method to continuously monitor the change in perceived quality of three artificial bitrate variations which include stalling and bitrate fluctuation with different amplitudes, and find that mid-range quality level switches are mostly not perceivable by users. Hoßfeld et al. [98] assess the effect size of different influencing factors in the HAS QoE model and find that the time duration playing on the highest quality and the amplitude of video quality switches significantly influence the QoE. However, in a recent study, Egger et al. [73] show that frequent or abrupt quality switching does not significantly affect the QoE. Other than the video quality and rebuffering events, Tani and Nunome [226] analyze the relationship between seeking operation and the QoE. Context information, such as system properities and states, are also found very useful in predicting the QoE [99]. A number of comprehensive surveys about measuring the QoE of HTTP streaming can be found in [214, 52, 120].

Table 2.1: Summary of different video streaming events investigated in the literatures.

Work	Bitrate		Bitrate switching		Initial delay	Rebuffering events	Rebuffering duration
	Initial	Highest	Frequency	Amplitude			
[179]			↓				
[97]						↓	↓
[94]					↓	↓	
[155]	↑		↓			↓	
[220]			–	↓		↓	
[98]		↑	–	↓			
[73]			–	–		–	
[165]					↓	↓	↓
[170]				↓			
[169]	↑		↓				

Note: ↓, –, ↑ represents higher frequency or longer duration of that event is found to be improving, having no effect, or degrading the QoE, respectively.

2.3.2 Pseudo subjective metrics based approach

Another class of works does not assess the QoE directly. Instead, they infer the QoE using some pseudo subjective metrics, such as user engagement and video abandonment rate. Dobrian et al. [70] correlate two levels of engagement metrics with five quality metrics. The engagement metrics include the duration of the viewing session, the number of views per viewer, and the total play time of all videos watched by the viewer. On the other hand, the quality metrics are application layer events, including join time, buffering ratio, rate of buffering events, average bitrate, and rendering quality. The higher the engagement level are regarded as the better the QoE. In [139], the authors further include the abandonment rate and the viewer's return rate as the engagement metrics. The user engagement metrics in [215] also considers the skipping of video. Furthermore, the authors correlate lower layer information, including the information from TCP/IP layer and features in cellular network, with the user engagement metrics. In a recent study, Chen et al. [51] combine data packets captured in a campus WiFi network and a survey collected from the users of that WiFi network to understand the relationships among video streaming performance, user engagement

and viewer experience. They argue that user engagement can only partially reflect the viewer experience.

2.3.3 Model based approach

Because the impairments in HTTP streaming are different from UDP-based streaming, the objective metrics focusing on spatial artifacts, such as PSNR, and [57], are no longer applicable. Therefore, a number of new objective models or metrics are developed using the findings obtained from subjective assessments.

Singh et al. [216] employ a Random Neural Network (RNN) to learn the relationships among rebuffering, quantization parameters, and the QoE. The total number of rebuffering, average and maximal rebuffering duration, and the number of quality levels are considered in their model. They find that users are more sensitive to rebuffering events than lower bitrates. Casas et al. [40] reconstruct YouTube video flows from packet traces and estimate their performance in terms of the number of rebuffering events. They propose a mapping function to predict the QoE from the number of rebuffering events. Therefore, network operators can have a clearer view on the QoE only using passive monitoring techniques. Similarly, Alberti et al. [20] estimate the QoE by computing an Estimated Mean Opinion Score (eMOS) using the rebuffering frequency, average duration, and the rate of quality switching. The average and standard deviation of video bitrate and the frequency of quality switches are considered in [232]. The predicted MOS is based on a linear combination of the three metrics.

Chen et al. [44] propose time-varying subjective quality (TVSQ) of videos to infer the QoE. TVSQ is a metrics based on continuous-time records of viewer's perceived quality of short video clips. In [44], the authors propose a dynamic system model to smooth out the variations estimated by short-term quality prediction algorithm. However, their model assumes that there is no rebuffering event occurred in the playback.

Balachandran et al. [27] develop a decision tree-based predictive model from the application layer information, including the join time, buffering ratio, and ratio of buffering, to predict the user engagement level. They also identify a number of confounding factors which can affect the engagement level. Their results show that the decision trees are useful for selecting the CDN and bitrate for clients.

2.4 Improving the QoE of video streaming

Having discussed the factors influencing the QoE, we move on to the problems of mitigating the impairments and further enhancing the QoE of HTTP video streaming. We will first review the methodologies for adapting video bitrate on the client side, which is the most common approach in HAS. After that, we will highlight some related works employing middleboxes or server-side approaches to improve the streaming performance from other layers, such as the network layer. Finally, we will present some works focusing on the initial bitrate selection problem.

2.4.1 Quality adaptation algorithms

The quality adaptation algorithm plays an important role in HAS, because it is responsible for selecting the quality levels to be downloaded from the server. One of the major goals of the algorithm is to mitigate rebuffering events, which impact the QoE more than other factors. On top of that, some algorithms also consider the stability of the video bitrate, which can further improve the QoE. In addition, some algorithms can alleviate the interference caused by competitive video flows sharing the same network bottleneck [18]. In this section, we will summarize existing quality adaptation algorithms by focusing on three aspects—1) bandwidth estimation, 2) video bitrate adaptation, and 3) segment download schedule. Bandwidth estimation can measure

the network condition. By incorporating bandwidth estimates and other information, the video bitrate adaptation decides the quality level to be downloaded which can reduce the occurrence of rebuffering and the impact of video quality switching on the QoE. Segment download schedule is particularly important for accurately estimating the bandwidth when multiple video flows share the same network bottleneck. Table 2.2 lists the quality adaptation algorithms we will discuss in this section. "T", "R", and "B" in the table represent the bitrate adaptation or the download schedule is determined by the throughput information, the current/historical bitrate information, and the information from the video buffer, respectively. We focus on HTTP streaming systems, so we do not consider UDP-based video streaming systems as a recent taxonomy does [65]. Table 2.3 shows the notations used in this section for describing the algorithms.

Table 2.2: A list of quality adaptation algorithms studied in this literature.

Algorithms	Bandwidth estimation	Video bitrate adaptation	Video segment download schedule
[152]	Mean	T	B
FESTIVE [118]	Harmonic mean	R	Random, B
[230]	PID controller	B	B
[228]	EWMA	T	B
[229]	EWMA	T	B
QAAD [223]	EWMA	T, B	B
[106]	n/a	B, R	B
PANDA [149]	EWMA (Bandwidth share)	T	T, B
PANDA/CQ [145]	EWMA (Bandwidth share)	T, B	T, B
SARA [121]	Harmonic mean	T	B
[243]	Harmonic mean	T, B	B

Note: "T", "R", and "B" represent "Throughput-based", "bitRate-based", and "Buffer-based", respectively.

Bandwidth estimation

There is a considerable amount of research investigating the bandwidth estimation problem [115, 102, 23, 222]. Some of them can be applied to UDP-based streaming [25]. However, HTTP-based streaming runs on top of browsers which is not able

Table 2.3: Notation used in this section.

Notation	Explanation
n	The number of downloaded video segments
μ	Segment fetch time
B_{cur}	Current buffered video size (in video time)
B_{max}	Maximum buffer size (in video time)
B_{min}	Lower bound of desired buffered video size (in video time)
$\mathcal{L}$	A set of quality levels supported by the video, $\mathcal{L} = \{l_1, l_2, ..., l_k \| k = 0, 1, ...\}$
t_{seg}	Length of a video segment (in video second)
l_j^s	Quality level selected for j^{th} video segment, $l_j^s \in \mathcal{L}$
$\mathcal{S}_k$	The actual size of video segment of quality level k
$\mathbb{T}_j$	Actual download time of j^{th} video segment
$\mathcal{I}_j$	The inter-request time between the requests for j^{th} and $(j-1)^{\text{th}}$ video segments
$b(\cdot)$	A function that returns the bitrate of a quality level
$\mathcal{Q}(\cdot)$	Video bitrate quantization function
$\tilde{x}$	Measured TCP throughput
$\overline{x_k}$	Estimated throughput of j^{th} video segment

to have fine-grained control on probe packets' sending time or sizes. Therefore, these methods cannot be easily applied. Most of the quality adaptation algorithms measure network throughput at the application layer. The results are then used to predict the throughput of the next video segment. A straightforward approach is to use the throughput of downloading the last video segment as the estimated throughput of the next one. The network throughput can be computed by dividing the size of the previous video segment by its download time as shown in Equation (2.1). This kind of throughput measurement is similar to the flooding-based bandwidth measurement, which can be inaccurate especially when a single TCP flow is used and the video segment size is not sufficiently large. Therefore, smoothing or filtering techniques are usually necessary in order to obtain a more stable estimate.

$$\overline{x_{n+1}} = \tilde{x} = \frac{\mathcal{S}_{l_n^s}}{\mathbb{T}_n}. \tag{2.1}$$

Liu et al. [152] propose using the ratio of video segment duration to segment fetch time μ as shown in Equation (2.2). However, this metric is actually directly propor-

tional to the throughput obtained from Equation (2.1). Tian and Liu [230] employ a PID control module to accommodate the fluctuation. Therefore, the video quality can be more stable. The control module also considers the size and the trend of the video buffer in order to provide an estimate of the highest supported video bitrate.

$$\begin{aligned} \mu &= \frac{t_{seg}}{\mathbb{T}_n} \\ &= \frac{\mathcal{S}_{l_n^s}}{l_n^s \mathbb{T}_n} \\ &= \frac{\tilde{x}}{l_n^s}, \end{aligned} \tag{2.2}$$

Other than the PID controller, a number of algorithms apply exponential weighted moving average (EWMA) to mitigate the throughput fluctuation as shown in Equation (2.3). The parameter δ is a weighting value which controls the contribution of the most recent throughput measurement result to the smoothed throughput. QAAD [223] uses a constant value of δ and a fixed throughput sampling period. Comparing to sampling the throughput at completion of downloading a video segment, a fixed sampling period may obtain more throughput estimates and respond quicker to throughput changes. In [228] and [229], the value of δ is adjusted according to a logistic function, which is derived from the normalized throughput deviation. Equations (2.4) and (2.5) show the computation of the logistic function and the deviation, respectively.

$$\overline{x_{n+1}} = \begin{cases} (1-\delta)\overline{x_{n-1}} + \delta \frac{\mathcal{S}_{n,l_n^s}}{\mathbb{T}_n}, & \text{if } n > 1 \\ \frac{\mathcal{S}_{l_n^s}}{\mathbb{T}_n}, & \text{if } n = 1 \end{cases} \tag{2.3}$$

$$\delta = \frac{1}{1 + e^{-k(p-P_0)}}, \tag{2.4}$$

$$p = \frac{|\frac{\mathcal{S}_{l_n^s}}{\mathbb{T}_n} - \overline{x_{n-1}}|}{\overline{x_{n-1}}}, \tag{2.5}$$

where the values of k and P_0 are constant and set according to the characteristics of the network, and p is the deviation of throughput.

Harmonic mean is another averaging function commonly employed by quality adaptation algorithms. Equation (2.6) shows the equation for computing the harmonic mean of the last a throughput samples, where w_i is the weight to the i^{th} historical throughput. FESTIVE [118] smooths the throughput estimation by applying harmonic mean to the last 20 samples with equal weights (i.e., $a = 20$ and $w_i = 1, \forall i$). SARA [121] assigns the weights proportional to the actual video segment sizes, which can be found in the enhanced MPD file. In [243], the authors claim that their model predictive control approach does not restrict on the throughput prediction algorithm. The harmonic mean to the throughput of the past five chunks is applied in their implementation of FastMPC and RobustMPC methods (i.e., $a = 5$ and $w_i = 1, \forall i$).

$$\overline{x_{n+1}} = \frac{\sum_{i=n-a}^{n} w_i}{\sum_{i=n-a}^{n} w_i / (\frac{\mathcal{S}_{i,l_i^s}}{\mathbb{T}_i})}. \tag{2.6}$$

Some algorithms indirectly consider the throughput information rather than directly applying the smoothing functions. PANDA [149] and PANDA/CQ [145] do not directly apply smoothing to the measured throughput. Instead, they apply an EWMA smoother to the estimated bandwidth share, denoted by $\widehat{x_{n+1}}$. The goal of this approach is to filter the overestimation caused by the *OFF* period of competitive streaming flows. Equation (2.7) shows the computation of the bandwidth share.

$$\frac{\widehat{x_{n+1}} - \widehat{x_n}}{\mathcal{I}_n} = \kappa(w - \max(0, \widehat{x_n} - \frac{\mathcal{S}_{l_n^s}}{\mathbb{T}_n} + w)), \tag{2.7}$$

where κ and w are the convergence rate and the adaptive increase bitrate, respectively, to control the rate of increasing or decreasing the video bitrate. In [106], the authors advocate a buffer-based approach, which only considers the buffer occupancy infor-

mation for adapting the video bitrate.

Video bitrate adaptation

After estimating the throughput, the next step is to select a suitable quality level of the next video segment to be downloaded. A major goal is to avoid the buffer underruns and overflows (i.e., $B_{min} < B_{cur} < B_{max}$). A simple approach is to select the highest quality level with a video bitrate supported by the most recent estimated throughput [152, 228, 229, 121]. A safety margin ε is often added to mitigate the throughput overestimation as shown in Equation (2.8). However, this approach does not consider the QoE and can possibly result in an abrupt change in quality when the throughput drops suddenly, which can adversely affect the QoE.

$$l_{n+1}^{s} = \mathcal{Q}(\overline{x_n} - \varepsilon). \tag{2.8}$$

Besides the throughput, a number of works considers the previous quality level, or/and buffer status in the algorithms. For example, QAAD [223] increases the quality level additively. Moreover, the algorithm uses the estimated throughput to predict the time the buffer reaches the minimum buffer level B_{min}, and selects the highest quality level which can avoid the depletion of the buffer for at least one video segment. Tian and Liu [230] use a dynamic piece-wise constant function to control the responsiveness of bitrate adaptation. A faster growth of the buffer allows a more aggressive increase of bitrate. FESTIVE [118] employs a monotonic decreasing function to control the rate of increase of quality levels according to the current quality level. The rate of switching up the quality level is faster when the current level is low and vice versa. PANDA [149] implements a dead-zone quantizer which introduces two different margins to control the upshift and downshift of quality levels.

Some works do not directly consider the possibly highly fluctuated throughput in-

formation. A Markov decision process (MDP) based controller is designed in [116] to control the switching of the video bitrate mainly relying on buffer information. In [106], the authors solely consider the buffer states and propose a mapping function for selecting the next video segment's bitrate by the buffer occupancy. The higher video bitrate is selected when the buffer contains more video data.

Apart from only considering the quality level of next video segment, recent works employ different models to estimate the impact of multiple steps of bitrate decision on the QoE. PANDA/CQ [145] employs a dynamic programming approach to compute the *finite horizon*, which predicts the upcoming steps based on throughput and buffer information. Yin et al. [243] maximize a QoE metrics which combines the average quality, quality variations, and total rebuffering time from the beginning. A Model Predictive Control (MPC) approach utilizes the QoE metrics, throughput estimation, and buffer information to generate and predict the optimal setting for the player. Xing et al. [238] use two discrete-time finite-state Markov models which consider the state within a smooth window size.

Video segment download schedule

Not all the quality adaptation algorithms explicitly specify the download schedule after determining the quality level (e.g., [223, 228, 229]). Their download schedule of a new video segment is indeed indirectly controlled by the buffer size as illustrated in Equation (2.9). The new video segment is fetched immediately if the buffer is not full. Otherwise, the next video segment will be downloaded after playing at least one segment [243].

Kuschnig et al. [140] and DAVVI [78] propose to employ multiple parallel TCP connections to download video segments for better utilizing the network bandwidth. The HTTP requests and video segments are sent through three TCP connections. In

[152], the system considers the maximum amount of video data used when the throughput drops to the bitrate of the lowest quality level as shown in Equation (2.10). SARA [121] and Tian and Liu [230] define a threshold B_β to determine the download schedule as shown in Equation (2.11). When the buffer level is higher than B_β, a delay is inserted. These algorithms will produce a periodic *ON-OFF* pattern when the throughput is stable.

$$\mathcal{I}_n = \begin{cases} 0 & \text{if} B_{cur} < B_{max} \\ t_{seg} & \text{if} B_{cur} = B_{max}, \end{cases} \tag{2.9}$$

$$\mathcal{I}_n = \max(0, B_{cur} - B_{min} - \frac{l_n^s}{l_1} t_{seg}), \tag{2.10}$$

$$\mathcal{I}_n = \begin{cases} 0 & \text{if} B_{cur} < B_\beta \\ B_{cur} - B_\beta & \text{if} B_{cur} \leq B_\beta. \end{cases} \tag{2.11}$$

The aforementioned methods may suffer from an instability problem caused by concurrent video flows [18]. FESTIVE [118] proposes a randomized scheduler to ensure the request time is independent of the video start time, so that the throughput estimate can be converged. However, the randomized schedule may not be able to fully utilize the network throughput. PANDA [149] and PANDA/CQ [145] employ a closed-loop design to determine the segment download schedule. It considers both throughput and buffer information as shown in Equation (2.12).

$$\mathcal{I}_n = \frac{b(l_{n+1}^s) t_{seg}}{\widehat{x_{n+1}} + \beta(B_{cur} - B_{min})}. \tag{2.12}$$

2.4.2 Server-side and middleboxes approach

Although the client side approach is flexible to deploy, the video player runs at the application layer and cannot obtain information of other clients. For example, it is in-

feasible to obtain a global view of the server status or modify the default TCP behavior of video flows. Therefore, a number of server side, middlebox, caches, and in-network approaches are proposed.

Server-side shaping methods are proposed in [17, 154] for achieving a similar goal. Trickle [85] proposes to throttle the video flows at the video server. The TCP congestion window is also adjusted according to the streaming rate rather than only considering network events. Their results show that the packet loss events due to bursty traffic can be significantly reduced.

Middleboxes, including home gateway and proxies, can be used to regulate the traffic of video flows. The home gateway is used in [100] to shape the video traffic to improve the stability of multiple video flows (from different clients) sharing the same network bottleneck. A recent work [130] proposes to use a network proxy to evenly share the bandwidth for its clients. When the client requests a higher bitrate than its share, the proxy rewrites the HTTP requests for the client to avoid potential performance degradation.

Caches and proxies, especially the CDN, are commonly used to improve video streaming performance. Carbunar et al. [38] propose a network-aware caching algorithm, which considers both video popularity and the network fetch cost in replacing an object in the cache. Zhang et al. [247], on the other hand, consider the ratio of required playback rate to the actual playback rate as a QoE metrics to manage the video cache in wireless network.

The video cache may not store the complete sets of video segments. Therefore, the video streaming throughput can drop significantly when cache misses occur in fetching some video segments. The client can perceive an instable video quality. ViSIC [141] is proposed to address this problem by shaping the traffic in the cache. This prevents the client from requesting a video quality that exceeds the network throughput. iPac

[150] analyzes the video requests and pre-fetches uncached video segments from the content server which are likely to be requested in the near future. Therefore, the video streaming performance can be increased with a higher cache hit rate. Wang et al. [235] propose to utilize the idle CDN resource to perform online video transcoding based on user request predictions and their geo-location.

In the future Internet architecture, in particular Software Defined Network (SDN), can facilitate in-network approaches to improve the QoE of video streaming. OpenFlow, which is an implementation of SDN, specifies the protocol of exchanging information between the OpenFlow switches and the controllers. An OpenFlow controller can capture the information, redirect, and/or modifying the video flows through the OpenFlow switches. QFF [84] is proposed to use an OpenFlow switch to fairly share the bandwidth of multiple video streaming devices in a home network. Petrangeli et al. [189] archive a similar goal by prioritizing video flows.

OpenCache [83] is proposed to provide in-network caching between clients and video servers. The controller redirects the clients to an appropriate cache node to download video segments. Bouten et al. [32] use packet sampling measurement obtained by OpenFlow switches to estimate the remaining bandwidth. The OpenFlow switches then feedback the network information to the clients for a better selection of quality levels.

2.4.3 Initial video bitrate selection

Most of the aforementioned quality adaptation algorithms initiate the video stream to a default quality level. Without any information of the network condition to base on, users can perceive unnecessary quality switching at the video start-up phase. Hsu [101] invents a system for determining the startup bitrate in adaptive bitrate streaming. His system stores the measured throughput measured from previous video streaming in the

browser's cookie. At the next video playback, the startup bitrate is estimated using the stored historical throughput. However, the startup bitrate cannot be determined at the first visit, or after the cookie is cleared. Netflix [176] and Google [89] archive the throughput data from video streaming clients by countries, ISPs, and access methods, but these data will not be used in deciding the initial video bitrate. Furthermore, more fundamentally, the historical data may not correctly reflect the latest network condition.

A number of works consider information apart from network throughput to determine the initial bitrate. Liu et al. [153] propose a coordinated video control plane to optimize the video delivery by a global view of clients and their network. Later on, C3 [81] introduces a split control plane architecture to further improve the performance of CDN and video bitrate selection. However, the summarized global view may not be able to react to short-term performance degradation promptly for a small number of clients. Riiser et al. [205], Fardous and Kanhere [79], and GTube [92] utilize the geographic location to estimate the bandwidth for mobile users' next locations, but these methods are location-specific, and the training cost is often expensive.

Another direction is to pre-load the video segments of the same [135] or different [136] video clip using parallel TCP connections. In [135], the authors aim at improving the playback performance of interactive branched video, which assumes the users traverse to a particular time point of a video by clicking the timeline. After the jump, users may experience a long interruption and low video bitrate when downloading new video segments. Therefore, a pre-fetching mechanism is proposed in [135], which downloads the video segments of some predicted branch point in advance to reduce the start-up delay and enhance the bitrate. This approach is later extended to preload video segments of the next video clip likely to be watched in [136]. Similarly, the preloading video segments are downloaded during the OFF period of the current HAS flow through a parallel TCP connection. Therefore, if the prediction is correct, the

initial video bitrate of the next video can possibly be improved. However, this method cannot improve the initial video bitrate of the first video.

2.5 User-behavior based QoE assessment

User-behavior based QoE assessment employs the events or activities that users interact with the video streams to infer the QoE. Most of the existing works analyze the user/viewer behavior including viewing time and video seeking by summarizing their overall distributions. Dobrian et al. [70] first propose to infer the QoE from user engagement level, which includes join time, viewing time, and application events collected on the client-side. The user behavior Tencent's video service [48, 49] is analyzed using server logs. A similar study [53] is performed on another VoD platform, PPLive. Besides the engagement level, video clip switching, seeking behavior, and video popularity are considered.

From the ISPs and network operators point of view, the server log is not readily available to them. Shafiq et al. [215] analyze the user engagement level from an ISP perspective. A number of network factors are used to classify the viewing sessions into four classes (completed vs. abandoned and non-skipped vs. skipped) and the video download completion rate. Another work [51] predicts the user engagement level by deducing application events from packet traces captured from campus network. Furthermore, a survey is conducted with the network users to understand the general perceived quality.

Besides analyzing natural user behavior, OneClick framework [47] requires users to explicitly click a dedicated button whenever they perceive quality degradations. The frequency of clicks is used to train a Poisson regression based model to find out the factors at the network layer responsible for the QoE degradation.

2.6 Detecting low-quality workers in QoE crowdtesting

To improve the reliability of QoE crowdtesting, a considerable amount of works study on screening out low-quality workers from the crowd. Some previous works employ worker behavior or application layer metrics to identify low-quality workers. Rzeszotarski et al. [206] propose to use several user behaviors to infer the quality of workers. However, they aggregated mouse cursor movements into events without storing the coordinates. Therefore, their analysis on mouse cursor movement was coarse. In their follow-up paper [207], the authors focus on visualizing the user behavior, which can help the experimenters to manually screen out potentially low-quality workers. Costagliola et al. [60] capture the student behavior in an e-learning system and detected cheating by analyzing the sequence of answering questions. Hirth et al. [93] analyze some application layer metrics, such as consideration time and completion time, and then flag the outliers as low-quality workers. Gardlo et al. [82] employ a credit based scheme to estimate the reliability of workers. The worker's reliability points are deducted when some suspicious behaviors are identified.

Other than analyzing worker behavior, a number of studies focus on processing the data after the workers complete the tasks. Buchholz and Latorre [36] propose comparing the data with the gold standard data collected in laboratory experiments. CrowdMOS [202, 201] computes the average and the deviation of the results submitted by workers for each scenario. The workers providing results significantly different from the average value will be considered as low-quality ones. Instead of directly giving the ratings, Wu et al. [236] only require workers to give a binary decision on which case is better. The rankings are then compared among workers to find the outliers. However, these methods often cannot be applied to surveys or assessments that have no absolute answers or rankings. Joglekar et al. [119] propose techniques to generate confidence intervals for worker error rate estimates to determine the worker's

quality. However, their method only supports binary questions.

Instead of detecting cheaters after the completion of the whole crowdtesting campaign, some existing works proposed a better job design to evade cheaters. Using CAPTCHAs or asking questions with known answers are effective in foiling software bots which automatically complete the task [72, 74]. Redesigning the task as a game [75] can attract entertainment-seeking workers, who are shown to be more reliable than money-driven workers. In [195], a hybrid approach is adopted. Three levels of the filtering scheme are used based on both application layer information and control questions to filter unreliable workers. Another method is to adopt a two-phase approach to screen out the pseudo-reliable crowd before conducting the actual assessment [218]. A qualification task is first deployed. The workers who passed the task are believed to be more reliable. They are then invited to work on the actual assessments. However, these two methods can be easily noticed by careful cheaters, and they also inevitably increase the length of the assessment period.

Chapter 3

Understanding the Influencing Factors to the QoE

HTTP streaming delivers video streaming data over TCP. The reliable feature of TCP distinguishes HTTP streaming from traditional UDP-based streaming. Because the video codec is free from handling packet losses, the picture quality is not degraded due to missing/damaged video frames. In this chapter, we present our study on the influencing factors in the QoE of HTTP streaming and HAS. We first study the correlation between network quality of service (QoS) and the QoE in HTTP streaming, which is one of the first systematic studies on this problem. Network QoS is usually represented by network path quality metrics, such as round-trip time, packet loss rate, and capacity. Poor network quality can reduce the TCP throughput. Meanwhile, the application layer performance can be affected by the throughput. For example, when the throughput is lower than the playback rate, the video playback will pause and wait for new video data. For HAS, even though the video playback may not pause, the video quality can drop abruptly. These events can be regarded as application QoS. In HTTP streaming, the application QoS layer focuses on the *temporal structure*. Rebuffering events can be mitigated by adjusting the video bitrate in HAS, hence the application

QoS in HAS also includes the transition of video quality levels. Finally, users perceive the application QoS as a part of their QoE.

Obviously, common users can only directly perceive the performance from the application, instead of the network. Therefore, it is challenging to directly correlate between the network QoS and the QoE. In the first part of the chapter (Section 3.1–3.4), we introduce a two-step approach to study this problem. The first step is to correlate between the network QoS and the application QoS layer, and the second step is to investigate the correlation between the application QoS and the QoE. Therefore, we form a conceptual relationship between the network QoS and the QoE into a protocol stack [227, 237] as shown in Figure 3.1.

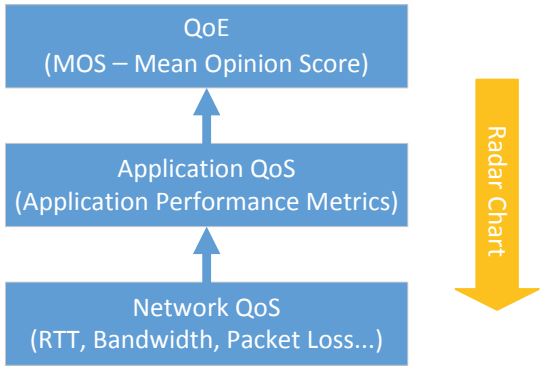


Figure 3.1: Three levels of QoS considered in this thesis.

To characterize the relationship between the network QoS and application QoS, previous works [128, 233, 180] perform analytical studies to model the video streaming performance using TCP. An algorithm is proposed to estimate the receiver buffer requirement based on the model in [128]. In section 3.1, we adopt both analytical and empirical approaches to study the correlation between the network QoS and application QoS. In particular, we propose a set of *application performance metrics* (APM) for the study: (1) Initial buffering time, (2) mean duration of a rebuffering event, and (3) rebuffering frequency. On the other hand, the network QoS can be measured based on active measurement (e.g., OneProbe [157] and YouTube Video Speed History [8])

or passive measurement (e.g., [58, 41]).

We investigate the correlation between the application QoS and the QoE of HTTP streaming by conducting subjective assessments. We have implemented a customized Flash video player, which emulates a pre-defined set of APM. The subjects can rate their QoE using the *Mean Opinion Score* (MOS) of 1 ("Bad") to 5 ("Excellent") [112]. A regression based model is then employed to acquire the relations between the application QoS and the QoE. We find that the rebuffering events can significantly impact the QoE. This finding is later confirmed and employed by a number of studies (e.g., [97, 70, 85, 94, 40]). Furthermore, a radar chart [43] is employed to visualize the relationship between the network QoS and the QoE. We also investigate the impact of network path asymmetry on the performance of HTTP streaming.

In the second part of this chapter (Section 3.5), we focus on investigating the impact of quality adaptation on the QoE in HAS. The chance of triggering rebuffering events can be reduced in HAS, because the quality adaptation algorithm can switch to a lower quality level when the network throughput decreases. On the other hand, the quality level can automatically switch up in high network throughput environment. For the same network conditions, there are multiple schemes to transit the video quality quality which can be perceived differently by users. We conduct a laboratory-based and a crowdsourcing-based subjective assessments to investigate the QoE of different video quality transition schemes.

3.1 Network QoS and application QoS

In this section, we investigate the relationship between the network QoS and application QoS. Network QoS is the network path performance between a server and a client, including the round-trip time (RTT), packet loss rate, and network bandwidth. Application QoS, on the other hand, reflects the performance from an application point of

view. In the ensuing discussion, we propose three APMs to quantify the application QoS for HTTP video streaming. We then correlate both QoS using analytical modeling and empirical evaluation.

3.1.1 Application performance metrics

We propose three APMs to quantify the application QoS for HTTP video streaming, and these metrics represent the temporal structure of a video playback, regardless of the video content.

1. *Initial buffering time* (denoted by T_{init}): This metric measures the period between the starting time of loading a video and the starting time of playing it.
2. *Mean rebuffering duration* (denoted by T_{rebuf}): This metric measures the average duration of a rebuffering event.
3. *Rebuffering frequency* (denoted by f_{rebuf}): When the amount of buffered video data decreases to a low value, the playback will pause, and the player will enter into a rebuffering state. This metric measures how frequent the rebuffering events occur.

Figure 3.2 plots the time series of the video playhead time (i.e., the current position of the video) and the amount of video buffered by *FlashTrack*, our implementation of a customized Flash video player which will be presented in section 3.1.4. The solid line refers to the video playhead time, and the dotted line to the amount of buffered video. The circles on the dotted line correspond to the empty-buffer events which occur whenever the amount of buffered video falls to a low value. The video playback pauses until the buffer is refilled. Therefore, the video playhead time stops increasing for a period of rebuffering after the onset of an empty-buffer event.

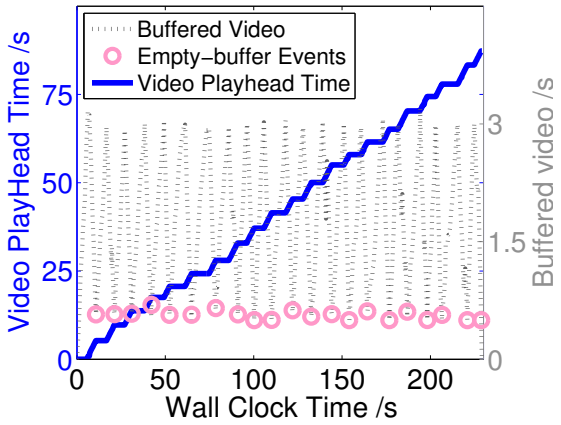


Figure 3.2: Time series of the video playhead time and the amount of video data buffered at the player.

3.1.2 Modeling the APMs

We construct a simple approximate model to correlate network QoS with the three APMs in HTTP streaming. To simplify the model, we make the following assumptions:

1. The network bandwidth, RTT, and packet loss rate are constant during the video download.
2. The client does not interact with the video during the playback, such as pausing and forward/backward seeking.
3. The average bitrate of cross traffic between the server and the client is constant.
4. The fluctuation of video bitrate is not large.
5. The video buffer must be filled up before exiting the initial buffering and re-buffering states, and its size is smaller than the video's length.

Initial buffering time and mean rebuffering duration

The estimates (in seconds) of the initial buffering time and mean rebuffering duration can be computed by

$$\widehat{T_{init}} = \frac{B_{full} \times \lambda}{\beta}, \tag{3.1}$$

$$\widehat{T_{rebuf}} = \begin{cases} 0, & \text{if } \beta \geq \lambda, \\ \left(B_{full} - B_{empty}\right) \lambda / \beta, & \text{otherwise,} \end{cases} \tag{3.2}$$

where B_{full} is the size (in seconds of video) of the video buffer, B_{empty} ($< B_{full}$) is the threshold below which rebuffering event occurs, λ is the video's bitrate (in bits/s), and β is the average TCP goodput (in bits/s) for the video streaming. $\widehat{T_{rebuf}} = 0$ for $\beta \geq \lambda$ because the rebuffering event occurs only when the average TCP goodput is less than the video's bitrate.

To estimate the average TCP throughput, we employ the throughput model for a TCP Reno flow [184], given by

$$s(p) = \frac{1}{R\sqrt{\frac{2bp}{3}} + T_0 \min\left(1, 3\sqrt{\frac{3bp}{8}}\right) p\left(1 + 32p^2\right)}, \tag{3.3}$$

where $s(p)$ is the packet sending rate per RTT, p is the packet loss rate, b is the number of packets that are acknowledged by an ACK, R is the RTT, and T_0 is the retransmission timeout. As a result, the estimated average TCP goodput for the given RTT and loss rate is $\beta = s(p) \times M \times 8/R$, where M is the size of the data packets sent from the server.

While the throughput model can estimate the TCP packet sending rate, the TCP throughput is also affected by the available bandwidth which affects both the packet loss rate and RTT experienced by the TCP flow. In particular, when the packet sending

rate is greater than the available bandwidth, the network path will become congested, thus increasing the queueing delay of TCP packets or even discarding some packets due to buffer overflow. To compute the throughput, we resort to OneProbe [157] to measure the RTT and loss rate for the network path with a particular bandwidth configuration emulated in a testbed.

Rebuffering frequency

Given a video's length of l (in seconds), the rebuffering frequency estimate is given by

$$\widehat{f_{rebuf}} = \begin{cases} 0, & \text{if } \beta \geq \lambda, \\ n_{rebuf}/l, & \text{otherwise,} \end{cases} \tag{3.4}$$

where

$$n_{rebuf} = \left\lceil \frac{l'}{b_{rebuf}} \right\rceil, \tag{3.5}$$

$$l' = l - B_{full} / \left(1 - \frac{\beta}{\lambda}\right), \tag{3.6}$$

$$b_{rebuf} = \left(B_{full} - B_{empty}\right) / \left(1 - \frac{\beta}{\lambda}\right). \tag{3.7}$$

When the average TCP throughput is less than the video's bitrate, we will encounter n_{rebuf} rebuffering events during the video playback given by Equation (3.5), where l' is the remaining length of the video (in seconds) upon the onset of the first empty-buffer event, and b_{rebuf} is the length of the played video (in seconds) before the next empty-buffer event. When $\beta \ll \lambda$ (i.e., $\beta/\lambda \approx 0$), the maximum rebuffering frequency is given by

$$\max(f_{rebuf}) = \frac{1}{l} \left\lceil \frac{l - B_{full}}{B_{full} - B_{empty}} \right\rceil. \tag{3.8}$$

From Equation (3.8), the maximum rebuffering frequency only depends on the video length and buffer size, and is independent of the TCP throughput.

3.1.3 Measuring the APMs from clients

We have implemented a customized Flash video player, named FlashTrack, to compute the actual APMs based on the status of video playbacks obtained from the client. FlashTrack uses the Flash `Netstream` class to record the buffer status and current playhead time every 0.25 seconds, and special events, such as empty buffer. We have used the property `BufferTime` in the `Netstream` class to adjust the size of the player buffer. The buffer must be filled up before starting the playback or after the rebuffering event. In our experiments, we set `BufferTime` to 3 seconds, because we have observed that it is the value possibly used by YouTube.

3.1.4 Testbed experiments

Figure 3.3 shows the testbed setup for evaluating our model for the correlation between network QoS and application QoS. A web server is installed with Ubuntu 10.04 (kernel 2.6.32-22) and Apache 2.2.14 to host video clips for a client to download and play using FlashTrack. The client runs Ubuntu 9.04, Firefox 3.6.8, and Flash Player 10.1. The TCP congestion avoidance algorithm for both machines are configured as "Reno." A Click router [131] is installed between the server and the client to emulate different network bandwidths, packet loss rates and RTTs.

Table 3.1 lists the network QoS parameters emulated by the Click router. The bandwidth was chosen between 1 Mbits/s and 15 Mbits/s to emulate the bandwidth of common home users, while 100 Mbits/s was chosen to serve as a control. The choices for RTT represent the local, inter-continent, and transoceanic paths. We also varied the round-trip packet loss rate from 0% to 8% to investigate the impact of packet loss.

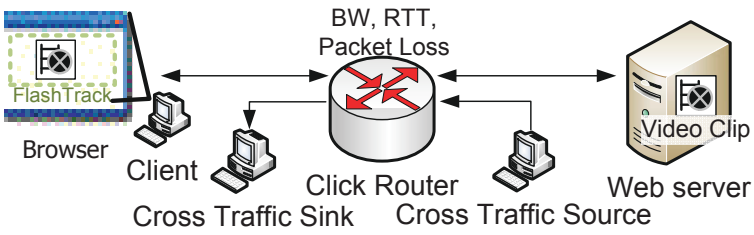


Figure 3.3: A testbed setup for evaluating the correlation between network QoS and application QoS.

Moreover, we introduced background cross traffic to the forward path of the web server using D-ITG [26]. The cross traffic was a TCP flow with packet size drawn from Pareto distribution of shape 1.2 and exponential inter-departure time with mean 500 ms, and had the average bitrate of around 100 kbits/s.

Table 3.1: The network QoS parameters emulated by the Click router.

Network QoS	Parameters
Network bandwidth (Mbits/s)	1, 5, 10, 15, 100
RTT (ms)	0, 25, 50, 75, 100
Round-trip packet loss rate (%)	0, 2, 4, 6, 8

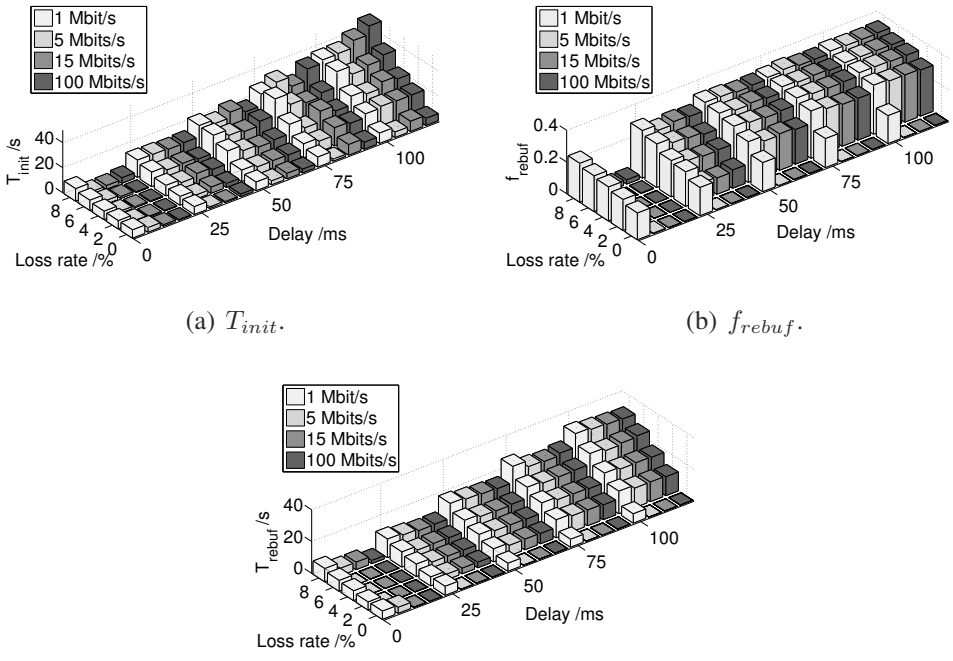
For every set of parameters, we ran FlashTrack from the client to download a video clip from the web server for three trials. The browser's cache folder was first cleared, and the private mode of the Firefox was used to ensure that the video will not be saved to the local cache after quitting the browser. The video clip was extracted from the movie trailer of "The Twilight Saga: New Moon" which belongs to the type of "Movies, movies tailers" stated in the VQEG test plan. Table 3.2 summarizes the specification of the video clip.

Figure 3.4 shows the histograms of the actual APMs measured by FlashTrack under different network QoS parameters specified in Table 3.1. The results are the averages from three independent trials. The bars with different grey levels represent different emulated bandwidths. The x-axis is the delay, whereas the y-axis is the packet loss rate. As shown in Figure 3.4, all three APMs increase with the packet loss rate and delay,

Table 3.2: A specification of the video used in the testbed experiments and QoE assessment.

Items	Parameters
Video length	87 seconds
Video format	H.264
Audio format	ACC
Resolution	864×480
File size	10.6 MBytes
Frame rate	23.97 fps
File format	FLV

but decrease with the network bandwidth. As shown in Figures 3.4(a) and 3.4(c), the distributions of T_{init} and T_{rebuf} exhibit similar patterns. Moreover, Figure 3.4(b) shows that f_{rebuf} is significantly reduced by the network path with high bandwidth and low packet loss rate. f_{rebuf} reaches near the maximum value, which is obtained from Equation (3.8), when the delay is larger than 50 ms with loss rate greater than 4%.



(a) T_{init}.

(b) f_{rebuf}.

(c) T_{rebuf}.

Figure 3.4: The three APMs under different network path quality.

We now compare the three APMs estimated by our model with the actual APMs

obtained by FlashTrack. To obtain the (round-trip) packet loss rate and RTT for Equation (3.3), we ran OneProbe [157] from the client to measure the network path between the web server and itself. OneProbe was executed three seconds after launching FlashTrack with a periodic probe rate of 2 Hz for 60 seconds. Based on OneProbe measurement, we computed the median RTT and average packet loss rate which are the parameters for estimating the TCP goodput.

Figure 3.5 shows the cumulative distribution function (CDF) of relative errors between the APM estimates and actual APMs. We compute the relative errors by $(\hat{x} - x)/x$, where $\hat{x}$ and x are the APM estimates and the actual APMs, respectively. As shown, more than 90% of the rebuffering frequency estimates have errors less than 50%, while over 75% of rebuffering duration estimates and over 60% of initial buffering time estimates have errors less than 50%. The larger error in the initial buffering time estimates is probably due to a small congestion window at the beginning of the connection which limits the packet sending rate. However, the TCP throughput model assumes that the flow has already been in the steady state, thus overestimating the TCP goodput.

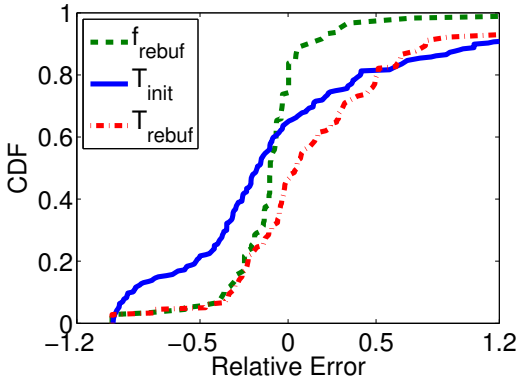


Figure 3.5: The CDF of relative errors between the APM estimates and actual APMs.

We have further investigated the rebuffering frequency using our analytical model. As will be seen later, the rebuffering frequency is the major factor affecting the QoE.

Figure 3.6 shows the rebuffering frequency for different buffer sizes and goodput-to-bitrate ratio (β/λ) with $l = 87$ seconds (same as the length of the video clip for the previous experiment). As shown, the rebuffering frequency decreases with β/λ and B_{full}, because a small buffer is used up quickly especially for a small TCP goodput (or large bitrate). When B_{full} is greater than 10 seconds, the rebuffering frequency stays at a low level (even if β/λ is small), because the maximum rebuffering event is bounded by the video length.

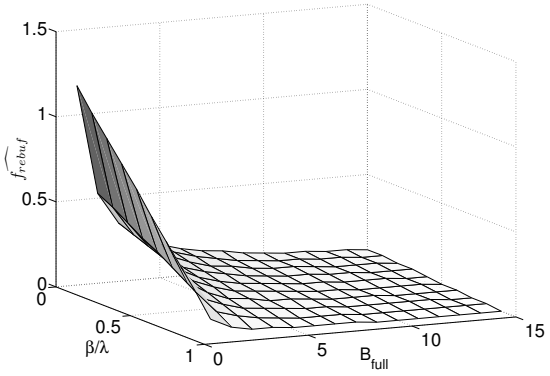


Figure 3.6: The rebuffering frequency estimates for a 87-second video clip under different buffer sizes and goodput-to-bitrate ratios.

3.2 The QoE measurement

We have performed a subjective assessment to measure QoE of Flash video perceived by users and to quantify how the QoE is influenced by the application QoS (i.e., the APMs). To this end, we have implemented a platform using Flash to emulate video playback under various levels of APMs as shown in Table 3.3. We divide each APM into three levels (low, medium, and high) which are based on the 25th, 50th, and 75th percentiles of the actual APMs (obtained by FlashTrack) reported in section 3.1.4.

Instead of delivering the video in real time, we simulate the rebuffering events by pausing and resuming the video, during which a message "buffering ..." and the cur-

Table 3.3: Three levels of application performance based on the APMs.

Level	APMs		
	T_{init}	f_{rebuf}	T_{rebuf}
Low	$0-1$ seconds	$0-0.02$	$0-5$ seconds
Medium	$1-5$ seconds	$0.02-0.15$	$5-10$ seconds
High	>5 seconds	>0.15	>10 seconds

rent buffering progress are shown on the interface. The advantage of this approach (over generating video playback in real time) is to minimize the variations (e.g., network conditions) among different subjects during the QoE measurement. To minimize the subjective bias, the player's interface, similar to common video sharing web sites, includes a progress bar showing the video playhead time, buffered video length, and video's length.

Each subject participating in the subjective assessment was required to fill in their basic personal information (e.g., gender and age) and watched the same video clip specified in Table 3.2 for 30 rounds, which include all the possible combinations of APM levels ($3^3 = 27$) and three replications to validate the reliability of the subjects' scoring. The replications were based on the APM levels randomly selected from all the 27 possibilities. After each round, the subject was asked to give a score immediately, and the whole experiment did not last more than an hour to avoid burdening the subject. Due to this time limitation, the duration of each round was limited to 120 seconds. Therefore, the subjects may not watch the video completely in every round. Finally, the sequence of video playback was randomly shuffled by a pseudo random function (`Math.random` in Flash) to mitigate the possible ordering bias resulted from the watching sequence.

After excluding three outliers who produced unreliable scores, we have successfully examined ten subjects: seven of them are male and three of them are female. Their ages ranged between 23 and 35. All of them were non-experts in evaluating

video quality. For each combination of the APM levels, we use the scores obtained from the ten subjects to compute a MOS and therefore obtain 27 MOSes to represent the QoE of the Flash video. An ANOVA analysis reveals that the rebuffering frequency is the only main factor influencing the MOS. Users are generally annoyed by the video pausing due to the rebuffering events. Moreover, there is no interaction of variables. As a result, a higher rebuffering frequency will generally lower the user-perceived quality. The effects of the initial buffering and mean rebuffering duration, on the other hand, are not significant, because users are generally willing to tolerate a longer start-up delay for a better video-watching experience.

We have performed a regression analysis to acquire a relationship between QoE and application QoS. As shown in Equation (3.9), the coefficients of the three APMs are all negative, thus a higher level of APMs giving a lower MOS.

$$\text{MOS} = 4.23 - 0.0672L_{ti} - 0.742L_{fr} - 0.106L_{tr}, \tag{3.9}$$

where L_{ti}, L_{fr} and L_{tr} are the respective levels of T_{init}, f_{rebuf}, and T_{rebuf}. We use 1, 2, and 3 to represent the "low", "medium", and "high" levels, respectively.

To minimize the variability caused by the video content, only one video clip is used in the subjective experiment. We performed pilot studies on four other video clips of different content—-sports game, news, TV comedy show and music video. Figure 3.7 plots the MOS against the three levels of rebuffering frequency. The result shows that the level of rebuffering frequency is negatively correlated with the MOS, which is consistent with our previous findings. Quantifying the correlation of various video types will be our future work.

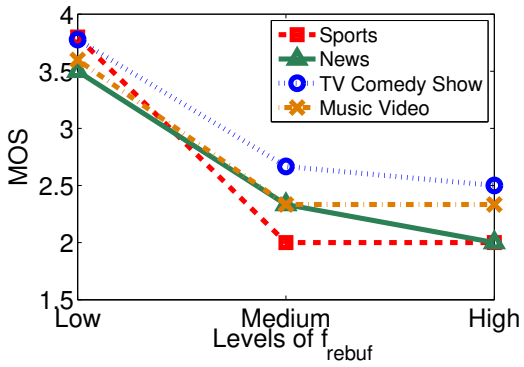


Figure 3.7: MOS against levels of f_{rebuf} for four types of video clips.

3.3 Correlating QoE with network QoS

We now describe our methodology for correlating QoE of HTTP video streaming with network QoS. Specifically, we first estimate the three APMs for a given network QoS (measured by OneProbe) using our model presented in section 3.1.2. With the three APM estimates, we then look up their levels according to Table 3.3 and finally obtain the corresponding MOSes to represent the user-perceived performance under the network path quality. Moreover, we use the radar chart proposed in [43] to inspect the correlation between the QoE and network QoS. For the network QoS parameters, the bandwidth varies from 1 Mbit/s to unlimited, delay from 0 ms to 100 ms, and packet loss rate from 0% to 8%.

3.3.1 Visualizing the correlation

Figure 3.8 shows the radar chart that maps network QoS to QoE. The MOS is divided into three levels – greater than 1, 2 and 3, which are represented using different color levels. The network path metrics with the same MOS level are bounded into areas. Sectors AB, BC, and CA fix one of the network path metrics to the "best" value – unlimited bandwidth, zero packet delay and zero packet loss rate, respectively. The other two metrics, on the other hand, vary within the sector. The three axes A, B, and

C in the chart extended from the center of the circle vary one of the network metrics from the best to the worst. Moreover, the values of axes A, B and C belong to sectors dd', ee', and ff' respectively. For example, in sector Ad', while the bandwidth is unlimited, the packet loss rate increases from 0% to 8% in clockwise direction and the packet delay increases from the center to the edge of the chart. Therefore, we could observe how the two network metrics interact with each other.

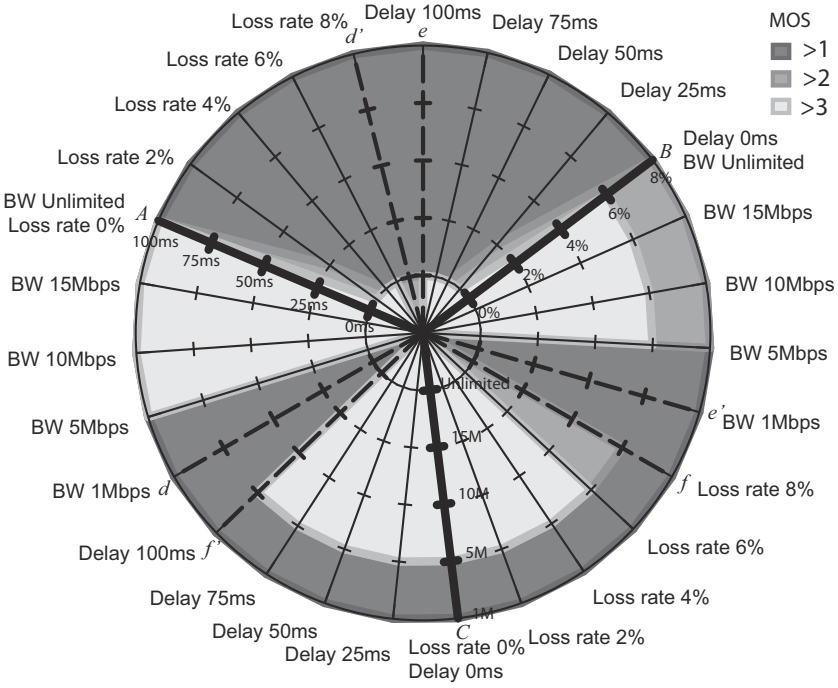


Figure 3.8: The radar chart mapping network QoS with QoE.

As shown in Figure 3.8, both packet loss rate and packet delay are the dominating factors affecting the QoE. Sector AB of the chart, which varies the packet loss rate and packet delay, is mainly a dark color region (i.e., a low MOS). This effect is also reflected in the large light color regions shown in sectors CA (with zero packet loss rate) and BC (with zero packet delay). However, a very small RTT (packet loss rate) could partially compensate the low QoE due to the packet loss (high packet delay). For example, a small region of semi-light color region in sector Ad' could still be seen even for high packet loss rate.

3.3.2 Applications

We demonstrate how the quality of HTTP video streaming can be predicted based on network QoS obtained from our recent Internet measurement study. We have conducted OneProbe measurement from a probing machine located at a local university in Hong Kong to the Lenovo web server in China between 26 August 2010 to 7 September 2010 (UTC). Since OneProbe uses HTTP requests as probe packets to elicit HTTP data response packets from remote web servers, we expect that OneProbe can observe similar network path quality experienced by the HTTP video streaming traffic. During the measurement period, the measuring node performs one-minute OneProbe measurement every ten minutes.

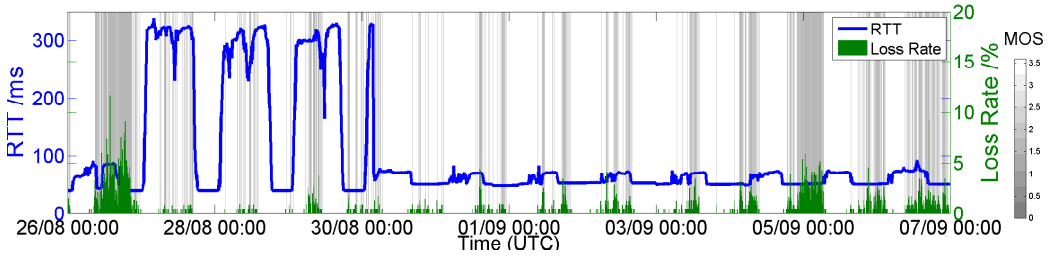


Figure 3.9: Time series of the HTTP video streaming's MOS, and the RTT and packet loss rate of the network path.

Figure 3.9 shows the time series of the median RTT and average round-trip packet loss rate obtained from the two-week network measurement. The solid (blue) line is the RTT, and the green lines close to the x-axis is the average round-trip packet loss rate. The grey-scale spectrum at the background shows the MOS estimated by our model. The lighter color in the spectrum represents a higher MOS score (i.e., better QoE) for users in watching videos hosted by the web server. We observe diurnal RTT and loss rate patterns throughout the measurement period. Moreover, the MOS is dominated by the packet loss rate, because the dark grey areas coincide mainly with higher packet loss rate, particularly on 26 August and 5 September when the packet loss rate reached around 5%. On the other hand, only sparse grey lines appear in the RTT inflation

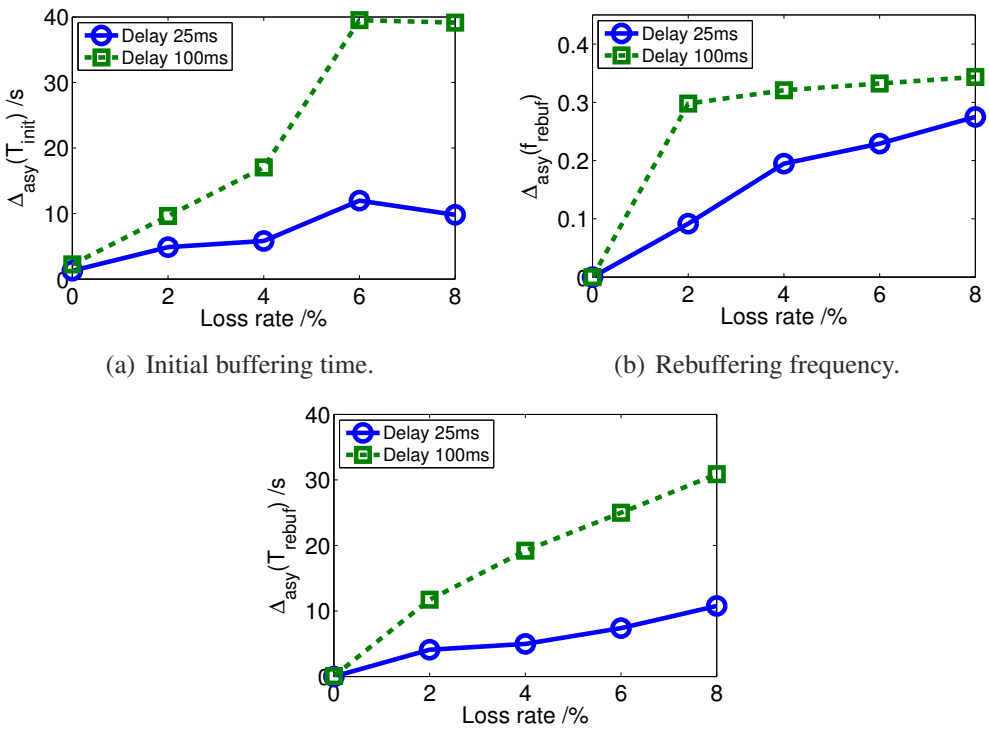
periods, thus implying that the users could still perceive an acceptable level of video quality.

3.4 Impact of network path asymmetry

In the previous sections, we consider the round-trip packet loss rate for evaluating the application QoS. However, network path asymmetry is a common phenomenon in the Internet, and it introduces different impacts on the TCP performance [28]. We therefore investigate the effect of packet loss asymmetry on the performance of HTTP video streaming by evaluating the APMs measured by FlashTrack under our testbed. To this end, we configured the testbed with network bandwidth of 5 Mbits/s, RTTs of $\{25, 100\}$ ms, and packet loss rate between 0% and 8% on a unidirectional path (forward or reverse path), while keeping the loss rate in the opposite direction to zero. Let $A(f, r)$ be the measured APM under forward-path and reverse-path packet loss rates of f and r, respectively. We compute the APM difference $\Delta_{asy}(A(p)) = A(p, 0) - A(0, p)$ for a loss rate p.

Figures 3.10(a)-3.10(c) show the values of $\Delta_{asy}(T_{init})$, $\Delta_{asy}(f_{rebuf})$, and $\Delta_{asy}(T_{rebuf})$ for different packet loss rates. The solid line and the dashed line show the results for RTTs of 25 ms and 100 ms, respectively. As shown, the APM differences are all positive, meaning that the forward-path packet loss introduces a more significant impact on the application QoS. When packet loss occurs in the forward path, the corresponding video data has to be retransmitted from the server, thus reducing the TCP goodput. However, packet loss in the reverse path affects mainly the ACK packets from the client, and a lost ACK packet could be compensated by a succeeding ACK packet with a higher acknowledgement number. Furthermore, the APM difference is more significant under the higher RTT as a result of slower packet retransmissions.

We also consider the effect of different TCP variants on the application QoS. While



(a) Initial buffering time.

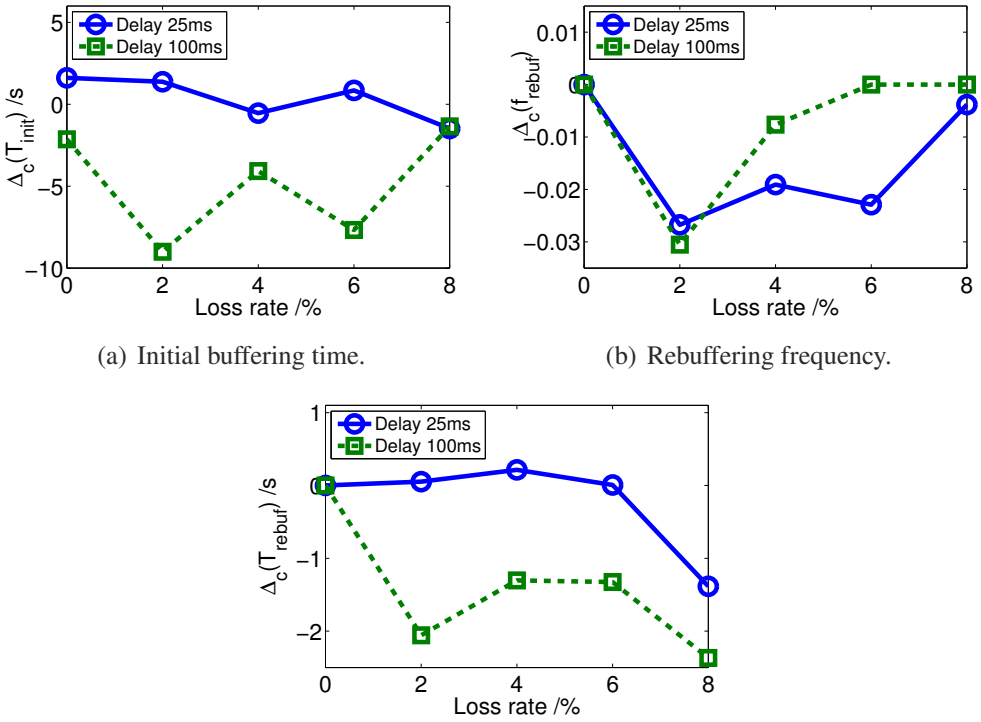
(b) Rebuffering frequency.

(c) Mean duration of a rebuffering event.

Figure 3.10: The APM differences under different asymmetric loss rates, bandwidth of 5 Mbits/s, and RTTs of $\{25, 100\}$ ms.

TCP Reno considered in the previous sections is supported by most operating systems, TCP Cubic [90] and Compound TCP [225] are the default congestion avoidance algorithms used by Linux and Microsoft Windows, respectively. These congestion avoidance algorithms could improve the fairness and throughput by better estimates of the TCP congestion window. Previous work [10] shows that TCP Cubic outperforms Compound TCP and New Reno in terms of goodput. To investigate the effect of TCP variants, we carried out another set of experiments under the same testbed except that TCP Cubic is used for both the client and server. We compare the performance by computing the APM difference $\Delta_c(A) = A_c - A_r$, where A_c and A_r are the respective APMs measured by FlashTrack under TCP Cubic and TCP Reno.

Figures 3.11(a)-3.11(c) show the APM differences of various packet loss rates and RTTs under a constant bandwidth of 5 Mbits/s. We also repeated the experiments with



(a) Initial buffering time.

(b) Rebuffering frequency.

(c) Mean duration of a rebuffering event.

Figure 3.11: The APM differences for two TCP variants under different round-trip loss rates, bandwidth of 5 Mbits/s, and RTTs of $\{25, 100\}$ ms.

RTTs of $\{25, 100\}$ ms. As shown, the advantage of using TCP Cubic is not significant. For an RTT of 25 ms, $\Delta_c\,(T_{init})$ shows approximately the same performance, except for some performance gain when the loss rate increases to 8%. On the other hand, $\Delta_c\,(f_{rebuf})$ slightly decreases at a low loss rate. The performance of TCP Cubic is better when the delay increases to 100 ms, but only less than 10 seconds of reduction in $\Delta_c\,(T_{init})$ and $\Delta_c\,(T_{rebuf})$. Moreover, $\Delta_c\,(f_{rebuf})$ decreases when the loss rate is 2%, implying a reduction of the rebuffering events. The TCP Cubic could maintain a larger congestion window in the presence of occasional loss events. Therefore, TCP Cubic receives a performance gain at loss rate less than 6%. However, when the loss is too heavy, none of the TCP variants can have a clear performance advantage.

3.5 Measuring the influence of bitrate adaptation to the QoE

In this section, we consider two problems related to the transition of quality levels. The first problem is on the best way of switching down the video quality, when the network throughput dramatically reduced. Previous studies [61, 191, 249] have shown that users tend to heavily criticize quality degradations. The second problem is on the selection of initial quality level. As the evolution of the broadband network, the user expectation increases. Users may not satisfy with a slow ramp up time to the High Definition (HD) quality.

To study the impact of these two problems on the QoE, we carefully design two sets of subjective assessments. The assessments are conducted using both laboratory-based and crowdsourcing-based approaches. To the first problem, we find that introducing an intermediate quality level can mitigate the impact of quality degradation on the QoE. In the second study, our results show that starting at a high video quality can provide the best QoE. Otherwise, even the video quality switch up to the same level, the QoE can be significantly degraded.

3.5.1 Evaluating the QoE impact on video quality down-switching

We consider a scenario that the network throughput drops from supporting the highest quality level to the lowest one. As quality degradations are usually noticeable and cause larger impact to the QoE [61], we carry out subjective assessments on their perception of quality adaptation which is possible under the same network conditions. To ensure the consistency among subjects, we pre-define a list of rule sets to profile the quality level changes. The subjects rate their QoE on different rule sets in terms of MOS.

Experiment setup

We have implemented an experimental video delivery platform to carry out subjective assessments. The video server is installed with Debian 6.0, Apache web server 2.2.16, and Adobe's HTTP Origin Module [14] in order to support Adobe Dynamic Streaming. In addition, the video player is developed using the Open Source Media Framework (OSMF) [5] and Strobe Media Playback (SMP) [11]. We replace the original quality adaptation algorithm in the OSMF, so that the quality level switching follows a pre-defined scenario.

We define each scenario with a rule set, $\mathbb{R}$, expressed by Equation (3.10). Each rule set contains at least one rule tuple represented by $< l_k, \mathcal{S}_{j,l_k} >$, where l_k is the k^{th} quality level and $\mathcal{S}_{l_k}$ is the number of bytes to be downloaded at that quality level. One rule tuple is completed when the number of bytes downloaded at the quality level is greater than or equal to the defined value. The player proceeds to the next rule tuple until all the rule tuples have been used. At this point, the quality level stays at the last level. Therefore, we can emulate different kinds of quality changes by defining these rule sets.

$$\mathbb{R} = \quad \{< l_0, \mathcal{S}_{l_0} >, < l_1, \mathcal{S}_{l_1} >, ..., < l_j, \mathcal{S}_{l_j} >\} \quad , j = 0, 1, 2, ... \tag{3.10}$$

Generation of rule sets

In our experiments, we emulate a sudden drop of throughput from 4Mbps to 400Kbps after playing the first three fragments. Therefore, the highest possible quality drop is from l_4 to l_0. We also emulate three buffer sizes—1 segment, 3 segments, and 8 segments. These three buffer sizes are typical values employed by DASH video players. We also compute the maximum number of segments supported at the intermediate quality levels n_{frag} (5.2). Table 3.4 lists the values of n_{frag} for our experiments. For

the 1-segment buffer, only quality levels below l_2 allow downloading at least one complete segment during the buffer compensated period.

Three different initial quality levels, l_4, l_2, and l_0, are employed in the experiments. As the subjects have not watched the video before, varying the initial quality levels can give an insight as to how the subjects perceive the picture quality in a no reference context.

Table 3.5 lists the rule sets we employed in our experiments. Except for $\mathbb{R}_0$, the subjects watch the first three segments of the video at the original quality level defined in the first tuple of the rule sets. Then, the video player plays all the segments in the buffer to emulate the consumption of the buffer due to a decrease in throughput. After that one or two intermediate quality levels are inserted and played. Finally, the player reaches the target level, l_0.

$\mathbb{R}_0$ is the base case providing the worst video quality. The rule sets denoted with $\mathbb{R}_{1,x}$, $\mathbb{R}_{3,x}$, and $\mathbb{R}_{8,x}$ are of buffer size 1, 3, and 8 segments, respectively. $\mathbb{R}_{3,8}$, $\mathbb{R}_{3,9}$, and $\mathbb{R}_{3,10}$ have two intermediate levels. Half of the buffer is used to load a higher quality level, and the remaining half is used to load a lower quality level. Figure 3.12 depicts the quality level transition of $\mathbb{R}_{1,4}$, $\mathbb{R}_{3,7}$, and $\mathbb{R}_{3,8}$. $\mathbb{R}_{1,4}$ and $\mathbb{R}_{3,8}$ have one and two intermediate levels, respectively, while $\mathbb{R}_{3,7}$ starts with quality level l_2. The buffer of $\mathbb{R}_{8,4}$ and $\mathbb{R}_{8,5}$ is large enough to allow the intermediate level to be used until the end of video clip. Therefore, these two rule sets do not switch to quality level l_0.

Table 3.4: The maximum number of segments supported at intermediate levels, n_{frag}.

	Buffer size		
Quality levels	1 (segment)	3 (segments)	8 (segments)
l_4	0	1	4
l_3	0	2	6
l_2	1	4	11
l_1	5	15	42

Table 3.5: Summary of experiment rule sets defining different quality transition schemes.

$\mathbb{R}_0$	$< l_0, 23 \times \mathcal{S}_{l_0} >$
$\mathbb{R}_{1,1}$	$< l_4, 3 \times \mathcal{S}_{l_4} >, < l_4, 1 \times \mathcal{S}_{l_4} >, < l_2, 1 \times \mathcal{S}_{l_2} >, < l_0, 17 \times \mathcal{S}_{l_0} >$
$\mathbb{R}_{1,2}$	$< l_4, 3 \times \mathcal{S}_{l_4} >, < l_4, 1 \times \mathcal{S}_{l_4} >, < l_1, 5 \times \mathcal{S}_{l_1} >, < l_0, 13 \times \mathcal{S}_{l_0} >$
$\mathbb{R}_{1,3}$	$< l_4, 3 \times \mathcal{S}_{l_4} >, < l_4, 1 \times \mathcal{S}_{l_4} >, < l_0, 19 \times \mathcal{S}_{l_0} >$
$\mathbb{R}_{1,4}$	$< l_2, 3 \times \mathcal{S}_{l_2} >, < l_2, 1 \times \mathcal{S}_{l_2} >, < l_1, 5 \times \mathcal{S}_{l_1} >, < l_0, 13 \times \mathcal{S}_{l_0} >$
$\mathbb{R}_{1,5}$	$< l_2, 3 \times \mathcal{S}_{l_2} >, < l_2, 1 \times \mathcal{S}_{l_2} >, < l_0, 19 \times \mathcal{S}_{l_0} >$
$\mathbb{R}_{3,1}$	$< l_4, 3 \times \mathcal{S}_{l_4} >, < l_4, 3 \times \mathcal{S}_{l_4} >, < l_4, 1 \times \mathcal{S}_{l_4} >, < l_0, 16 \times \mathcal{S}_{l_0} >$
$\mathbb{R}_{3,2}$	$< l_4, 3 \times \mathcal{S}_{l_4} >, < l_4, 3 \times \mathcal{S}_{l_4} >, < l_3, 2 \times \mathcal{S}_{l_3} >, < l_0, 15 \times \mathcal{S}_{l_0} >$
$\mathbb{R}_{3,3}$	$< l_4, 3 \times \mathcal{S}_{l_4} >, < l_4, 3 \times \mathcal{S}_{l_4} >, < l_2, 4 \times \mathcal{S}_{l_2} >, < l_0, 13 \times \mathcal{S}_{l_0} >$
$\mathbb{R}_{3,4}$	$< l_4, 3 \times \mathcal{S}_{l_4} >, < l_4, 3 \times \mathcal{S}_{l_4} >, < l_1, 15 \times \mathcal{S}_{l_1} >, < l_0, 2 \times \mathcal{S}_{l_0} >$
$\mathbb{R}_{3,5}$	$< l_4, 3 \times \mathcal{S}_{l_4} >, < l_4, 3 \times \mathcal{S}_{l_4} >, < l_0, 17 \times \mathcal{S}_{l_0} >$
$\mathbb{R}_{3,6}$	$< l_2, 3 \times \mathcal{S}_{l_2} >, < l_2, 3 \times \mathcal{S}_{l_2} >, < l_1, 15 \times \mathcal{S}_{l_1} >, < l_0, 2 \times \mathcal{S}_{l_0} >$
$\mathbb{R}_{3,7}$	$< l_2, 3 \times \mathcal{S}_{l_4} >, < l_2, 3 \times \mathcal{S}_{l_4} >, < l_0, 17 \times \mathcal{S}_{l_0} >$
$\mathbb{R}_{3,8}$	$< l_4, 3 \times \mathcal{S}_{l_4} >, < l_4, 3 \times \mathcal{S}_{l_4} >, < l_3, 1 \times \mathcal{S}_{l_3} >, < l_2, 2 \times \mathcal{S}_{l_2} >, < l_0, 14 \times \mathcal{S}_{l_0} >$
$\mathbb{R}_{3,9}$	$< l_4, 3 \times \mathcal{S}_{l_4} >, < l_4, 3 \times \mathcal{S}_{l_4} >, < l_3, 1 \times \mathcal{S}_{l_3} >, < l_1, 7 \times \mathcal{S}_{l_1} >, < l_0, 9 \times \mathcal{S}_{l_0} >$
$\mathbb{R}_{3,10}$	$< l_4, 3 \times \mathcal{S}_{l_4} >, < l_4, 3 \times \mathcal{S}_{l_4} >, < l_2, 1 \times \mathcal{S}_{l_2} >, < l_1, 7 \times \mathcal{S}_{l_1} >, < l_0, 9 \times \mathcal{S}_{l_0} >$
$\mathbb{R}_{8,1}$	$< l_4, 3 \times \mathcal{S}_{l_4} >, < l_4, 7 \times \mathcal{S}_{l_4} >, < l_4, 4 \times \mathcal{S}_{l_4} >, < l_0, 9 \times \mathcal{S}_{l_0} >$
$\mathbb{R}_{8,2}$	$< l_4, 3 \times \mathcal{S}_{l_4} >, < l_4, 7 \times \mathcal{S}_{l_4} >, < l_3, 6 \times \mathcal{S}_{l_3} >, < l_0, 7 \times \mathcal{S}_{l_0} >$
$\mathbb{R}_{8,3}$	$< l_4, 3 \times \mathcal{S}_{l_4} >, < l_4, 7 \times \mathcal{S}_{l_4} >, < l_2, 11 \times \mathcal{S}_{l_2} >, < l_0, 2 \times \mathcal{S}_{l_0} >$
$\mathbb{R}_{8,4}$	$< l_4, 3 \times \mathcal{S}_{l_4} >, < l_4, 7 \times \mathcal{S}_{l_4} >, < l_1, 13 \times \mathcal{S}_{l_1} >$
$\mathbb{R}_{8,5}$	$< l_2, 3 \times \mathcal{S}_{l_2} >, < l_2, 7 \times \mathcal{S}_{l_2} >, < l_1, 13 \times \mathcal{S}_{l_1} >$

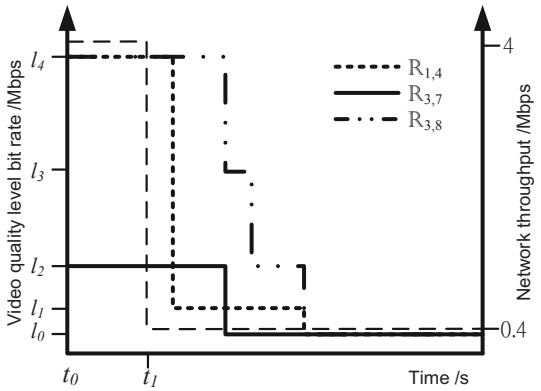


Figure 3.12: The quality transition of $\mathbb{R}_{1,4}$, $\mathbb{R}_{3,7}$, and $\mathbb{R}_{3,8}$.

Video materials

We prepared 11 short video clips, each with a duration of about 90 seconds. The video clips were taken from various kinds of sources, including sports, movie trailers, animations, and music videos. The quality of the source video was at least equal to that

of l_4 in Table 3.6. We subsequently downsampled the source video clips to other quality levels. To emulate more realistic environment, we used the video quality specification adopted by Akamai Adaptive Video Streaming. Table 3.6 shows the profiles of all the quality levels [63]. Adobe's File Packager [14] was then used to package the video files of different quality levels and translate the encoded video files into segments. We define segment length, θ, be four seconds of video, which is the default value in the File Packager. Therefore, each video clip contains about 23 ($\lceil 90/4 \rceil$) segments.

Table 3.6: Profiles of the video quality levels [63].

Parameters	l_0	l_1	l_2	l_3	l_4
Video height (pixel)	180	360	360	720	720
Video width (pixel)	320	640	640	1280	1280
Avg. video bit rate (kbps)	300	700	1500	2500	3500
Avg. audio bit rate (kbps)	160	128	128	128	128
Video frame rate (fps)	29.97	29.97	29.97	29.97	29.97
Video codec	H.264	H.264	H.264	H.264	H.264
Audio codec	AAC	AAC	AAC	AAC	AAC

Subject assessment

Subjects were told that a sudden drop in network throughput was emulated during each experiment. They were advised to watch the video in full screen, and not to pause or time-shift the video playback. Each subject was asked to watch 11 video clips, as mentioned in Section 3.5.1. We applied one of the experiment rule sets from Table 3.5 to the video playback. $\mathbb{R}_0$ was shown to all subjects. For the other ten experiment sessions, the video player randomly selected one of the experiment rule sets. Therefore, the sequence for all subjects was randomized to mitigate the order effect.

After completing each playback, subjects were immediately required to answer questions on the video playback they just watched. We first asked the subject if they

notice any quality change in video quality during the playback. Then, the subjects were asked to separately rate the perceived quality on the following aspects – picture quality, sound quality, playback smoothness, and video content. Finally, they were asked to give a composite score on the overall perceived quality. We adopt a 7-point Likert scale, from '1' (Bad) to '7' (Excellent), to measure the MOS for higher granularity.

Assessment results

Descriptive statistic A total of 24 subjects, 19 males and 5 females, participated in the subject assessment. All the subjects were volunteers, non-experts in video quality assessment, and with normal vision. They have the basic computer skills to use the experiment platform. We obtained 242 valid samples of rating on overall perceived quality. Figure 3.13 depicts the frequency distribution of the overall QoE rating. No subject rated '1' and only two experiment sessions gave a rate of '7'. We think this is reasonable as there was no service interruption and only quality degradation took place in all cases. We also validated that all the rule sets and the choice of video were evenly distributed among all the samples.

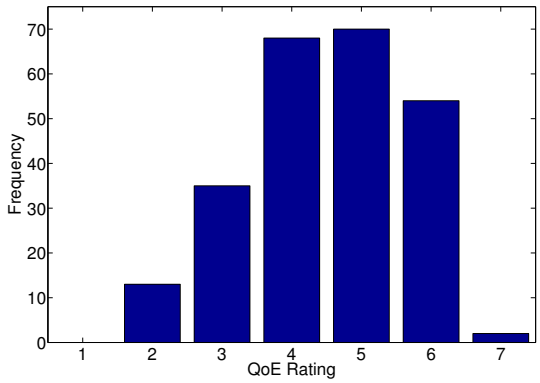


Figure 3.13: The overall distribution of overall QoE rating.

Comparisons on MOSes We compare the QoE between all the rule sets by computing the MOS difference as given by (3.11).

$$\Delta MOS(i,j) = MOS(\mathbb{R}_i) - MOS(\mathbb{R}_j) \tag{3.11}$$

,where $MOS(\mathbb{R}_j)$ is the mean QoE rating obtained using rule set $\mathbb{R}_j$. In addition, we also determine the significance level of the MOS difference, p, using Independent-samples *t*-test. Table 3.7 shows the MOS difference between rule sets, ΔMOS(column,row). We highlight the MOS differences at the significant level less than 0.1 (p <0.1).

Our results indicate that, $\mathbb{R}_{3,4}$, $\mathbb{R}_{8,4}$, and $\mathbb{R}_{8,5}$ obtain a negative MOS difference with respect to most of the other rule sets in the respective rows, meaning that these three rule sets have a higher MOS compared to other cases and outperform the other rule sets in terms of QoE. However, no significant difference in MOS is observed among these three cases. $\mathbb{R}_{8,4}$ and $\mathbb{R}_{8,5}$ do not drop to the lowest quality level, l_0, at the end of the video playback. Therefore, the higher QoE could be due to the higher final picture quality. The MOSes of $\mathbb{R}_{8,4}$, and $\mathbb{R}_{8,5}$ are significantly higher than that of $\mathbb{R}_{8,1}$ which plays the longest time in highest quality level. This shows that providing as high video quality as possible does not lead to the highest QoE. Surprisingly, $\mathbb{R}_0$, which plays at the lowest quality level throughput the whole playback, does not obtain the lowest MOS. As there is no quality transitions, the subjects did not know they were watching the video with the lowest quality.

Rule sets without intermediate quality levels usually have lower perceived quality. One example is that $\mathbb{R}_{3,5}$ has lower MOS than six other rule sets with intermediate levels. However, using two intermediate levels ($\mathbb{R}_{3,8}$, $\mathbb{R}_{3,9}$, $\mathbb{R}_{3,10}$) shows only slight impact in improving the QoE. Among the three rule sets, the QoE for $\mathbb{R}_{3,8}$ is significantly higher than five other rule sets.

On the other hand, large video buffer size does not necessary improve the QoE.

Many rule sets with small buffer size have no significant difference in MOS to those with large buffer size. One example is $\mathbb{R}_{3,4}$ which provides better perceived quality than $\mathbb{R}_{8,1}$. However, larger video buffer consumes more resources at the client-side, such as cache memory.

To summarize the results, inserting intermediate levels usually gives a better QoE, while one intermediate level gives the highest perceived quality. A small video buffer can only have short period to have intermediate level, and usually produces lower QoE.

3.5.2 The impact of initial bitrate setting to the QoE

The default bitrate can definitely impact the QoE. If the bitrate is too high, the playback will be stalled by rebuffering events. Previous works had shown that these events can degrade the QoE [165] and also the user engagement [70]. However, we found that using a conservative choice cannot meet user expectations and impacts to the overall QoE. To quantify the impact of low default quality to the QoE, we carry out subject assessments to measure the user perceived quality under different initial settings and transition schemes. Our assessment emulates users streaming videos with DASH using a high-speed network connection, which can support the highest bitrate of the video without any rebuffering. But, the initial bitrate is set to a suboptimal value and then switches up until the highest bitrate.

Experiment settings

To emulate different quality adaptation methods, we composed 17 (15+2) cases, denoted by R_{id}, to switch up the quality levels to the BIBR as listed in Table 3.8. The transitions are represented using the right arrows, $\rightarrow$. For example, $l_x \rightarrow l_y$ means the quality levels switch from l_x to l_y. Each level was played for two segments before

Table 3.7: The MOS difference, ΔMOS(column,row).

	$\mathbb{R}_0$	$\mathbb{R}_{1,1}$	$\mathbb{R}_{1,2}$	$\mathbb{R}_{1,3}$	$\mathbb{R}_{1,4}$	$\mathbb{R}_{1,5}$	$\mathbb{R}_{3,2}$	$\mathbb{R}_{3,3}$	$\mathbb{R}_{3,5}$	$\mathbb{R}_{3,6}$	$\mathbb{R}_{3,7}$	$\mathbb{R}_{3,10}$	$\mathbb{R}_{8,1}$	$\mathbb{R}_{8,2}$
$\mathbb{R}_{3,1}$							-1.03†							
$\mathbb{R}_{3,3}$							-0.70†							
$\mathbb{R}_{3,4}$	-1.03*	-1.25*	-1*	-0.88†	-1.23**	-0.93*	-1.67**	-0.97*	-1.25**	-1.03*		-1.21*	-0.98*	
$\mathbb{R}_{3,7}$		-0.92†			-0.9†		-1.33*		-0.92†					
$\mathbb{R}_{3,8}$		-0.92†			-0.9*		-1.33**		-0.92*			1.12†		
$\mathbb{R}_{3,9}$							-1.23†		-0.82†					
$\mathbb{R}_{3,10}$											0.88†			
$\mathbb{R}_{8,2}$							-1.03*							
$\mathbb{R}_{8,3}$								-1†						
$\mathbb{R}_{8,4}$	-1.12*	-1.33**	-1.10*	-0.96*	-1.32**	-1.02*		-1.75***	-1.05**	-1.33**	-1.12*	-1.29**	-1.06*	-0.72†
$\mathbb{R}_{8,5}$		-1.52*	-1.27*	-1.15†	-1.5*	-1.2*		-1.93*	-1.23*	-1.52**	-1.3*	-1.48*	-1.24†	

Note: $\dagger p$ <0.1, $*p$ <0.05, $**p$ <0.01, $***p$ <0.001

the next transitions. R_{-2} and R_{-1} are control cases emulating the worst and the best overall picture quality, respectively. These two cases do not have any quality change in the entire video playback. For other cases, the quality levels will monotonically switch up to l_4 and keep steady until the end. The video was streamed through a Flash-based customized video player, which was implemented on top of the Strobe Media Playback, and allowed us to override the internal quality adaptation algorithm and switch the quality levels according to the preset rules.

We downloaded four different kinds of HD video clips from the YouTube (including News, sports video, music video, and movie trailer) to randomize the effect from the video content. Then, the video clips were trimmed to 1 minute and were encoded into five quality levels (denoted by l_0 to l_4) according to Adobe's encoding recommendation [142] (first five levels of Variant 3), where l_0 is the lowest quality. The segment length was 4s, which is the default value for the Adobe HTTP Dynamic Streaming's file packager. We argue that the video length in our subjective test is sufficient and realistic. Measurement studies of YouTube [86, 55] and Akamai [137] showed that short video clips are still very popular in today's Internet.

A crowdsourcing-based QoE assessment

We employed MTurk and CrowdFlower to carry out the subjective tests using crowdtesting approach [95]. We implemented a web-based crowdtesting platform similar to [195]. The workers were instructed to visit our test system, which was a website hosted on an Amazon EC2 instance. Because this assessment focused on the effect of picture quality instead of the playback smoothness, we used CloudFlare CDN to cache the video data to mitigate any rebuffering events due to insufficient network throughput at the server side. From the log returned by the video player, we confirmed that no rebuffering events were observed. Each worker was asked to watch and rate on

perceived quality of the four video clips. Among the four video clips to be watched by the worker, one of them is a control case (either R_{-2} or R_{-1}) and others are randomly selected from the 15 cases with quality transitions. After finished playing each video, a questionnaire immediately shows up and requires the worker to rate on the overall perceived quality in a 5-point Likert scale (1: Bad - 5: Excellent) [114]. Furthermore, we also measure their expectation by asking them whether the initial and overall picture quality meets their expectations in a 5-point Likert scale (1: Strongly disagree - 5: Strongly agree), denoted by E_{init} and $E_{overall}$, respectively. Each worker was awarded from USD 0.5 to 1. We successfully recruited 209 subjects to perform this assessment after screening our low-quality workers using [167].

Table 3.8: Summary and the MOS of different quality transition schemes.

ID	Transition	MOS	$\overline{E_{init}}$	$\overline{E_{overall}}$	
R_{-2}	l_0	3.21	3.07	3.09	129
R_{-1}	l_4	4.16	4.23	3.99	90
R_0	$l_0 \rightarrow l_4$	3.92	3.05	3.65	130
R_1	$l_1 \rightarrow l_4$	4.06	3.86	4.06	35
R_2	$l_2 \rightarrow l_4$	4.15	3.94	3.88	33
R_3	$l_3 \rightarrow l_4$	4.13	3.98	3.90	40
R_4	$l_0 \rightarrow l_1 \rightarrow l_4$	3.55	3.00	3.50	38
R_5	$l_0 \rightarrow l_2 \rightarrow l_4$	3.90	3.00	3.57	30
R_6	$l_0 \rightarrow l_3 \rightarrow l_4$	3.79	2.79	3.29	34
R_7	$l_1 \rightarrow l_2 \rightarrow l_4$	4.00	3.83	4.11	35
R_8	$l_1 \rightarrow l_3 \rightarrow l_4$	4.26	4.03	4.13	31
R_9	$l_2 \rightarrow l_3 \rightarrow l_4$	4.13	3.63	4.03	40
R_{10}	$l_0 \rightarrow l_1 \rightarrow l_2 \rightarrow l_4$	3.78	3.24	3.49	37
R_{11}	$l_0 \rightarrow l_2 \rightarrow l_3 \rightarrow l_4$	4.00	3.50	3.87	30
R_{12}	$l_0 \rightarrow l_1 \rightarrow l_3 \rightarrow l_4$	3.63	3.47	3.63	30
R_{13}	$l_1 \rightarrow l_2 \rightarrow l_3 \rightarrow l_4$	4.23	3.58	4.02	43
R_{14}	$l_0 \rightarrow l_1 \rightarrow l_2 \rightarrow l_3 \rightarrow l_4$	3.77	3.42	3.74	31

The MOS, $\overline{E_{init}}$, $\overline{E_{overall}}$, N column in Table 3.8 show the mean opinion score, the mean value of the expectation rating of initial, overall picture quality, and the number of samples, respectively. All the cases are evaluated by at least 30 subjects. The cases with darker background color have more video quality transitions. We can see that

all the cases (R_0, R_4, R_5, R_6, R_{10}, R_{11}, R_{12}, R_{14}) started from the lowest quality level, l_0, have a lower MOS, E_{init}, and $E_{overall}$ than the similar schemes starting from l_1 (R_1, R_7, R_8, R_{13}). We further compute the correlation coefficient among these three metrics. The MOS values show significant positive correlation with both E_{init} ($r = 0.42, p < 0.001$) and $E_{overall}$ ($r = 0.71, p < 0.001$). Although the correlation coefficient for E_{init} is smaller than $E_{overall}$, we can conclude that the initial quality shows significant influence to the overall QoE. Furthermore, R_{14} shows a lower MOS to other schemes with less number of transitions (e.g., R_5, R_6, R_{11}). We can reveal that the longer duration of the transition phase can also hurt the QoE.

3.6 Summary

This chapter presented our QoE measurement studies on HTTP streaming and HAS. In the first part, we studied how network path quality affects QoE of HTTP streaming. We addressed the problem by dividing it into two subproblems: measuring the correlation between the network QoS and application QoS, and measuring the correlation between application QoS and QoE. In the first subproblem, we proposed three application performance metrics for HTTP video streaming and used both analytical model and empirical evaluation to characterize their correlation. In the second subproblem, subjective assessments were conducted to correlate the QoE in terms of MOS and the application QoS.

Our main finding is that network throughput is lowered by packet losses and the RTT, thus increasing the rebuffering frequency. We also identified the rebuffering frequency to be the main factor responsible for the MOS variance. This shows that the temporal structure, instead of spatial artifacts, is also an important factor affecting the QoE. The QoE could be improved by both network QoS or application QoS management. Moreover, our approach allows us to inspect the correlation between QoS

and the QoE from different aspects, and the approach could be easily applied to other application environments.

In the second part, we investigated the impact of quality adaptation in HAS on the QoE. We designed two subjective assessments to examine the differences in perceived quality under the two emulated network conditions. In the first assessment, we emulated a steep in network throughput. We found that inserting an immediately quality level in between a downshift of quality levels can cushion the effects of spatial quality degradations. We emulated a high throughput environment in the second crowdsourcing based assessment. We found that starting from a low video quality can result in lower QoE even the quality eventually ramps up. We believed that this is because the low initial quality fails to meet user expectations. These findings are very important in designing a QoE-aware video streaming system.

Chapter 4

Improving the Initial Video Bitrate

Our subjective assessments in the previous chapter reveal that low initial video cannot provide the best QoE to users. Motivated by this finding, we design and implement a practical system to determine the Initial bitRate (abbv. *IRate*). Our user-behavior driven design helps explore possible time frames for pre-stream measurements. Moreover, we exploit a number of tactics in the TCP/IP layer and the web technology to implement the measurement component. The measurement core of IRate is designed as a measurement box, which can be easily deployed along with the existing video caches and servers to collect accurate network path quality data at the server side. The measurement core can accommodate packet-pair based bandwidth estimation algorithms to measure the network quality at the server side. By imbedding a script in web pages, clients can measure dedicated IRate-enabled video caches through the browser. Based on the measurement results, IRate profiles clients by determining their BIBR to video caches. The web server utilizes this information and redirects users to a better video cache which could serve the highest BIBR for the best QoE.

We have conducted extensive testbed experiments to evaluate IRate's accuracy of estimating the BIBR and its robustness when operating under diverse network conditions. Our results show that using IRate can achieve a 80% accuracy using only 10

seconds of measurement data. It also achieves better stability and higher efficiency than DASH with the default setting. We further conducted user assessments to compare the performance of IRate and the predefined approach in terms of QoE.

In this chapter, we first survey the default initial bitrate settings used in several popular video streaming services and a subject assessment on different initial bitrate settings in Section 4.1. We present our user-behavior driven approach for estimating the BIBR in Section 4.2 and then describe IRate, our implementation based on this approach, in Section 4.3. Section 4.4 presents our testbed results for evaluating the accuracy and robustness of IRate. We present a user QoE experiment for IRate in Section 4.5. We then discuss some limitations of IRate in Section 4.6.

4.1 Background

4.1.1 Default settings for initial video quality

Video streaming providers generally offer several quality levels or video bitrates for each video. However, whether they support adaptive bitrate streaming or not, they set their default initial bitrates oblivious to the performance of the end-to-end path between the client and video server. We have surveyed several video content providers on their default settings, and summarized in Table 4.1. Since the default settings are not always stated clearly in public documentations, we have performed testings to uncover their initial bitrate settings. However, we cannot perform the tests for Hulu, because its service are not available locally. We also show in the table whether they support adaptive streaming, the number of supported bitrates, and whether they have any pre-roll advertisement.

As shown in Table 4.1, there are three types of initial bitrate settings. YouTube sets the initial bitrate to the highest quality based on the video player size [244]. We

Table 4.1: Streaming video services provided by major video service providers.

Providers	Streaming type	No. of bitrates/ quality levels supported	Initial bitrate/quality	Advertisement
YouTube	DASH-enabled	2-6	Player size [244]	Non-skippable/Skippable after 5s
Netflix	DASH-enabled	12-14 [12]	Country [178]	No
Hulu	DASH-enabled	3	N/A	No
Vimeo	Static	2	Predefined (HD)	No
Dailymotion	Static*	2-5	Predefined (HQ) [163]	Skippable after 4s
Youku	Static*	3	Predefined ($\sim$200 Kbps)	Non-skippable
Tudou	Static*	3-4	Predefined ($\sim$500 Kbps)	Non-skippable

*These results are based on our own tests as we cannot obtain the data from public sources.

also observe from our tests that the corresponding quality levels for “small”, “large”, and “full screen” are 360p, 480p, and 1080p, respectively. Netflix, on the other hand, determines the default video quality settings by countries. For example, for Canada and Brazil users, the quality level is set to “good”, and other users are set to “best” [178]. We are however unable to determine the setting for Hulu. As for the other three, our tests show that they choose predefined quality for the initial bitrates. For example, Youku streams video clips in a standard definition. Dailymotion selects “HQ” as the default bitrate [163], where we observe that the resolution can be 360p or 480p.

We also perform supplementary tests to verify whether the default settings for new clients will be changed according to different network conditions. We run a Windows 7 virtual machine as the client. We then configured the capacity of the virtual machine’s network interface in the `VMware` to 10Kbps, 200Kbps, and unlimited. The downstream link for receiving video is therefore bandwidth throttled. After setting the network, we arbitrarily select at least three video clips from the front page to stream from the YouTube, Vimeo, Dailymotion, Youku, and Tudou websites with Internet Explorer. To emulate new clients with no prior knowledge of the throughput, the *InPrivate* mode is used to avoid local caching or any tracking of user preferences. We believe the videos shown on the front page are popular videos which have been cached nearby CDN caches. All the tests show that their default values do not change for the three different downlink bandwidth scenarios, thus further supporting that their initial bitrates are oblivious to the path performance.

4.1.2 Impacts of the default settings

To understand the impact of the default settings, we employ a two-step approach. The first step is to measure the degree of discrepancy between the default setting and the BIBR. Then, we present a subjective experiment to show how the default settings im-

pact on the user perceived quality.

Discrepancy between default settings and the BIBR

We performed throughput measurements for YouTube, Dailymotion, Youku, and Tudou. Our methodology is to arbitrary select a video from the front page of each provider. We captured the traffic using `wireshark` while playing the video clip. Then we computed the average throughput of the whole video stream. Furthermore, to capture the potential influences from the TCP slow-start, we also compute the average throughput of downloading the first 250KBytes of video data as the start-up throughput. For each video, we repeated the download for five times. Finally, the BIBR is determined by mapping to bitrate closest to the measured throughput.

Table 4.2 shows the results of comparing the default initial bitrates and the BIBRs based on the throughput measurement. Except for YouTube, the initial bitrates set by the other three providers do not agree with the BIBRs. Both Youku/Tudou can provide a higher quality than the default settings, whereas Dailymotion's exceeds the actual throughput provision. YouTube, on the other hand, can provide the highest quality for the BIBR. Surprisingly, the start-up throughput in Youku/Tudou and Dailymotion is higher than the average throughput of the whole video stream. The start-up throughput for both providers can support one quality level higher than the overall one. Even though YouTube shows a lower start-up throughput, the start-up throughput is still sufficient for supporting the same (highest) video bitrate. Since their default settings for all player sizes are less than the BIBR, theirs are sufficient for guaranteeing the best QoE at the onset of the viewing. YouTube's exceptional throughput performance is due to their local cache servers [13], our laboratory's high-speed network (100 Mbps), and their aggressive buffering strategies [80]. For the other three providers, we performed `traceroute` with TCP SYN packets and confirmed that the video servers are

not located locally. It is believed that the throughput is limited by the overseas link.

We next present two testbed traces to illustrate the impact of suboptimal initial bitrates (the full set will be presented in Section 4.4). The first case is classic HTTP streaming, and there are packet losses on the path. We plot in Figure 4.1 the video playhead times (the solid lines) and buffer status (the dotted lines) for two cases: a default initial bitrate of level 3 in a scale of 0 to 4 (the light red) and the BIBR (the dark green). Notice that the default bitrate is too high for the network condition. Hence, the playback suffers from a long initial buffering time (>10s). It then pauses at around 30s for rebuffering and resumes at 60s. The case using the BIBR, on the other hand, chooses the lowest bitrate, thus yielding a smooth playback.

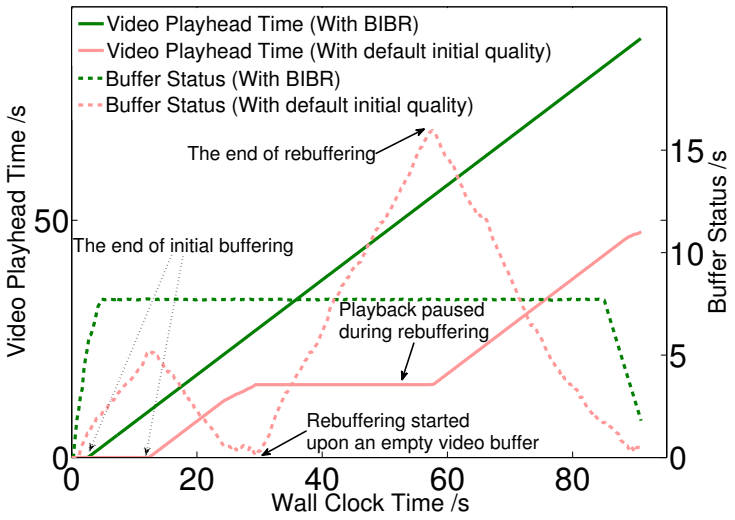


Figure 4.1: Video playhead times and buffer statuses for default initial bitrate and BIBR in classic HTTP streaming under a lossy network condition.

The second case shows the first 15s of the traces for DASH streaming with a default initial bitrate and the BIBR. The network condition is better than the first case, which can support level 4, the highest quality. Figure 4.2 shows the quality levels against the video playhead time. The BIBR is used throughout the entire period, whereas the default case is started with the lowest quality (level 0) and reaches the final quality

Table 4.2: A comparison of the default initial bitrates and the BIBRs for sample videos.

Providers	Possible bitrates of the sample video	Default initial bitrates	Source server	Avg. throughput (BIBR)	
				Overall	Start-up
YouTube	144p (82Kbps), 240p (226Kbps), 360p (546Kbps), 480p (756Kbps), 720p (1577Kbps), 1080p (3364Kbps)	Based on the player size	202.40.221.15	9.22Mbps (1080p)	3.63Mbps (1080p)
Dailymotion	Low (246Kbps), Medium (459Kbps), Standard (829Kbps), HD (1618Kbps)	Standard	188.65.126.17	0.66Mbps (Medium)	1.23Mbps (Standard)
Youku/ Tudou✠	Standard (285Kbps), High (547Kbps), Super High (1197Kbps)	Standard	23.236.118.48	1.08Mbps (High)	1.36Mbps (Super High)

Note: ✠In our experiment, we found that Youku and Tudou streamed the video from the same video server.

level after two consecutive up-switchings at 4s and 8s. The shaded region therefore refers to the amount of under-utilized bandwidth for the default case. Depending on the video segment length, buffer size and the aggressiveness of the quality adaptation algorithm, users could experience an unstable video quality for more than 150s [106].

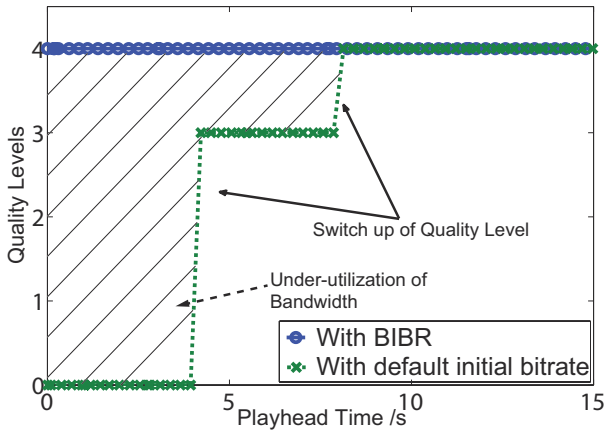


Figure 4.2: Quality levels vs. video playhead times for default initial bitrate and BIBR under DASH streaming in a good network condition.

4.2 User-behavior driven design

4.2.1 Design goals

We design IRate by considering three aspects—users, video service providers, and system accuracy. Our objective is to derive the BIBR from pre-stream measurement results. Moreover, the current infrastructure and user habit can be preserved. From users' point of view, the pre-stream measurement cannot disturb the normal user behavior, especially blocking the onset of the streaming after they selected the video or generating excessive amount of measurement traffic. On the other hand, the design of IRate has to be practical for service providers to deploy it to the current distributed video delivery infrastructure. More importantly, IRate needs to collect accurate enough path performance data, in a lightweight way, to determine the BIBR.

These issues are not independent of each other, because the measurement accuracy also depends on the measurement method and measurement duration. The amount of time allocated to the pre-stream measurement is clearly very limited. Moreover, in order not to block the video streaming, the pre-stream measurement must be conducted and completed before the user selection of the video. As a result, the measurement must be conducted in the "background" when the user is making his video selection. As the measurement is now run in parallel with the download of web objects of the current page, the choice of measurement method has to be lightweight. Furthermore, a practical design should also be able to accommodate the distributed architecture of large-scale video delivery systems.

4.2.2 Pre-stream measurement window

Our approach to designing IRate is to first study a user's typical behavior before selecting a video to watch. The user behavior will inform us the constraints, as well as the opportunities, for conducting pre-stream measurement to determine the BIBR. The five dark-bordered boxes in Figure 4.3 show the main actions performed by a user and his browser:

(1) The user visits the front page of a video streaming website, such as YouTube, which usually displays a video catalog.

(2) The browser starts downloading and rendering the web objects in the front page.

(4) When the page is partially parsed and rendered, the user may browse the page and acquire the video clips information from the page.

(7) After selecting a video to watch, the user clicks on the video's hyperlink/thumbnail to view the video page.

(8) The user may need to explicitly start the video streaming. A pre-roll video advertisement may be shown before streaming the requested video.

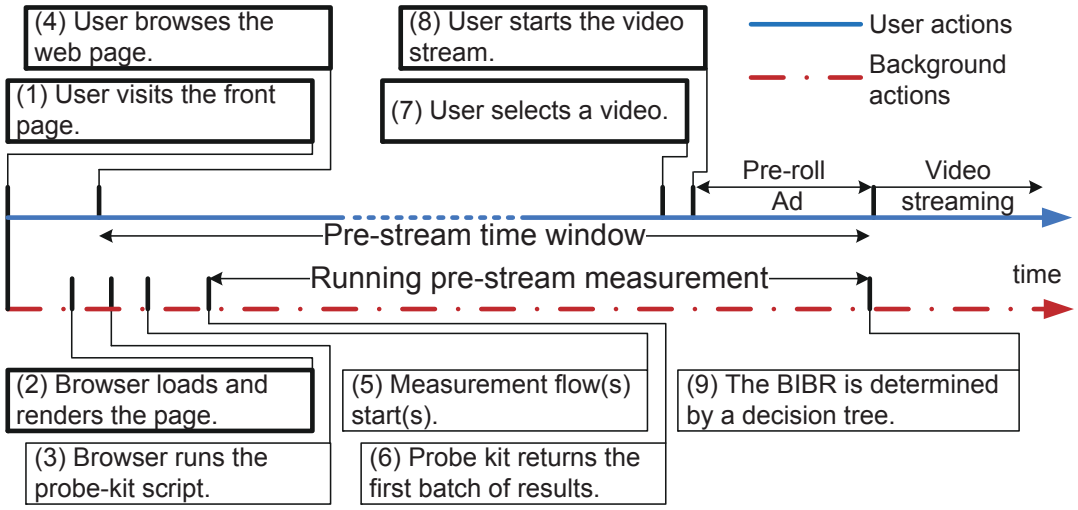


Figure 4.3: Typical user actions and background actions before starting a streaming video. The dark-bordered boxes are the actions for typical video streaming, whereas the light-bordered boxes are added for IRate-enabled video streaming.

We refer the period between action (4) and the onset of the video streaming to as a *pre-stream time window*. In this period, the user may engage in different types of activities. One of them is considering which page to visit next, which can be modeled as user think time. A measurement study [87] shows that the user think time for YouTube is about 30s, which is longer than traditional web transactions. Another typical event occurring in the pre-stream time window is showing short video advertisements (Ads), also known as pre-roll Ads, (typically 15-30s) [109, 138]. The Ads in some sites are not skippable. Even though YouTube's TrueView advertising package [9] and Dailymotion provide a "skip" option for users to jump over the Ad, users have to wait for at least 4s. As a result, the pre-stream time window will provide a window of about 34(=30+4)s for conducting the pre-stream measurement.

4.2.3 Measuring network path-quality metrics

Using a more conservative estimate, the window for pre-stream measurement is about 10s. Within such a small time window, it is very challenging to collect accurate enough

measurement data for determining the BIBR. A general approach is to estimate the network throughput and then select the video bitrate that is closest to the throughput. There are a number of methods/tools for measuring the (available) throughput. Besides flooding based method, packet pairs or packet trains can be used to measure the available bandwidth. Pathload [115], pathChirp [203], and PTR [103] employ probe rate model, which measures the network by self-induced congestion. Spruce [222] and IGI [103] are based on the probe gap model, which relies on the timing information carried in the time gap between a pair of probe packets after traversing the network. However, these tools are not designed for web clients, because they often need raw socket to send packets in a particular pattern.

To enable web clients to perform network measurement, a number of browser-based measurement tools have been developed in recent years. They are based on Adobe Flash [181], Java applet [133], and JavaScript [122]. Browsers allow these tools to elicit HTTP requests or initiate connections to servers. However, browsers are run on the application layer, which has limited access to the information from the network layer. It is hard to capture timestamps of specific probe packets and to send probe packets in a particular pattern. Besides, these tools may suffer a higher delay as they are running on the application layer [143]. Hence, they, such as speedtest [181], often measure the throughput by flooding, which incurs high overhead in order to obtain reasonably accurate results.

We investigate the accuracy of flooding based method by downloading five objects of different size (*cf.* Section 4.4.1 for details). We compare the average TCP throughput obtained between the largest object (4.3MBytes) and those obtained from the smaller objects by computing the percentage differences, $\Delta\beta_s$, by Equation (4.1).

$$\Delta\beta_s = \frac{\beta_s - \beta_{\text{4.3MB}}}{\beta_{\text{4.3MB}}} \times 100\%, \tag{4.1}$$

where β_s is the throughput measured using one of the smaller objects, and $\beta_{4.3\text{MB}}$ is the throughput measured by the 4.3-MB object.

Figure 4.4 shows a box-and-whisker plot of $\Delta\beta$ of the four small objects. The lower and upper edge of box gives the 25^{th} and the 75^{th} percentile, respectively, and the central line inside is the median. The lower and upper whiskers respectively are the minimum and maximum, after excluding the outliers. Outliers are defined as the data points exceeding 1.5 of the upper quartile, and those below the minimum are less than 1.5 of the lower quartile, which are marked as dots outsides the whiskers. From the figure, we can see that using small objects (18KBytes and 240KBytes) show a large disagreement and variance to $\beta_{4.3\text{MB}}$. Some cases overestimate the throughput by 200%. The difference reduces as the file size increased to 1.9MBytes. The inter-quartile range of $\Delta\beta_{1.9\text{MB}}$ is about 14.8%. Hence, high overhead of this kind of speed measurement is unavoidable.

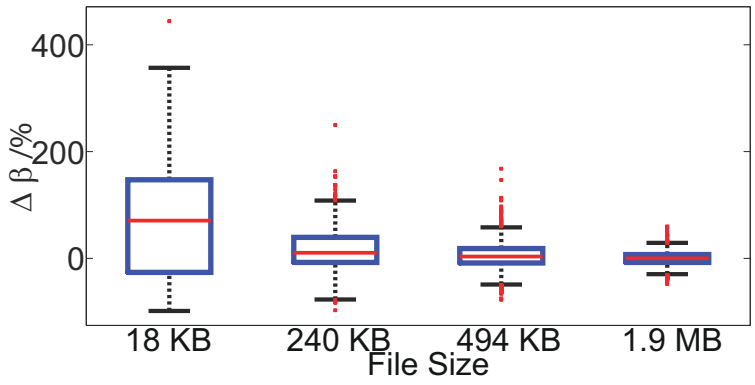


Figure 4.4: The percentage difference of average throughput against different web object sizes.

Another important consideration for measuring the network path quality is which side (user or video server) to conduct the measurement. Recall that the pre-stream measurement is to be performed before selecting the video to download. However, the measurement may have to terminate once users make the selection. Client-side tools may not be able to feedback the results timely to the server when the measurement is

still in progress.

This constraint therefore motivates us to employ a server-side active measurement paradigm for IRate. In this paradigm, the server side masters the measurement process, while the browser running at the client side is required only to send some dummy data to the server. This paradigm has several advantages.

1. It can provide more accurate network-layer measurement by optimizing the implementation and unifying the measuring agent. For example, a hardware-assisted measurement middlebox could be implemented based on Endace DAG card [76] or NetFPGA for high performance and accuracy.
2. The implementation can incorporate into other server-side in-line network appliances (e.g., firewall, IPS), shaper [17], or measurement middleboxes (e.g., QDASH [170]). These middleboxes are very common in today's enterprise network [213].
3. The browser does not need to install extra plug-ins or tools to cooperate with the measurement. In IRate, a probe-kit script is written in commonly used Web technology (e.g., a Flash object or HTML5 script), and is imbedded in a web page to induce data for measurement.
4. The server-side measurement facilitates the BIBR estimation, because all the data are collected on the server side and are readily available for BIBR estimation.

Our design does not restrict the probing method. But, in particular, we use a server-side version of TRIO [42] for network measurement. TRIO is a light-weight, non-cooperative packet-pair based measurement method. Figure 4.5 shows a box-and-whisker plot of the amount of data used in 10-second IRate measurement across different network conditions against the RTTs. The dotted blue and orange lines, respectively, show the minimum number of data used in downloading a 494-KB object and

a 1.9-MB object in Speedtest Mini package [182] as reference points. The median number of data used in IRate is less than that of downloading the 494-KB object. Although IRate consumes more data in some low-RTT cases, all of them are much less intrusive than downloading the 1.9-MB object. Hence, IRate is more light-weight than the flooding based measurement tools.

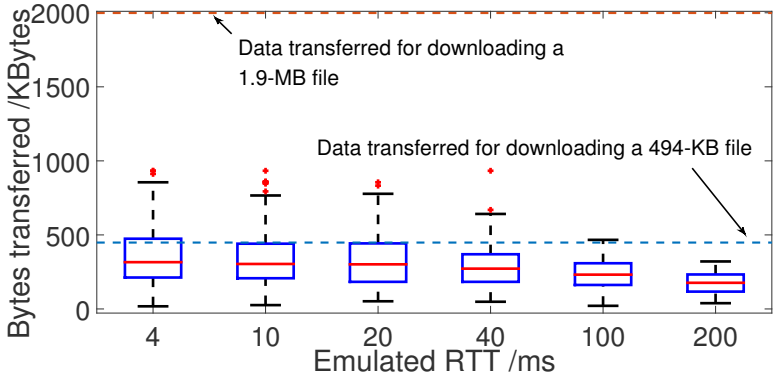


Figure 4.5: The number of Bytes transferred in 10-second IRate measurement against RTTs.

4.2.4 Methods for estimating the BIBR

We use the set of path-quality metrics collected in the pre-stream measurement to estimate of the BIBR. The metrics includes the delay, delay jitter, loss rate, reordering rate, and capacity for two paths: video server→user and user→video server. A major concern of estimating the BIBR is the computational speed, because the web server has to wait for the estimated BIBR for generating the parameters on the video page to control the initial bitrate. Otherwise, the IRate could become a bottleneck of the video delivery system. Instead, we can tolerate larger errors as the difference of bitrates among quality levels are often large.

We therefore consider a much faster and lightweight approach that assumes a pre-computed throughput model. Two examples are equation-based [184, 29] and decision tree. The equation-based method mathematically models the steady-state throughput

of a TCP flow using round-trip delay, packet loss rate, and bottleneck capacity. The decision tree, on the other hand, is based on a set of training data to construct a decision tree for determining the BIBR directly.

It is not our goal to compare the two methods' accuracy in this work. Instead, we focus on their computational efficiency and scalability, because of the small time window for measurement and large number of clients. Figure 4.6 shows a comparison of execution time between Padhye's model [184] for TCP Reno and the decision tree (*cf.* Section 4.3.2 for details). We implemented both methods with `perl` and randomly generated 100K to 10M sets of network path metrics as the input to predict the quality levels. For the equation-based approach, we use our bitrate quantization in Equation 4.2 (in Section 4.3.2) to convert the estimated throughput to a video quality level. We ran both methods on a Dell R210 Rack Mount server and used `time` command in Linux to record the execution times.

The results show that the decision tree (marked with crosses) is much faster than the equation-based method (marked with circles) by nearly three times. The execution time could be made shorter if we skip the bitrate quantization step (marked with squares). However, the equation-based method is still two times slower than the decision tree. The reason, we believe, is that the CPU requires more clock cycles for computing floating points than determining cases in decision tree. Hence, we adopt the decision tree approach in IRate as it can scale to a large number of users. We also note that we have considered other machine learning approaches, such as [35, 164, 127], but they are not suitable for IRate for various reasons. For example, Mirza's work [164] uses SVR to estimate TCP throughput, but the input metrics must be obtained from a cooperative measurement which is hard to deploy in browsers. Nunes et al. [35] combine a number of machine learning algorithms for predicting RTTs, but they cannot predict packet loss, which is a key metric for estimating TCP throughput.

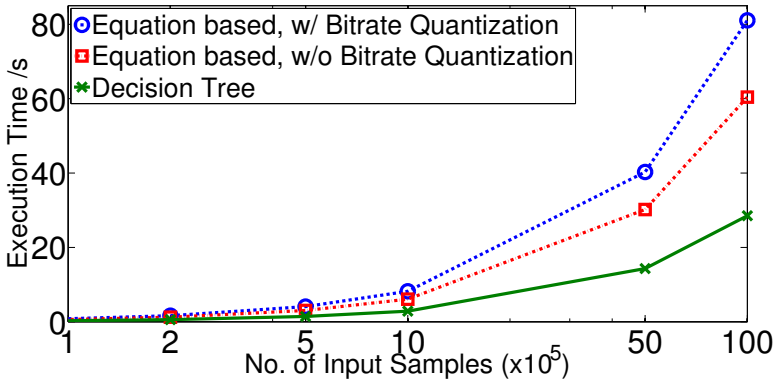


Figure 4.6: An execution-time comparison of the equation-based and decision tree methods for estimating the BIBR.

With IRate-enabled video streaming, four more light-bordered boxes ((3), (5), (6), and (9)) are added to Figure 4.3 for background actions. Besides, a probe-kit script is imbedded to the front page which is accessed by users in (1).

(3) The browser runs the probe-kit script.

(5) The probe-kit script starts establishing TCP measurement flow(s) with the video server.

(6) Pre-stream measurement begins when receiving the first batch of measurement results.

(9) Based on the network measurement data, a decision tree method is used to determine the BIBR for the video streaming.

4.3 An IRate implementation

We have implemented a prototype called IRate to estimate the BIBR based on the methods discussed in the last section. The square box in the middle of Figure 4.7 depicts IRate which is located before the streaming video website and video servers. To simplify the ensuing discussion, both the website which is first contacted by a user and the video server are assumed located in the same domain.

IRate consists of two building blocks: a *probe kit* and a *quality oracle*. The probe kit (*cf.* Section 4.3.1) is responsible for measuring the network performance during the pre-stream time window. It is transparent to both browser and video server, as it intercepts the measurement packets destined to the video server. The quality oracle (*cf.* Section 4.3.2) maintains a decision tree for determining the BIBR with the network performance data as inputs. Note that the numbers inside parentheses correspond to those in Figure 4.3.

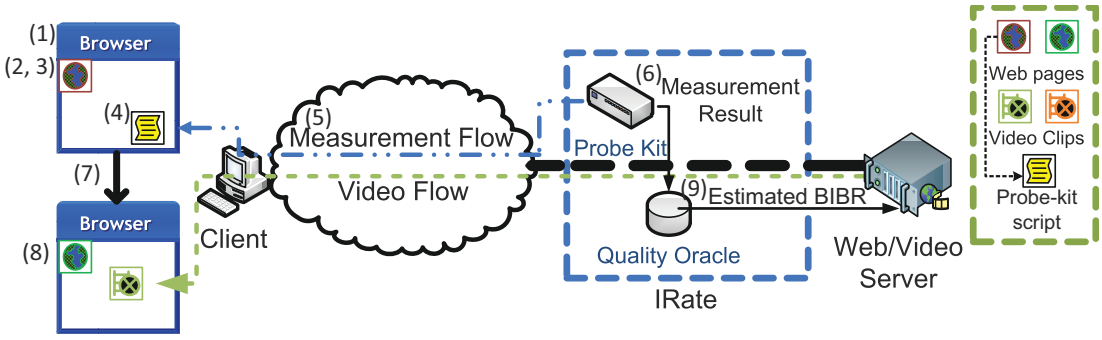


Figure 4.7: The main steps in IRate-enabled video streaming.

The design of IRate has considered the deployment in the large-scale video delivery infrastructure. Figure 4.8 shows the way of deploying IRate when the front-end and video servers are under different domains. We assume the front-end web server is `videoweb.com` for hosting the web pages of the video site. There are two video caches at different locations and domains to the web server, namely `v1.cache.com` and `v2.cache.com`.

The key is that the probe kit script and the IRate middlebox can be separately located at the front-end server and the video caches, respectively. When a client reaches the front-end server, the server can assign one or more IRate-enabled video cache(s) as the measurement target in the probe kit script. By utilizing the cross domain policy in Adobe Flash and WebSocket, the client can measure the two video caches at different domains to the web server. Then, the front-end server can query the respective IRate

middlebox for the BIBR of the client at the end of pre-stream timing window.

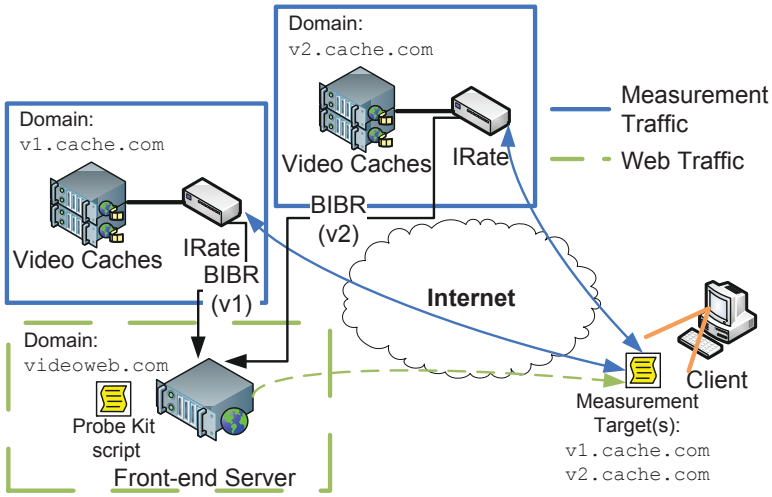


Figure 4.8: Deploying IRate in a large-scale video site.

4.3.1 Probe kit

The probe kit offers a *probe-kit script* and a *network measurement middlebox*. The measurement middlebox is located on the incoming path to a video server, and the probe-kit script is run at the user's browser. Together, they allow the middlebox to measure the quality of the network path between the middlebox and the user without the user's and video server's intervention. As pointed out in the last section, this server-side measurement paradigm alleviates the user from installing any measurement tool (e.g., Wireshark) and browser plugin (e.g., Fathom [67]).

Probe-kit script

The probe-kit script, hosted at the web server, requires high compatibility with various kinds of clients. Therefore, in our implementation, we prepare two versions of the probe-kit script using Adobe Flash and HTML5 WebSocket. Adobe Flash is still a de facto standard in Windows clients, while WebSocket is supported by latest version

of Apple Safari browser and mobile clients. To facilitate our testbed experiments, we have also implemented a text-based version of probe-kit script which can be run on Linux clients.

When the script is executed at the browser, it calls `Socket` in Flash ActionScript or the `WebSocket` to establish at least one TCP connection to the video server as measurement flows. It then prepares and sends a long dummy string to the measurement flow's socket buffer. As the data has been sent to the network stack, the delay overhead at the application layer can be mitigated. The script can also receive command from the middlebox to close the measurement flows.

Measurement middlebox

We implemented a prototype in a Linux box. The middlebox acts as a measuring node, while the user's machine as a remote node. Figure 4.9 shows the details of a measurement flow. The probe-kit script establishes at least one TCP connection with the web server through the browser. The middlebox also records the TCP states kept by both sides by examining the packets exchanged. Moreover, when the middlebox detects the IRate URI in the HTTP `POST` message used for measurement flow, it hijacks the flow by terminating the connection to the web server. We use the `NFQUEUE` library to intercept packets passing through the middlebox and raw socket to send out measurement probes.

After successfully hijacking the measurement flow, the middlebox continues to send probe packets (according to the TRIO probes [42]) to elicit more data from the user's browser for network measurement. The content of the probe packets is a legitimate HTTP response message, emulating a complete HTTP transaction in the measurement flow. This can effectively prevent raising the alarm of firewalls. When the response data prepared by the script are used up, the middlebox terminates the connec-

tion. According to [157], the middlebox can measure the round-trip time, and detect packet loss and reordering events on both unidirectional paths based on the response packets. In addition, TRIO cleverly exploits two types of probes to obtain three minimum RTTs to compute both forward and reverse capacities, and another minimum RTT for measurement validation.

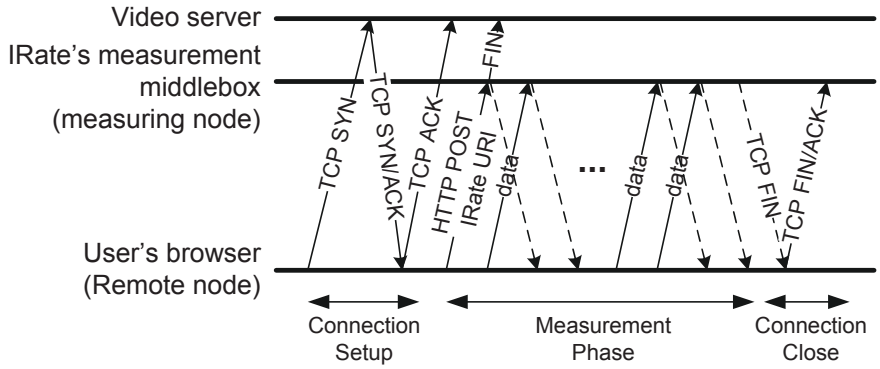


Figure 4.9: Probe kit's measurement flow between a user's browser and the measurement middlebox.

4.3.2 Quality oracle

The quality oracle returns an estimated BIBR or initial quality level when given the path measurement data from the probe kit. Due to the storage space consideration, only four to five quality levels are usually available for each video [63]. The quality oracle determines the BIBR or quality level based on a decision tree constructed from a set of training data. As the computation details of decision tree is out of the scope of this work, we mainly describe the method on how to prepare the inputs and build the decision tree. The single decision tree may not be able to capture the characteristic of different clients. The server-side may be manually built multiple trees with their own historical data to better capture the characteristics of different sets of clients, such as in certain ISPs, ASes, connection methods, or geographical locations [176, 89].

The training data for the decision tree generation is composed of a set of metrics

characterizing the performance of a network path and the actual BIBR. Table 4.3 summarizes the six network performance metrics considered here. The forward/reverse direction is referenced from the middlebox, because it acts as a measuring node. We do not include the packet reordering rates and reverse capacity, because our experience shows that they are not important for determining the BIBR. On the other hand, the actual BIBR is usually based on the actual throughput measurement. When the quality level is used (instead of the bitrate), the throughput measurement is converted to the number of levels using Equation 4.2. In our implementation, we employ C4.5 [193] to generate the decision tree $\mathbf{D}$. The quality oracle then uses $\mathbf{D}$ to obtain a BIBR estimate $\widehat{\mathbb{L}}$ for a set of network performance data given by the probe kit.

Table 4.3: Input attributes and class variable for decision tree building.

Notation	Description
$\bar{d}$	Mean round-trip time, RTT (ms)
$\tilde{d}$	Median round-trip time, RTT (ms)
j_d	Delay jitter (ms)
ι_f	Forward Packet Loss Rate (%)
ι_r	Reverse Packet Loss Rate (%)
c_f	Forward Capacity (Kbps)
$\mathbb{L}$*	BIBR

Note: *: Class variable

Equation (4.2) converts throughput measurement, β, to the BIBR in quality levels, denoted by $\mathbb{L}$. A speed factor, f_{speed}, is multiplied with the average video bitrate of different levels to reduce the influence caused by the bitrate fluctuation for video clips encoded with VBR (Variable Bitrate). f_{speed} is set to 1.25, which is adopted by bandwidth throttling strategy in YouTube [22].

$$\mathbb{L} = \begin{cases} l_{min}, & \text{if } \beta \leq (b(l_{min}) \times f_{speed}), \\ l_{max}, & \text{if } \beta \geq (b(l_{max}) \times f_{speed}), \\ l_i, & \text{otherwise,} \end{cases} \tag{4.2}$$

where $b(\cdot)$ is a function to map the quality level to its video bitrate. i is an integer such that $b(l_{i-1}) \leq \beta / f_{speed} \leq b(l_i)$. l_{min} and l_{max} represent the minimum and maximum quality levels, respectively.

4.4 Evaluation

In this section, we mainly present testbed results to demonstrate how we train the decision tree in the quality oracle for profiling clients as an application of the network performance data we collected. Besides, we evaluate the accuracy of the estimated BIBR by comparing the actual streaming speed.

4.4.1 Testbed experiments

Testbed setup

We setup a testbed to generate data for decision tree building and evaluate IRate. Figure 4.10 shows the testbed topology. The web server and the IRate middlebox are directly connected. The middlebox is connected to a result database through another internal network, so that the database traffic will not interfere with the network measurement.

S_1, S_2, and S_3 are Gigabit Ethernet switches. R_1, a Linux router installed with Ubuntu 12.04 LTS, emulates different sets of network path quality with `tc`. The link capacity of R_1 is set to 1 Gbps for emulating large capacity links in the Internet core, but it often times incurs a higher delay and packet loss. R_2 is a MikroTik 750G RouterBoard, emulating a home network by limiting the bottleneck link capacity. We choose three asymmetric capacity profiles which are commonly found in ADSL or VDSL users. Moreover, R_2 is configured to use a 50-packet FIFO queue. The network path is loaded with 20% of cross traffic generated by three Linux hosts (X_1, X_2, and X_3) using D-ITG [62]. The cross-traffic packets are UDP packets, being generated according to

Pareto inter-arrivals with a shape parameter $\alpha = 1.9$ [71].

The client is installed with Ubuntu 10.10 with a Firefox browser and Flash player 11, while the web server is installed Ubuntu 12.04 LTS, Apache 2.2.22, and Adobe F4F Apache Module 4.5.1 for supporting Adobe HDS. The IRate middlebox is also a Linux server installed with the software bridge (`bridge-utils` and `brctl`) for bridging two network interfaces. In this section, the direction of forward and reverse path are referred to the uplink and downlink of the server, respectively.

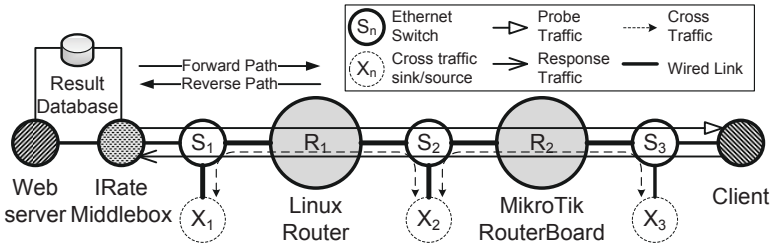


Figure 4.10: The testbed topology.

Preparing training data and the decision tree

To generate useful training data, we use a wide range of network path parameters to emulate different network connections (e.g., local access vs. access from another country). Table 4.4 summarizes the network parameters. We consider all the possible combinations of the parameters. That is, we have a total of 384 ($8\times4\times4\times3$) connection settings. For each setting, we repeatedly perform both IRate pre-stream measurement and TCP bulk download for three times. We then use these data to build a decision tree using C4.5.

For the IRate pre-stream measurement, we first let the testbed run for two seconds to allow the cross traffic to reach a steady state. We then ran the text-based probe-kit script to launch the pre-stream measurement for 60s. Three measurement flows are established between the client and the IRate middlebox. To mitigate possible effects from periodic events on the network path and the client, the second and the third mea-

Table 4.4: Network path parameters used for generating the training data.

Emulated Path Metrics	Devices	Values
Round-trip Time	R_1	4, 10, 20, 40, 70, 100, 150, 200 (ms)
Forward Packet Loss	R_1	0, 2, 4, 6 (%)
Reverse Packet Loss	R_1	0, 2, 4, 6 (%)
Bottleneck Capacity {Forward,Reverse}	R_2	{6,0.64},{8,0.8},{30,10} (Mbps)

surement flows are initiated at a random time between [0.5, 1]s after the start of first and the second measurement flow, respectively. Furthermore, we employ asymmetric IP packet sizes of 1500Bytes and 120Bytes for measuring the forward and reverse path, respectively. This packet size combination can mitigate the network congestion caused by the reverse-path bottleneck, which has slight impact on the video streaming performance.

For the bulk download, we measured the TCP throughput by initiating an HTTP bulk download of a 4.3MB file using `wget`, and `tcpdump` was run at the background to capture the traffic on the client side. The size of the file was approximately equal to a one-minute 500-Kbps video clip. Hence, the throughput of downloading that file is similar to that in streaming a short video clip. To speed up the experiments, the download tests lasted for at most 60s, which is long enough to leave the slow-start phase and capture the average TCP throughput.

We analyze the packet traces using `tshark` to extract the packet timestamps and compute the average throughput β. We then convert β to the BIBR using Equation (4.2). We adopt the five video quality levels, denoted by l_i , $i = 0, 1, 2, 3, 4$, used in our subjective assessment described in Section 3.5.2. Their bitrates are 300, 500, 1000, 1700, and 2500Kbps, respectively. Two additional levels $i = -1, 5$ are introduced to label the cases having a throughput much lower than the lowest and higher than the highest bitrates. Therefore, the number of samples in each level is more evenly distributed, which can alleviate the data overfitting problem.

We use C4.5 classifier in Weka [91] to generate a decision tree with the inputs of the network path parameters and β. We set the confidence factor to 0.25 for tree pruning. To prevent outliers increasing the height of the tree, we also require the number of records at each leaf to be at least 10% of the total number of test records. The resultant decision determines the BIBR mainly by the median RTT and forward packet loss rate.

Besides, measuring the asymmetric packet loss rate is very important for estimating the BIBR. As the forward loss rate has to be inspected in all the cases in the decision tree, while only four cases need the reverse loss rate. This is because the forward path is used for streaming the video data, but the reverse path is for TCP ACKs, which have only a slight impact on the throughput. If only round-trip loss rate is used, the BIBR will be under-estimated when the actual packet loss rates are asymmetric.

Accuracy of estimating the BIBR

We next use the 60-second training data to evaluate IRate's accuracy of estimating the BIBR. We slice the 60-second measurement data to shorter measurement durations from the beginning of the measurement to the required duration for emulating a shorter time window for pre-stream measurement. If the predicted quality level is one of the two additional levels $i = \{-1, 5\}$, it is mapped to the quality levels $\{0, 4\}$, respectively. Figure 4.11 shows the accuracy of IRate's prediction across the entire pre-stream time window, from 1s to 30s. We only show the accuracy up to 30s, because we find that the accuracy is converged after 30s. We compute $\Delta\mathbb{L} = \widehat{\mathbb{L}} - \mathbb{L}$, where $\mathbb{L}$ is the actual BIBR computed from the throughput measurement. The green (bottom), white (middle), and red (top) portions of the bars show the percentage of samples that IRate has under-, correctly-, and over-estimated the BIBR (i.e., $\Delta\mathbb{L} < 0$, $\Delta\mathbb{L} = 0$, and $\Delta\mathbb{L} > 0$), respectively.

The white portions indicating the correct estimation increase from 45.9% to 86.8%

as the pre-stream time window grows from 1s to 30s, because a larger time window can allow the probe kit to obtain more results and mitigate the effect of short-term fluctuations. Although about 4.8% of the samples under-estimates the BIBR by one level, the video playback under these cases can still play smoothly without any rebuffering. By also considering these cases as acceptable, the accuracy is above 80% for a pre-stream time window of 10s.

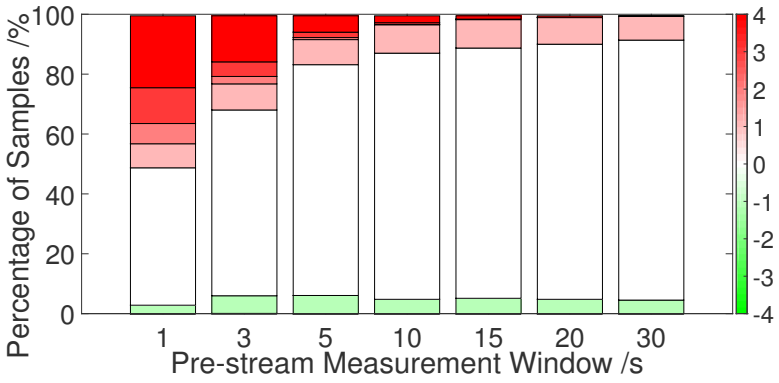


Figure 4.11: The accuracy of IRate's estimation of the BIBRs using different pre-stream time windows.

Video streaming performance with IRate

This set of experiments is to illustrate the benefit of IRate by comparing the streaming performance against the usage of BIBR to HTTP streaming under an unknown network environment. We have integrated IRate into a small video streaming system and run it on the same testbed. The client in the testbed streams video clips from the server with a customized Flash video player, which is modified from the *Strobe Media Playback* for supporting the similar functions as *FlashTrack* [165] for both HTTP streaming and DASH. The player can report application layer information, such as rebuffering events, the number of bytes downloaded. Particularly for DASH, the BIBR chosen by IRate is only effective to the first video segment. After that, the same quality adaption algorithm will take over the bitrate adaption process.

To emulate a more realistic environment to evaluate the performance of IRate, we generated a set of 200 samples by randomly choosing an RTT from the range of [4,150]ms, forward and reverse packet loss rates from [0,6]%, and one of the bandwidth profile in Table 4.4. For each set of metrics, the client ran the probe-kit script for 10s to determine the BIBR. Besides, the test video clip is a 60-second sports video, which is one of the test video clips used in the subjective assessment.

A Firefox browser was run on the client to load the video page and play the video using HTTP streaming with IRate, HTTP streaming with predefined initial bitrate. We assume that the predefined initial bitrate scheme uses quality level l_2. To speed up the experiment, we only allow the video to be played for 90s. Moreover, HTTP streaming with predefined initial bitrate and DASH without IRate will not be tested if the predicted BIBR is l_2 or l_0, respectively, because the predicted BIBR is the same as the default value.

The accuracy for IRate's estimation of the BIBR under randomly selected network metrics is about 75%. Furthermore, we quantify the improvement in the video streaming performance using IRate. We analyze the log captured by FlashTrack using the application performance metrics proposed in [165]: Initial buffering time (T_{init}), rebuffering frequency (f_{rebuf}), and mean rebuffering duration (T_{rebuf}). Figures 4.12(a), 4.12(b), and 4.12(c) plot the CDFs of T_{init}, f_{rebuf}, and T_{rebuf} for HTTP streaming and DASH, with and without IRate, respectively.

In our analysis, we exclude the cases that the estimated BIBR is equal to the predefined bitrate of the streaming method (i.e., l_2 for HTTP streaming and l_0 for DASH), because both methods are expected to have the same performance. There are $\{19\%$, $12.5\%\}$ of cases that have the estimated BIBR of $\{l_0, l_2\}$. Figure 4.12(a) shows that IRate reduces the initial buffering time in HTTP streaming. About 80% of HTTP streaming with IRate can start playing the video within 2s, which is 6.3% higher than

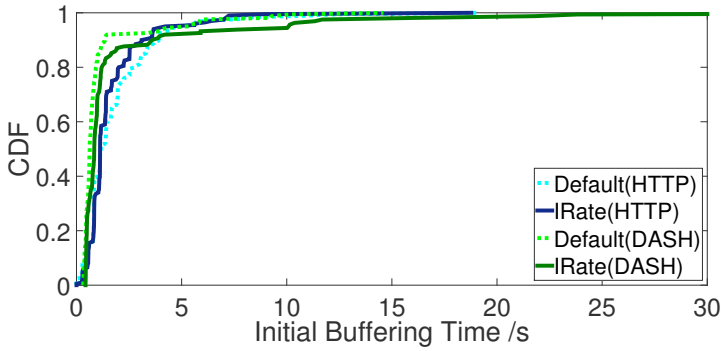
that of without IRate. The native DASH has a shorter T_{init}, because it always starts with the lowest quality. Although DASH needs more time for initial buffering when the predicted BIBR is used, the absolute time is still very short. T_{init} is less than 4s for 90% of the cases, but it also shows a long tail for over-estimated cases.

Rebuffering frequency quantifies how often rebuffering events occur during the video playback. The smaller the value means the playback is smoother, giving a better QoE [165, 70]. $f_{rebuf} = 0$ means no rebuffering throughout the playback. Figure 4.12(b) shows that 88% of the HTTP streaming (with IRate) cases encounter no rebuffering events, which is 25.7% more than HTTP streaming (with predefined quality). On the other hand, DASH greatly reduces rebuffering events by adapting the video bitrate, and the performance for DASH with IRate and the native DASH is comparable.

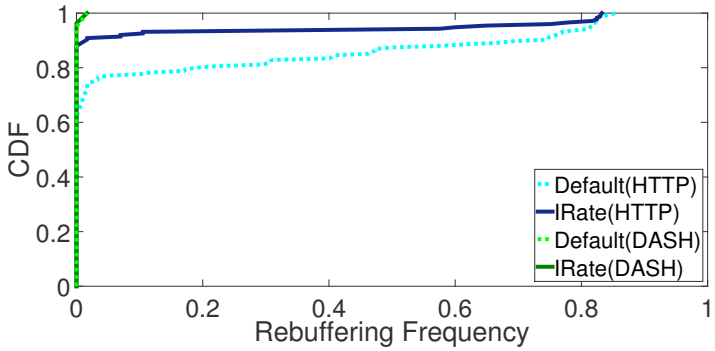
We average the rebuffering duration of all the rebuffering events of each video playback. Figure 4.12(c) plots the distributions of T_{rebuf} for those cases with $f_{rebuf} > 0$ in Figure 4.12(b). Using IRate, the median of T_{rebuf} is similar in HTTP streaming. However, some cases in HTTP streaming show higher mean rebuffering duration than the default, which can be due to over-estimated BIBR. Similarly, the performance of DASH for both default and IRate is generally comparable. However, we hesitate to draw any general conclusion as the rebuffering events are rare in our dataset.

DASH Stability and Efficiency

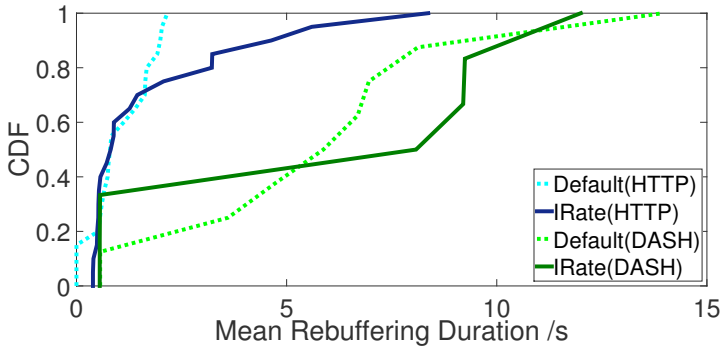
DASH stability refers to how frequent the bitrate changes during the video playback. Previous studies showed that unstable bitrate can hurt the QoE [179]. We define a stability metric, denoted by τ, in Equation (4.3) to quantify the improvement of DASH's bitrate stability by IRate. This metric is similar to the stability metric in [118]. However, we consider only the first seven video chunks ($\sim$28s) of the video playback which



(a) Initial buffering time.



(b) Rebuffering frequency.



(c) Mean rebuffering duration.

Figure 4.12: The CDFs of application performance metrics.

are more relevant to the choice of initial bitrate.

$$\tau = \frac{\sum_{d=2}^{d\leq 7} |b(l_{d-1}) - b(l_d)|}{\sum_{d=1}^{d\leq 7} b(l_d)}, \tag{4.3}$$

where l_d is the bitrate level used for streaming d^{th} video chunk.

Figure 4.13(a) shows the CDF of DASH stability for the predefined (lowest) quality

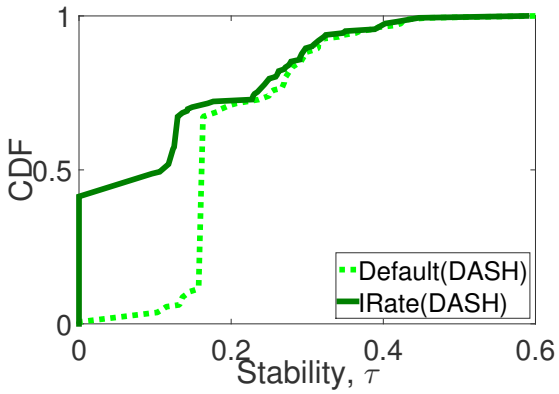
strategy and IRate. Nearly half of the cases in IRate has no bitrate switching (i.e., $\tau = 0$) in the first seven video chunks. Since we do not show the cases with BIBR equals to l_0, all the cases in the native DASH switch the bitrate at the beginning (i.e., $\tau > 0$). This shows that selecting the BIBR helps improve the stability by reducing quality up-switching at the beginning of video.

DASH efficiency, denoted as ϵ, quantifies the effectiveness of DASH on utilizing the network resources for video streaming. In the ideal case, the BIBR is equal to the network throughput. In other words, the time to download a video chunk is equal to the length of video it contains. In [118], they proposed to use the video bitrate and the average throughput to compute the efficiency. However, these two metrics cannot accommodate the video encoded with the variable bitrate (VBR). Hence, we propose to use the video chunk download time and the length of video (in seconds) contained in a video chunk, denoted by g, to compute the efficiency:

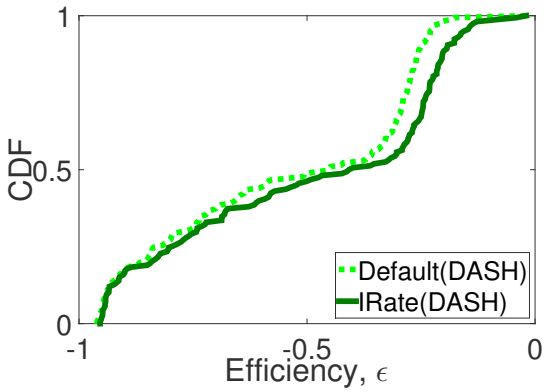
$$\epsilon = \frac{\sum_{i=1}^{i \leq \gamma} \frac{\omega_i - g}{g}}{\gamma}, \tag{4.4}$$

where γ is the total number of downloaded video chunks, and ω_i is the time spent on downloading i^{th} video chunk.

If DASH is completely efficient, the download time of a video chunk will be equal to the number of seconds of video contained in the chunk, (i.e., $\epsilon = 0$). When DASH cannot completely utilize the network bandwidth, less time is used to download the same video chunk, resulting in $\epsilon < 0$. Figure 4.13(b) shows the CDF of the efficiency metric. For $\epsilon \leq 0$, DASH with IRate obtains a closer-to-zero value, meaning a higher efficiency. The DASH with IRate has a median value 36% larger than the predefined quality case. This is because DASH with IRate can avoid ramping up from the lowest bitrate, therefore better utilizing the network bandwidth.



(a) DASH stability.



(b) DASH efficiency.

Figure 4.13: The CDFs of DASH efficiency and stability metrics.

4.5 User QoE experiments

To further compare the performance in terms of QoE, we conducted subjective assessments similar to the one described in §3.5.2. The main difference is that the video quality levels in this set of experiments are adapted according to the real Internet performance, instead of an emulated transition scheme. The goal of this assessment is to compare the difference in QoE between the default (lowest) initial quality and the BIBR estimated by IRate.

In this assessment, we create the pre-stream time window by requiring the subjects to fill a short survey, because we do not have considerable amount of content for subjects to choose from to generate the user think time. The pre-stream measurement was

conducted at the background to measure the network path quality between the clients and the IRate-enabled video server connected to our campus network until the completion of the survey. After the survey, each subject was required to watch two video clips randomly chosen from the four video clips used in §3.5.2. The approach to be used (default or IRate) to select the initial bitrate in the video clip was randomized and did not inform to the subjects. After that, the video player adapts the video quality according to the network throughput.

We have successfully collected the results from 22 volunteers. Instead of inviting them to the laboratory, we delivered the assessment site through email. Therefore, the subjects can stream the video clips using the real Internet networks. Table 4.5 shows the network types of the subjects according to the IP records. Most of the subjects accessed the experiment using local residential broadband network. One of the participants is from the US. The predicted BIBRs from IRate showed that one of local subject's network quality cannot support the highest quality level. The player logs recorded only one rebuffering event in one of the playback started with the default approach. We cannot observe any rebuffering event for all video playbacks used IRate. Therefore, the accuracy of IRate is 100%.

Table 4.5: Distribution of network types.

No. of volunteers	Network type	Location
10	Residential broadband (HKBN)	
6	Residential broadband (PCCW)	
2	University dormitory WiFi	Hong Kong
2	University campus network	
1	Residential broadband (Hutchison)	
1	Overseas academic network	US

Similar to the study presented in §3.5.2, we compare the MOS, $\overline{E_{init}}$, and $\overline{E_{overall}}$ between the video clips started with the estimated BIBR and the predefined quality level. We find that the MOS is increased from 3.82 to 4.09 (6.67%) when the estimated

BIBR from IRate is applied. Furthermore, IRate's average rating on the initial video quality $\overline{E_{init}}$ and the overall picture quality $\overline{E_{overall}}$ are higher by 24.4% and 11.8%, respectively. The differences in rating for $\overline{E_{init}}$ and $\overline{E_{overall}}$ are significant (p <0.05) in two-sample *t*-test.

4.6 Discussion

In this section, we discuss the limitations and issues of IRate.

4.6.1 Accessing videos from third-party video sites

In some circumstances, video pages are accessed directly without first visiting the video website. Users could be redirected from 1) embedded video objects in third party websites, 2) sharing forums in social networks, or 3) recommendations in search engines. IRate can still be used for the first case. The embedded video objects provided by the video sites for sharing are implemented as an `iframe`, which then loads a small page from the video website. In this case, the probe-kit script can be inserted to the page and executed by users. For the second and third cases, pre-stream measurement cannot be carried out as the social networks or the search engines only provide a link to the video website. Therefore, the probe-kit script cannot be inserted. Without any reliable measurement results, IRate can simply fallback to the default bitrate scheme. A better solution requires the co-operation between these websites and the video service provider in that the probe-kit script can also be inserted into the web pages of these websites and executed when the web pages are rendered. Another solution is to perform the measurement during the pre-roll advertisement, but more cross traffic can be incurred by the download of the Ad stream.

4.6.2 Scalability and security issues of IRate

The server-side design may suffer from scalability issue for large-scale video websites. Our current IRate prototype measures every client accessing the video website. However, measuring all clients may not be necessary. As the network condition may be stable within a short time [188, 248] and the average throughput can be better known after the first video is watched, the web server could also utilize these historical data to improve the accuracy of the BIBR. Moreover, IRate can reduce the measurement traffic by closing the measurement flows of some clients for which sufficient data have been collected for estimating their BIBRs. The IRate middlebox can send messages to the probe-kit script, so that the script will not establish any new measurement flow.

We believe that the IRate middlebox is not vulnerable to Distributed Denial-of-Service (DDoS) attacks, because the IRate middlebox is IP-less, therefore transparent to clients. Moreover, the middlebox only handles the TCP connections successfully established by web server. Malicious clients can trigger the middlebox to recognize the flow as a measurement flow by injecting the specially crafted HTTP requests in the network flow. However, the middlebox can close the existing connections and refuse any new measurement connections from the clients whenever the middlebox collects enough data or maintains sufficient measurement flows to the same client.

4.6.3 Short vs. long video clips

IRate is obviously most beneficial for short video clips, which are still very popular in today's Internet. On the other hand, for long videos or movies, the benefit to the overall QoE rating will be decreased as users may forget the experience at the beginning [191]. In a recent study, Staelens et al. [220] evaluated the QoE of long videos on tablets using SSCQE [110], which continuously measures the QoE. Their results showed that large-range bitrate switchings at the beginning of the video, which can be mitigated

by IRate, have significant impact to the QoE instantaneously. Further investigation on quantifying the effect of the video length will be our future work.

4.7 Summary

In this chapter, we presented a server-side pre-stream measurement system, IRate, which is compatible with existing video delivery infrastructure. IRate exploits the pre-stream time window to perform active measurement to the actual video cache. The performance data collected from IRate could help profile the client by estimating the best initial bitrate (BIBR) for HTTP streaming. Our testbed results showed that IRate can acquire enough path quality data for estimating the BIBR with 80% accuracy in 10s, and it can help improve the QoE of HTTP streaming. Our user experiment further validated that IRate can improve the QoE by more than 6% in the actual Internet environment.

Chapter 5

Towards an Accurate Bandwidth Estimate for DASH Video Streams

In Chapter 4, we showed that `IRate` can provide a better decision of the initial video bitrate. After the video playback is started, the quality adaptation algorithm decides the quality levels to be downloaded. This chapter proposes a QoE-aware DASH system, named `QDASH`, which aims at providing a practical solution to improve the QoE for the current DASH systems by 1) enhancing the mid-stream throughput measurement and 2) mitigating the impact on the QoE when network degradation occurs. The video service providers do not need to re-encode existing video clips and do not require to install extra softwares in the server. `QDASH` composes of two modules—`QDASH-abw` and `QDASH-qoe`.

`QDASH-abw` is a novel probing methodology which embeds available bandwidth measurement into video flows to detect the highest quality level the current network conditions can support. Similar to `IRate`, this module is implemented as a measurement middlebox, which is placed in front of the video server. Therefore, it can manipulate the packet sending patterns and headers in video data flows. Instead of applying flooding-based throughput measurement, `QDASH-abw` measures the available

bandwidth using Pathload [115].

There are two main challenges in measuring the available bandwidth. First, the original Pathload algorithm uses binary search to seek for the exact sending rate which results in inflated RTT. This method can be insufficient for DASH. Another challenge is that normal TCP flows do not have enough RTT samples due to TCP delayed acknowledgement mechanism [33].

We tackle the first challenge by only considering a few discrete sending rates which match with the bitrate of video quality levels to speed up the convergence of available bandwidth estimates. The second problem is alleviated by exploiting the packet sending sequence. In particular, we reorder the packets to trigger TCP pure ACKs to increase the number of RTT samples.

With the available bandwidth estimates, `QDASH-qoe` is responsible for helping clients to select the most suitable video quality level. In particular, we focus on switching down of video quality which can degrade the QoE the most. In Section 3.5.1, our subjective study investigated different approaches of switching down the quality levels under the same network conditions. Our results show that inserting intermediate levels provides a better QoE than directly switching to the target quality level. With this finding, we propose a QoE-aware adaption algorithm to utilize the video buffer and select a suitable quality levels.

The overview of `QDASH` system will be described in Section 5.1. In Section 5.2, we will illustrate the methodology of `QDASH-abw` in details. Section 5.3 describes `QDASH-qoe` which utilizes the findings in our subjective assessment to mitigate the QoE degradation caused by changes in quality level.

5.1 Overview of QDASH

QDASH consists of two building blocks—QDASH-abw and QDASH-qoe. QDASH-abw measures the network available bandwidth, and QDASH-qoe determines the video quality levels. These two modules can be integrated into existing DASH systems, while the modifications to the systems are kept to minimum.

QDASH is designed for streaming H.264/AVC video clips, and aims at immediate deployment to current systems. The main shortcoming for using H.264/AVC in DASH is that the storage overhead is large for multiple copies of video for different quality levels. To reduce the overhead, researchers recently propose employing H.264/SVC, which encodes video clip into enhancement layers [212], to improve the efficiency. However, billions of existing video clips have already been encoded into multiple bitrates with H.264/AVC codec. They only need to insert meta data in order to enable DASH, while H.264/SVC solution requires video re-encoding which is computationally expensive. Therefore, the proposed architecture is targeted for DASH systems using H.264/AVC.

Figure 5.1 shows the overall QDASH's architecture. A measurement middlebox, which is equipped with the QDASH-abw, is directly connected to the video server. Therefore, the middlebox can inspect and intercept the video data flows. It shapes the packet sending rate according to the bitrates of video quality levels and sends packets according to the QDASH-abw probing method (*cf.* Section 5.2). The middlebox is an IP-less device, which does not require an IP address and is transparent to both clients and servers.

Employing the measurement middlebox paradigm reduces the overhead of measurement. The middlebox manipulates real video data packets to perform *inline measurement*. Extra probing packets, such as the RTT test in Akamai's video streaming [63], are not required. Furthermore, by coupling the measurement flow with the video

data flow, we can avoid using additional measurement flows, which may not traverse the same path as the data flow because of load balancers on the path. Hence, measured performance can be obtained with higher accuracy and less overhead.

With the QDASH middlebox, the server-side is able to measure their clients without installing additional softwares or requiring root privilege at either side for lower levels' information. The measurement tasks are offloaded to the measurement proxy and do not induce additional loading to the video server. The server-side only needs to modify the video player at the application level. The client-side also does not need to install softwares, such as libpcap [7], to cooperate with the measurement [151].

At the client-side, we proposed a QoE-aware quality adaptation algorithm—QDASH-qoe (*cf.* Section 5.3). QDASH-qoe incorporates the finding in Section 3.5.1 to adjust the video quality. It can be implemented in the video player delivered to the clients' browser. On the other hand, QDASH enables the video player to establish a lightweight flow to receive updates about the measurement results measured by QDASH-abw. At the same time, this flow can report application level events or user-viewing activities to help inferring the QoE as we will discuss in Section 6.

The video delivery procedure for QDASH is described as follow. After the establishment of TCP connection and the download of MPD files, the client first sends an HTTP request to the video server to initiate the video streaming. The video server transmits the HTTP responses with the requested video segments. The measurement middlebox then hijacks the data flows and transmits the data packets according to QDASH-abw probing methodology. Hence, the RTT can be measured using the TCP acknowledgement packets triggered by the re-ordered data packets. At the same time, the client starts playing the video once the video buffer is full and connects to the measurement middlebox for the latest measurement results. QDASH-qoe can choose the most suitable quality level according to the measurement results.

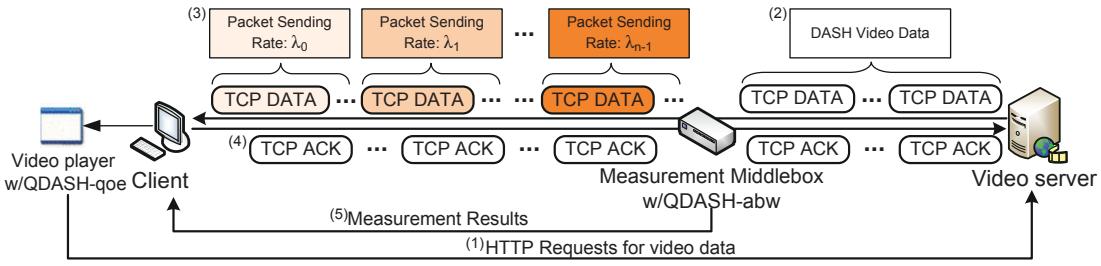


Figure 5.1: The overall QDASH architecture.

5.2 Measuring available bandwidth for DASH

Instead of simple throughput measurement, we employ available bandwidth measurement to assist clients to suitably select the video quality levels. Available bandwidth measurement methods, such as Pathload [115], aim at acquiring accurate estimates by varying the packet size or the packet sending rate. However, these tools need tens of second to obtain one estimate, and this time is too long for DASH to adjust the quality.

In fact, video quality levels have a limited number of values. To determine whether the current available bandwidth is higher than any of the video quality levels, a high-resolution estimate is not required. Instead, timely updates of estimates are more important for correctly selecting or withdrawing the quality levels. We therefore propose a quantization approach to reduce the number of probes and speed up the convergence of results.

5.2.1 Assumptions

We make the following assumptions for the available bandwidth measurement methodology:

1. The average bit rates of video quality levels are known.
2. The available bandwidth between the server and the client is always higher than or equal to the least video bitrate (the lowest video quality level).
3. The client has sufficient computational power to render all the video quality

levels.

These assumptions are realistic. The service providers are often responsible for encoding video clips. They can therefore easily obtain the video quality level information from the encoding profiles. Our probing methodology (*cf.* Section 5.2.2) does not test the available bandwidth lower than the least video bitrate. Moreover, DASH cannot help improve the scenario if the client consistently has low available bandwidth. In this chapter, we are only interested in the change in quality levels for adapting to the network conditions, instead of other factors, such as clients' computational power and system loading. Therefore, we assume all the clients can smoothly render the highest quality video clips.

5.2.2 QDASH-abw probing methodology

The basic idea of our measurement method comes from the observation that if the packet sending rate λ is higher than the available bandwidth A, the mean and the variance of the packets' RTTs will be larger than the values obtained when the packet sending rate is less than the available bandwidth [239]. Although the basic idea is simple, there are three challenging issues to be tackled:

1. How to generate the probing packets for measurement?
2. How to collect the measurement samples?
3. How to determine the acceptable sending rate?

These issues are closely related to the design of `QDASH` and we will elaborate on our solutions below.

To the best of our knowledge, existing tools for available bandwidth measurement use customized probing packets. Part of the available bandwidth is therefore consumed by the measurement. In contrast, we propose using inline measurement that employs the media data packets directly to determine whether a certain sending rate is supported

by the current available bandwidth. The rational behind such design is three-fold. First, measurement results from an additional TCP connection may not be accurate because the new TCP connection may not traverse the same network path as the media data because of the load balancing. Second, the additional measurement packets can compete the bandwidth with video flows and degrade the streaming performance. Third, our measurement methodology can throttle the bursty video traffic, which can induce packet loss in large-scale video systems [85].

A probing round is defined as a sequence of packets sending at the same rate from the measurement middlebox. For each probing round, the server side (measurement middlebox) elicits a packet train with K pairs of W-byte probe packets at a sending rate λ. We denote the TCP data packets sent from the video server by $Sa|b$ and the response packets from the client by $Ca|b$. a and b are the data segments' sequence number and acknowledgement number, respectively. Instead of showing the exact TCP sequence and acknowledgement number, we simply use $a = 1, 2, 3, ...$ to label server's TCP data segments and $b = 1', 2', 3', ...$ client's data segments.

Figure 5.2 illustrates an example of two probing rounds with sending rates (Kbps) λ_0 and λ_1, and the available bandwidth is between λ_0 and λ_1. We assume that the TCP connection between the client and the video server is established, and the server receives an `HTTP GET` request from the client for the first video segment. The response packets do not send to the client directly, and are intercepted by the measurement middlebox. At the slow start phase of TCP connection, the sending window in the video server is small. The number of on-the-flight respond packets is not enough for a probing round. Hence, the measurement middlebox needs to send pure TCP ACKs to the video server for fetching more video data packets. In each probing round, the measurement middlebox buffers $2K$ data packets, $S1, ..., S2K$. Then, it sends a TCP ACK with zero receive window to suppress the server from sending out more

video data packets and overflow the middlebox. After a new probing round starts, the middlebox re-opens the sending window of the video server by sending a duplicated TCP ACK with non-zero window size. In our implementation, we set the window size to two times of the Maximum Segment Size (MSS).

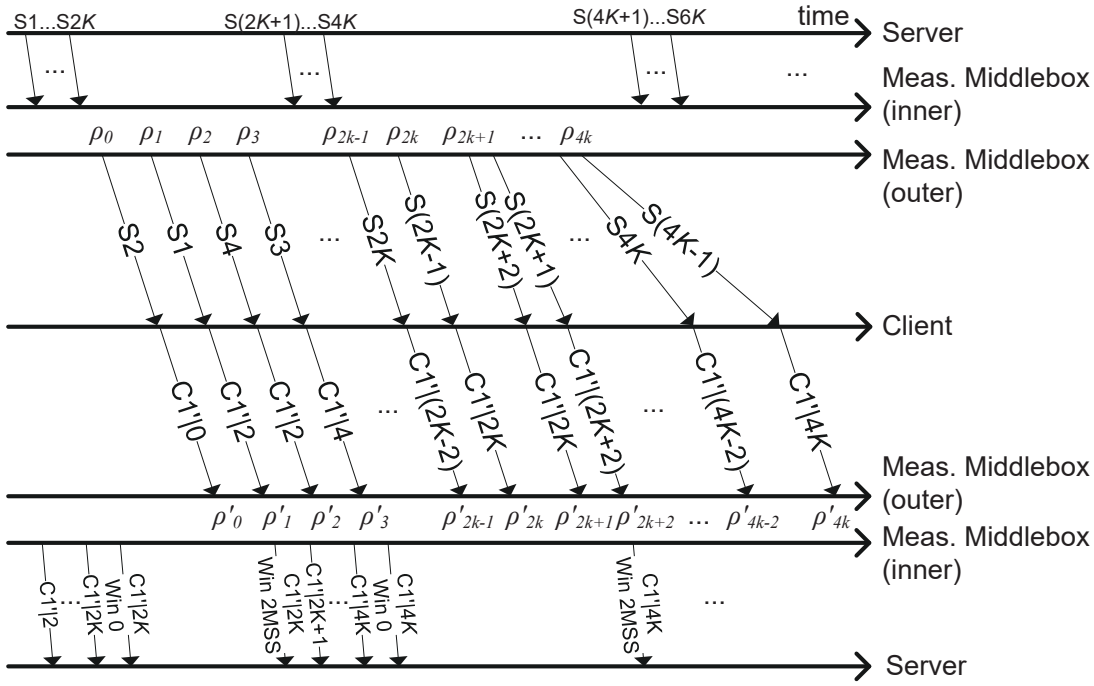


Figure 5.2: Two probe rounds of sending rates.

Existing tools for available bandwidth measurement usually adopt the binary search technique for the sake of determining the accurate available bandwidth in a short period [239]. However, the time required is still too long for seeking a suitable video quality level. In fact, it is not necessary to probe the network with all possible sending rates for deciding whether a video quality level can be smoothly played by the client under the current network condition. Instead, we propose probing the network with a set of selected sending rates that correspond with the bitrate of quality levels. By doing so, we can quickly know whether a sending rate is supported by the network through the obtained RTT samples.

We adjust the packet sending rate by varying the inter-departure time (IDT) of packets in each probing round. The IDT γ_k of packet sending rate λ_k is computed by

$$\gamma_k \quad = \quad (W \times 8)/\lambda_k, k = 0, 1, ... \tag{5.1}$$

The RTT samples can be computed as the duration from sending a TCP data packet to receiving a corresponding acknowledgement packet (i.e., TCP ACK). However, the TCP delayed acknowledgement mechanism [33] may bias the results because it allows the receiving side to acknowledge multiple data packets with a single TCP ACK. To solve this problem, the measurement middlebox reorders the data packets by pairs, so that clients send an ACK packet for every TCP data packet. After the reordering, the packet sending sequence for the first probing round is $\{S2$, $S1$, $S4$, $S3, ..., S2K, S(2K-1)\}$ with the sending time $\{\rho_0, \rho_1, ..., \rho_{2k-1}\}$. Assume there is no packet loss, the response packets for the first probing round are $\{C1'|0$, $C1'|2$, $C1'|2$, $C1'|4, ..., C1'|(2K-2), C1'|2K\}$ and arrive the measurement middlebox at time $\{\rho'_0, \rho'_1, ..., \rho'_{2k-1}\}$, respectively. Hence, the RTT estimates are retrieved by $\{\rho'_0 - \rho_0, \rho'_1 - \rho_1, ...\}$.

5.2.3 Determining the video quality levels

Before conducting the measurement, we can first measure the network path quality and predict the BIBR using IRate (*cf.* Chapter 4). After that, we use the sending rate that is equal to the BIBR to do a round of measurement and calculate the loss rate (i.e., U_{max}), the average RTT (i.e., T_{max}), and the variance of RTT (i.e., V_{max}). Since the available bandwidth should be less than or equal to the capacity, the values $\{U_{max}$, T_{max}, and $V_{max}\}$ serve as the upper bound of these metrics. Similarly, we use the minimal sending rate to perform another round of measurement and compute the loss rate (i.e., U_{min}), mean RTT (i.e., T_{min}), and variance (i.e., V_{min}). Since we assume that

the available bandwidth is larger than the minimal sending rate, these values $\{U_{min}$, T_{min}, and $V_{min}\}$ are the lower bound of the metrics.

When the sending rate is higher than the available bandwidth, the probability of packet loss increases [129]. An acceptable sending rate leads to a smaller packet loss rate than U_{max}. Using a packet sending rate higher than the available bandwidth will inflate the mean RTT and the RTT variance, both of which severely affect the average throughput of network path. In contrast, a lower RTT and RTT variance can be observed when a lower packet sending rate is used. The performance will be acceptable to users. Therefore, we adopt a conservative approach that bounds the mean RTT and the RTT variance to determine whether a sending rate is acceptable. More precisely, besides the loss rate should be smaller than U_{max}, an acceptable sending rate should result in a mean RTT less than $T_{min}+a\times V_{min}$ and a variance less than $b\times V_{max}+c\times V_{min}$. In our experiments, we set $a = 3$ and $b = c = 1/2$. The highest acceptable packet sending rate will be the *preferred rate*.

After each round of measurement, the measurement middlebox informs the latest preferred rate to the client for quality adaptation. If the client selects a quality level with bit rate lower than the preferred rate, video rebuffering is unlikely to occur. However, sometimes, we choose quality level with bit rate higher than the preferred rate for a short period of time. We will elaborate our quality adaptation algorithm, `QDASH-qoe`, in section 5.3.

5.2.4 Evaluating QDASH-abw

Experiment setup

We have implemented a prototype of measurement middlebox using a Linux bridge. The `QDASH-abw` module hooks to the bridge with `NFQUEUE` target in `iptables` [4]. This design enables the `QDASH-abw` module to capture, discard, or manipulate all the

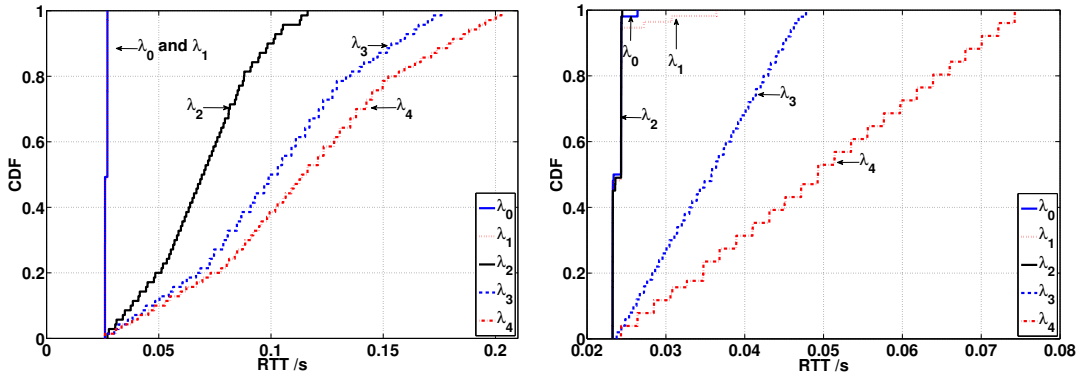
packets flowing between the video server and the clients. We inject a TCP MSS option with value W in the TCP SYN packet such that the server sends W-byte video data packets [42]. Once the `QDASH-abw` module recognized an `HTTP GET`, it buffers and resends video data packets according to the probing methodology described in Section 5.2.2.

In the following evaluation, we adopted a dumbbell client-server architecture. A click modular router [131] was placed between the video server and the client to control the available bandwidth. The packet size W and the packet train length, K were set to 576bytes and 25, respectively. The packet sending rates, $\lambda \in \{\lambda_0, \lambda_1, \lambda_2, \lambda_3, \lambda_4\}$, were respectively selected as $\{$0.3, 0.7, 1.5, 2.5, 3.5$\}$Mbps which are the bitrate of video quality levels adopted by Akamai Adaptive Video Streaming [63]. The client uses `wget` to send `HTTP GET` requests to download an MP4 video file from the server. For better illustrating the variations of RTT in different packet sending rate during the whole experiments, the sending rates were scheduled in a round-robin manner.

Experiment results

Figure 5.3 shows the CDFs of sampled RTTs under different available bandwidth. In Figure 5.3(a), the available bandwidth is 1Mbps. When λ_0 and λ_1 are used to measure the available bandwidth, the RTTs are similar and do not have large variance compared to the results from other sending rates. The reason is that λ_0 and λ_1 are less than the available bandwidth and the packets do not suffer from a long queuing delay. On the other hand, when using λ_2 to λ_4, we can observe much longer RTTs with larger variance. The reason is that the bottleneck needs much more time to process the packets and the queueing delay increase. In Figure 5.3(b), the available bandwidth is 2Mbps and we can observe similar results as Figure 5.3(a). The only difference is that since λ_2 is less than 2Mbps the corresponding RTTs are similar to those resulted from λ_0

and λ_1.



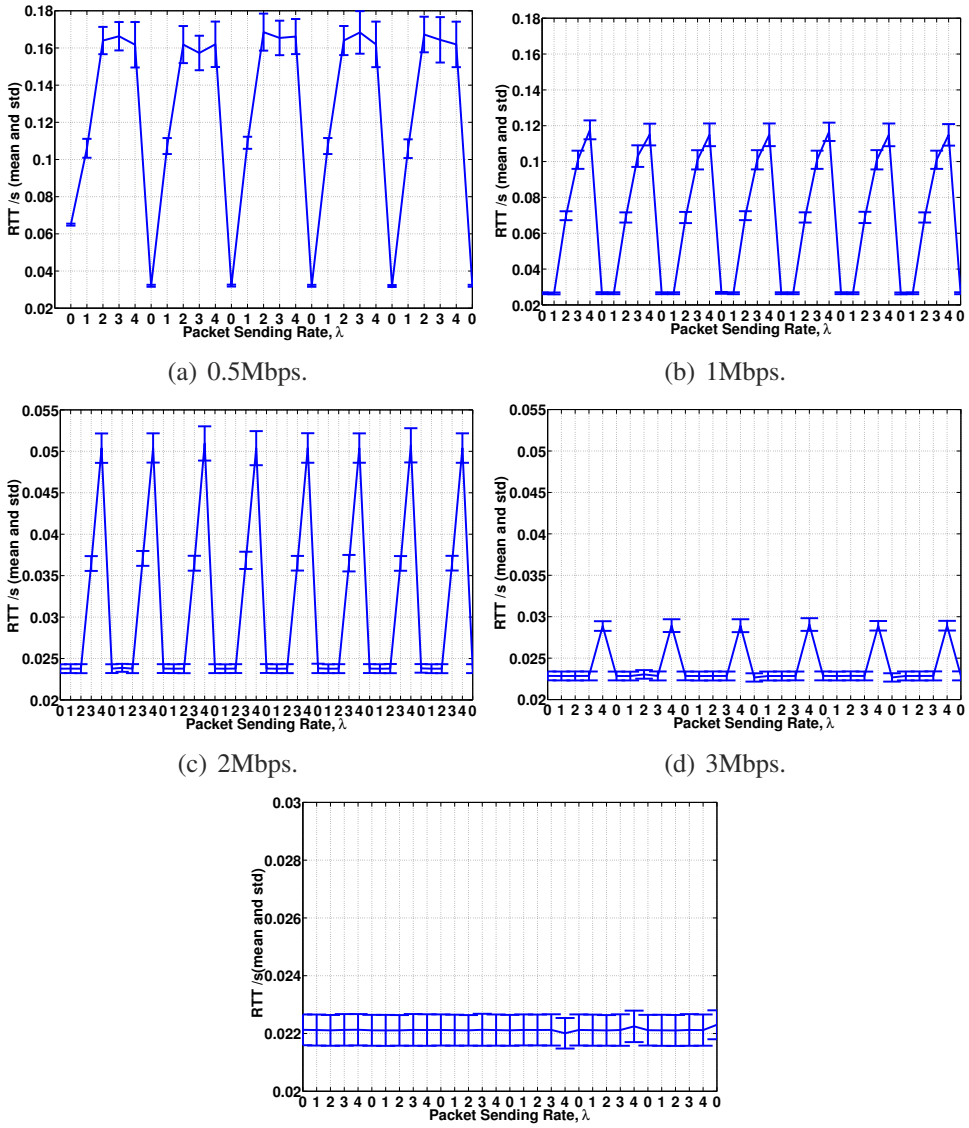
(a) 1Mbps (b) 2Mbps

Figure 5.3: CDF of sampled RTTs under available bandwidth of 1Mbps and 2Mbps.

Figure 5.4 illustrates the mean and the variance (expressed in standard deviations, std) of the sampled RTTs with different packet sending rates λ and available bandwidth. The number on the x-axis, $\{0, ..., 4\}$, represents the packet sending rate, $\{\lambda_0, ..., \lambda_4\}$, respectively. For each available bandwidth, our system conducts the measurement with different packet sending rates. The x-axis indicates the sending rate (from λ_0 to λ_4) and the y-axis is the mean RTTs. We plot the standard deviations (std) of the RTTs along with their mean values.

In Figure 5.4(a), since the available bandwidth is 0.5Mbps, all except λ_0 resulted in high average RTTs with large std. When the available bandwidth increases to 1Mbps as shown in Figure 5.4(b), the average RTTs and the corresponding std resulted from λ_2 to λ_4 are still larger than that of λ_0 and λ_1 because λ_2 to λ_4 are larger than the available bandwidth. However, compared to Figure 5.4(a), the average RTT and the std for λ_2 to λ_4 decrease. In Figure 5.4(c) and 5.4(d), the available bandwidth becomes 2Mbps and 3Mbps, respectively. From these two figures, we can still observe that the rate(s) higher than the available bandwidth led to larger mean RTT and std than the rates less than the available bandwidth. However, such difference decreases with the increase of the available bandwidth. In Figure 5.4(e), the available bandwidth is 5Mbps larger

than all packet sending rates. Therefore, the average RTTs and the std from different sampling rate are similar.



(a) 0.5Mbps. (b) 1Mbps.

(c) 2Mbps. (d) 3Mbps.

(e) 5Mbps.

Figure 5.4: The mean and the standard deviation of sampled RTTs with different sampling rate and different available bandwidth.

We adopted available bandwidth changing profiles in [19] to simulate changes in network conditions and examine the behaviors of `QDASH-abw`. Figure 5.5 shows the time series of mean RTT of different packet sending rates in four profiles—persistent variations, short-term variations with positive spikes, short-term variations with neg-

ative spikes, and persistent variations with stepwise increases. The dash-dotted lines are the mean RTTs measured with different packet sending rates. The grey dashed line in the figures plots the emulated network available bandwidth for reference. We also measure the throughput by continuously fetching a video segment with a size of 949KB by using `wget`. We capture the packets on the client side and sample the TCP throughput for every 0.5s using `tshark`. The solid line in the figures shows the harmonic mean of the last 20 throughput estimates as the average throughput, which is used in some quality adaptation algorithms to smooth out throughput fluctuations (*cf.* Section 2.4.1).

Figure 5.5(a) plots the results and the emulated available bandwidth for persistent variations. We emulate this condition by sequentially limit the available bandwidth to 5Mbps for 20 seconds, 1Mbps for 20 seconds, 5Mbps for 30 seconds, and finally 2Mbps until the experiment ends. For the first 20 seconds, the available bandwidth is sufficient for all the sending rates. The RTTs are close to the emulated delay (20ms). During the period of 20 to 40s, the sending rates λ_2 to λ_4 are higher than the available bandwidth. The RTTs for these three sending rates inflate at least 3 times than the RTT in the previous period. Then, the RTTs for all the sending rates fall back to around 20ms after the available bandwidth restored to 5Mbps. After 70 seconds, the available bandwidth is 2Mbps, which is higher than λ_0 to λ_2, but lower than that of λ_3 to λ_4. Hence, we only observe inflated RTTs for λ_3 and λ_4. In contrast, the average throughput converges to the available bandwidth in 10s after the changes of bandwidth.

Figure 5.5(b) shows a case of short-term variations with positive spikes. The available bandwidth for most of the time in the experiment is 1Mbps, except there is a 2-second spike to 5Mbps and a 5-second spike to 10Mbps at time 30s and 62s, respectively. The RTTs for λ_2 to λ_4 are significantly higher than that of λ_0 and λ_1 when the available bandwidth is 1Mbps. In the 2-second spike, the measured RTTs for λ_3 and

λ_4 drops sharply to the same level as λ_0 and λ_1. λ_2 does not show a RTT decrease, because there is no sample with λ_2 during that spike. However, the average throughput fails to react to the first spike due to the smoothing function. At the second spike to 10Mbps, the RTTs for all sending rates fall to the same level. This means the available bandwidth is higher than the highest sending rate. The average throughput increases to around 1.5Mbps near the end of the spike and overestimates the available bandwidth for more than 10s. The quality adaptation algorithm may not correctly respond to the spike.

In Figure 5.5(c), we illustrate the case of short-term variations negative spikes. The available bandwidth at most of the time is 5Mbps. We emulate three spikes with duration of 2 seconds, 5 seconds, and 10 seconds dropping to 1Mbps at the time 30s, 62s, and 97s, respectively. In the 2-second spike, the rates λ_0 and λ_2 shows RTT inflation. However, the increase in RTT for λ_0 is due to packet loss during the moment of switching the available bandwidth setting in the click router. Other sending rates are not scheduled during this short spike. For the spike longer than 5 seconds, our measurement can capture significant RTT inflation for the sending rate higher than available bandwidth. Interestingly, the measured average throughput over-reacts for this first spike. The value drops to below 2Mbps for more than 5s. For the other two spikes, we can also see the delayed responses in the average throughput.

The final profile, as shown in Figure 5.5(d), emulates stepwise increases of available bandwidth. The available bandwidth increases from 500Kbps, 1Mbps, 2Mbps, 3Mbps, to finally 5Mbps for every 20 seconds. For the first 20 seconds, the RTTs for λ_1 to λ_4 significantly inflated. Even for λ_0, we can also observe a slight RTT inflation. the measured RTTs for $\{\lambda_1, \lambda_2, \lambda_3, \lambda_4\}$ can observe RTTs close to the RTTs of λ_0 at time $\{20, 40, 60, 80\}$s, respectively, as the available bandwidth increases to a value higher than the packet sending rates. On the other hand, the average throughput

requires 8 to 10 seconds to converge to a steady value after each bandwidth change.

To sum up, QDASH-abw is a novel probing methodology to determine the highest quality level the network can support. It speeds up the existing available bandwidth measurements by sending probes with a few selected packet sending rate. Our evaluation shows that QDASH-abw is accurate and is more sensitive to available bandwidth variations than the average throughput. With a timely estimation of the network condition, the video player at the client can better select a suitable video quality levels to download.

5.3 A QoE-aware switching algorithm

5.3.1 Buffer-aware strategies

Similar to other streaming technologies, such as UDP streaming and classic HTTP streaming, a buffer is deployed to reduce the chance of playback interruptions. However, it is not feasible to stream videos at a higher bitrate than the end-to-end available bandwidth. In this case, congestion and packet loss will occur at the bottleneck link. As a result, spatial artifacts, such as blocking frames, are often seen due to incomplete video decoding.

DASH transfers video data via TCP/IP. The reliable service provided by TCP ensures the integrity of video data. Video with a higher bitrate than the network throughput can be delivered without loss of picture quality. Even though the download rate is lower than the playback rate, interruption-free playback can be sustained for a short period of time by consuming the video data in the buffer. During this short time period, we can first request the video segments with quality levels between the original and the final level to append to the video buffer. By inserting intermediate quality levels, the original and final (lower) quality levels can be bridged.

Figure 5.6 gives an example of how we can utilize the buffer in quality level transitions. The *y*-axis at the left is the bitrate of the different quality levels, while the *y*-axis at the right is the measured network throughput. The *x*-axis represents the time. This example illustrates a video encoded into a set of five quality levels, $\mathcal{L} \in \{l_0, l_1, ..., l_4\}$. The solid line, dashed line, and the dot-dashed line represent the video levels requested by the video player, the network throughput, and the quality level shown to the user, respectively.

At time t_0, we assume the video buffer of size B is full. The video player plays and requests video segments at quality level l_4. However, the network throughput decreases to a level that is barely higher than the bitrate of quality level l_1 at t_1. After the video player detects the change at t_2, existing adaptation algorithms [19] switch the quality level to l_1 in order to adapt to the network conditions. As a result, users would perceive a sudden decline in picture quality. In order to avoid this, we propose to insert an intermediate level, for example l_2, between l_4 and l_1. With the same scenario, the download rate is higher than the playback rate (i.e. the time to download the segments,$(t_3 - t_2)$, is longer than the length of segments $(t_5 - t_4)$). If this situation were maintained for a longer time period, rebuffering would occur. However, while the video player is downloading segments of l_2, it continues playing the buffered video at the original quality. Therefore, we can use the buffered video playback period, from t_2 to t_4, to download the video segments at the intermediate level.

Although intermediate levels can smooth out the picture quality change, rebuffering events also reduce the QoE [165]. Downloading intermediate quality levels should also avoid causing the buffer starvation. The maximum number of intermediate quality

video segments to be downloaded, n_{frag}, is given by Equation (5.2).

$$n_{frag} = \lfloor t_{buffer} \frac{\beta}{s_{frag}} \rfloor, \tag{5.2}$$

$$t_{buffer} = \frac{B_{cur}}{\left(1 - \frac{\beta}{s_{frag}}\right)}, \tag{5.3}$$

$$s_{frag} = \lambda_i \times \theta, \tag{5.4}$$

where λ_i is the average bitrate of the intermediate quality level, l_i with a segment length of θ seconds of video. B_{cur} is the current video buffer size in second of video, and β is the degraded network throughput. s_{frag} computes the average size of each video segment of quality level, l_i. t_{buffer} is the time period for which the video buffer is able to offset the download of intermediate quality levels. Therefore, n_{frag} gives the number of complete segments that can be downloaded within t_{buffer}.

5.3.2 Designing an QoE-aware switching algorithm

From the results obtained in Section 3.5.1, the QoE can be increased by inserting intermediate levels in between quality level down switching. Hence, we formulate and propose a QoE-aware switching algorithm, which is shown in Algorithm 1, by considering the intermediate levels and video buffer size.

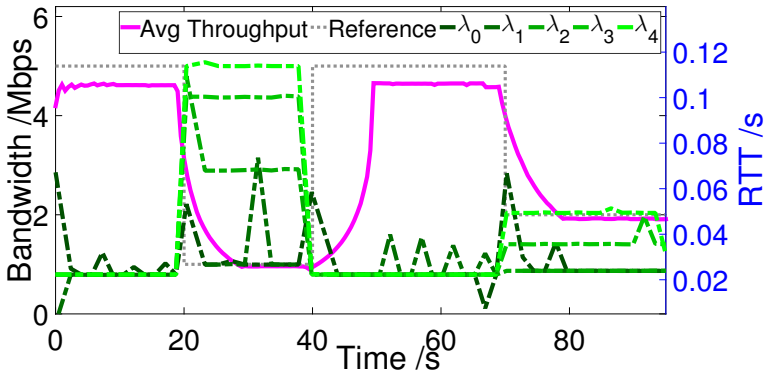
This algorithm is run before deciding the quality level of the next video segment. We obtain the supported quality level, $l_{support}$, by using `QDASH-abw`. If $l_{support}$ is lower than the current quality level, l_{cur}, by two levels, we compute the number of fragments of intermediate levels to be downloaded, n_{frag}, by using Equation (5.2)-(5.4). We choose the intermediate level as one level above the target quality level. So, the time period of watching at intermediate level can be maximized. It also shows effective in the QoE assessment in Section 3.5.1.

Algorithm 1 A QoE-aware quality adaptation algorithm

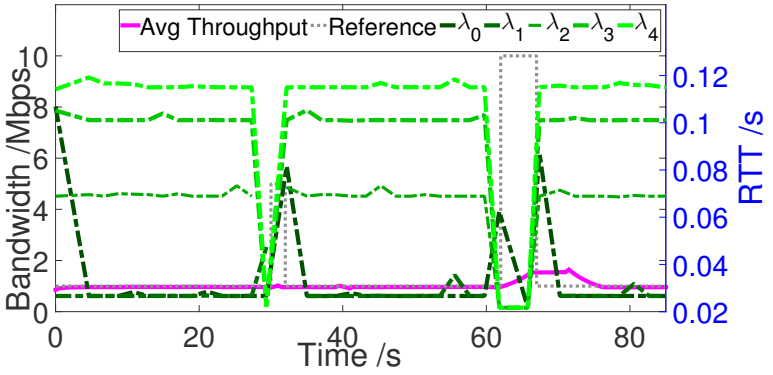
$l_{support}$: The quality level current network condition can support
l_{cur}: Current quality level
l_{nxt}: Proposed quality level
t_{buffer}: Number of intermediate quality video segments to be downloaded
B: Buffer size in video second
$b(l)$: Bit rate of quality level l
$s(l, \theta)$: Average size of 1 video fragment at quality level l
n_{frag}: Number of fragments for the proposed quality level
if $l_{support} < l_{cur}$ **then**
  if $(l_{cur} - l_{support}) > 1$ **then**
    $t_{buffer} \leftarrow \frac{B}{1 - b(l_{support})/s(l_{support}, \theta)}$
    $n_{frag} \leftarrow \lfloor t_{buffer} \times b(l_{support})/s(l_{support}, \theta) \rfloor$
    if $n_{frag} > 0$ **then**
      $l_{nxt} \leftarrow l_{support} + 1$
    else
      $l_{nxt} \leftarrow l_{support}$
    end if
  else
    $l_{nxt} \leftarrow l_{support}$
  end if
else
  $l_{nxt} \leftarrow l_{support}$
end if

5.4 Summary

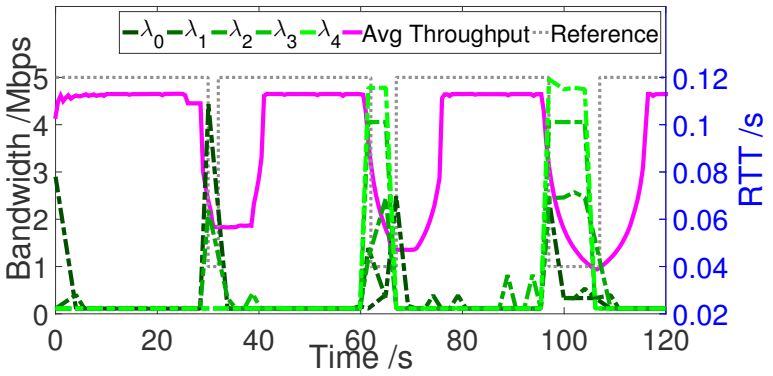
In this chapter, we proposed a QoE-aware DASH system—QDASH. It consists of two modules. QDASH-abw is a novel probing methodology which is tailor-made for DASH system to measure the network throughput. By reducing the choice of packet sending rate, QDASH-abw is sensitive to changes in available bandwidth and provides accurate decisions on which the video quality levels can be supported by current network conditions. On the other hand, we utilized our findings in our subjective assessments and designed QDAHS-qoe, a QoE-aware quality adaptation algorithm, to mitigate the QoE degradation caused by sudden decrease of network throughput.



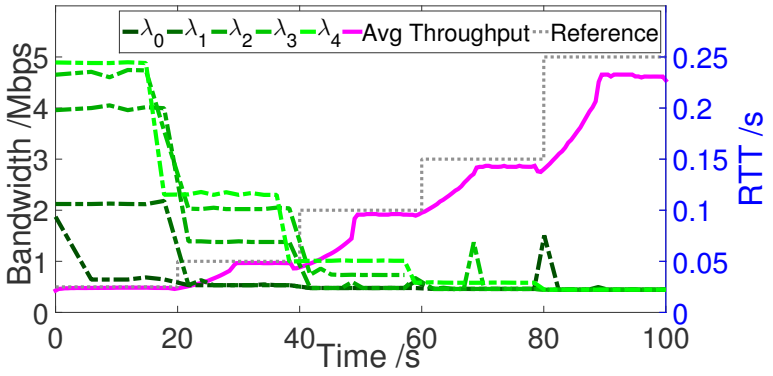
(a) Persistent variations



(b) Short-term variations – positive spikes



(c) Short-term variations – negative spikes



(d) Persistent variations – stepwise increases

Figure 5.5: Evaluations with four network profiles.

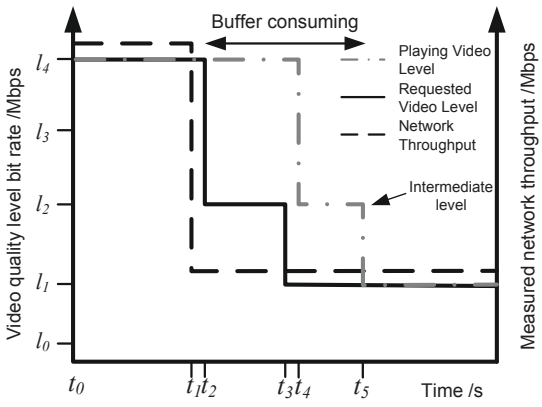


Figure 5.6: An example of buffer-aware quality transition.

Chapter 6

User-behavior Analytics for QoE Assessment

Assessing the QoE in real-world environment can be a very challenging problem. Users can be non-responsive, because they lack incentive unless the service level is unacceptable. In Chapter 3, we infer the QoE from the network QoS, and application layer events, which do not need to involve any user feedback. However, QoE is a complex construct, which includes both subjective and objective factors. Even though some works, as reviewed in Chapter 2, suggest to use user engagement as a measure of perceived quality, this metrics can be affected by other confounding factors, such as the video content and the availability.

To alleviate this problem, we propose to analyze the user behavior to help infer the QoE. In particular, user viewing activities, which refer to the activities that users interact with the video page or the video player interface, are employed. Figure 6.1 shows the enhanced QoE assessment model. We believe that the activities can reveal some subjective information relevant to the QoE, because the activities are generated when users interact with the application according to their cognitive process.

The major challenge of using user-viewing activities is to identify suitable user-

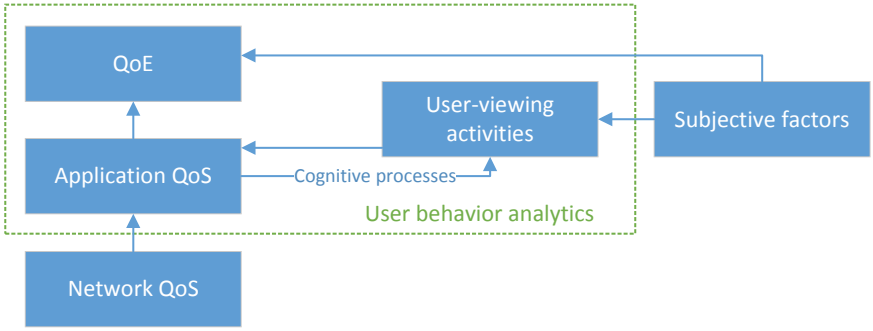


Figure 6.1: The enhanced QoE assessment model.

viewing activities in inferring the QoE. We set up a simple HTTP video streaming system and conduct subjective assessments to collect user-viewing activities from subjects. We analyze the data by listing out possible activities triggered by the impairments. After that, we use hypothesis testings to find out activities which are useful in estimating the QoE. Our results show that we can significantly improve the explanatory power of the QoE model by 8%.

In this chapter, Section 6.1 describes the user-viewing activities and overall methodology used in this study. Section 6.2 details the experiment setup, whereas Section 6.3 reports the experiment results.

6.1 User-viewing activities

User-viewing activities refer to the activities that a user interacts with a player interface. We have conducted a survey on how users behave when the playback is smooth or jerky. In the survey, we listed all the user-viewing activities in Table 6.1 except the two mouse movement items. The subjects were asked to rate their level of agreement on whether they will perform the listed activities under a given scenario (i.e., the playback is smooth/jerky). The level of agreement is measured with a 5-point Likert scale from 1 (strongly disagree) to 5 (strongly agree).

Two sets of ratings are obtained for each user-viewing activity. We compare the

two scenarios by subtracting the mean rating of each activity as shown in Equation (6.1). The mean rating difference of activity j, denoted by $\Delta\bar{r}^j$, is given by

$$\Delta\bar{r}^j = \bar{r}^j_{smooth} - \bar{r}^j_{jerky}, \tag{6.1}$$

where $\bar{r}^j_{smooth}$ and $\bar{r}^j_{jerky}$ are the mean rating of the user-viewing activity j given smooth and jerky playback scenarios, respectively. Using paired samples *t*-test, we can obtain the level of significance, p, for each activity.

The third column in Table 6.1 shows the mean rating difference of each user-viewing activity from a survey of 19 people. The activities with positive mean rating difference, $\Delta\bar{r}^j > 0$, means that users prefer those activities when the playback is smooth. Otherwise, users favor the activity in jerky playback scenario. The results show that users choose pausing, switching to a lower picture quality, and watching with normal screen size under jerky playback scenario. In contrast, for smooth playback scenario, only switching up the quality and enlarging the screen size are significant. Users show no significant preference for other activities, such as resuming and time shifting.

On the other hand, some user-viewing activities can help mitigate the temporal structure impairments [165]. Pausing that can increase the time for buffering video data is an example of positive technical impact (i.e., Tech. is $+$). Other examples include refreshing the page which allows the player to choose another video server from a content delivery network and switching to a lower video quality by reducing the video data size. In contrast, resuming and forward time shifting have negative impact. Resuming the playback consumes the buffered video, and forward time shifting gives up the buffered video and sends a new `HTTP GET` request. The effects of impact is shown in the second column of Table 6.1 represented by $+$, $\circ$, and $-$.

Table 6.1 also lists the possible user-viewing activities and their effects of impact,

explicit, and implicit meaning, particularly for HTTP video streaming.

6.1.1 The overall methodology

In this section, we describe the methodology for evaluating whether user-viewing activities are induced by temporal structure impairments or just user's random actions. Our research hypothesis is given in Hypothesis 1. To testify against the null hypothesis which claims that the activities are random, we analyze the activities recorded before and after the impairment events.

Hypothesis 1. *The user-viewing activities are more likely to be triggered after the presence of temporal structure impairments.*

We assume that an impairment event only affects the user-viewing activities nearby. Therefore, we inspect the user-viewing activities within a range around each impairment event. Figure 6.2 shows an example of a video playback timeline. The video starts playing at t_0 and ends at t_{end}. Three impairment events occur at times t_{i-1}, t_i, and t_{i+1}. Two user-viewing activities are recorded at times t_a and t_{a+1}. A time period δ, computed by Equation (6.2), is half of the time between the current impairment event and the nearest impairment events or the start or the end of the video playback. An upper bound for δ is arbitrarily set to 5 seconds to prevent the inclusion of irrelevant activities far away from the impairment event. Two other time periods d_{ai} and $d_{(a+1)i}$ are the time displacements from the activities at t_a and t_{a+1} to t_i, respectively.

$$\delta = \frac{\min(t_{i+1} - t_i, t_i - t_{i-1}, t_i - t_0, t_{end} - t_i, 10)}{2}. \tag{6.2}$$

Since random activities occur independent of the impairment events, they could occur before or after an impairment with equal probability. For the activities involving mouse clicks, the average time displacement to the impairment events is zero (i.e.,

Table 6.1: A list of possible user-viewing activities with the effect of technical impact, mean rating difference, explicit and implicit meaning.

Activities	Tech.	$\Delta\bar{r}^j$	Explicit Meaning	Possible Implicit Meaning
Pause	$+$	-2.37***	Stop playing the video playback for a short period of time.	More time is needed to buffer the video data.
Resume	$-$	0.26	Continue playing the paused video playback.	Reach the tolerance limit.
Refresh	$+$	-1.79	Reload the page and video.	The playback quality is unacceptable, and reloading may help.
Switch to a lower video quality	$+$	-1.79***	Watch the video with lower picture quality.	Scarify the spatial qualify for the playback smoothness.
Switch to a higher video quality	$-$	2.26***	Watch the video with better picture quality.	The user thinks the current speed is fast enough for watching with a better quality.
Play with full screen	$\circ$	2.42***	Watch the video with a larger size.	The playback is enjoyable.
Return to the normal-size screen	$\circ$	-0.74*	Watch the video with a smaller size.	The impairments are annoying.
Forward time shift	$-$	0.53	Watch the content after the current video position	The video content is not interesting.
Backward time shift	$+$	-0.58	Replay the content before the current video position.	Replaying the buffered video can result in a smoother playback.
Lost of window focus	$+$	-0.47	The browser window is covered or minimized.	The user may not be watching the video.
Frequent mouse movement	$\circ$	n/a	The user moves the mouse over the screen quickly.	The impairments are annoying.
Infrequent mouse movement	$\circ$	n/a	The user does not use the mouse.	The user is enjoying the video.

Note: $+$, $\circ$ and $-$ represent positive, neutral, and negative effect, respectively. For $\Delta\bar{r}^j$, *p <0.05, **p <0.01, ***p <0.001.

$\overline{D_i} = 0$), where D_i is the summation of time displacement within the range between $t_i - \delta$ and $t_{i+1} + \delta$ and n_i is the number of user-viewing activities within the range as shown in (6.3).

$$D_i = \sum_{j=0}^{n_i} d_{(a+j)i}. \tag{6.3}$$

On the other hand, the mouse movement at time t is quantified by the speed of the cursor movement, v_t, as shown in Equation (6.4), where (x_t, y_t) and (x_{t-1}, y_{t-1}) are the current and the pervious recorded coordinates, respectively. The speed of cursor movement is obtained by the Euclidean distance between the two coordinates over the difference in recorded timestamps, Δt. If the mouse movement is impairment-driven, the speed of the cursor movement is expected to be higher after the event.

$$v_t = \frac{\sqrt{(x_{t-1} - x_t)^2 + (y_{t-1} - y_t)^2}}{\Delta t}. \tag{6.4}$$

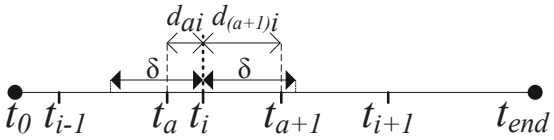


Figure 6.2: A timeline for a video viewing session with impairment events and user-viewing activities.

6.2 Evaluation

To validate our hypothesis, we have carried out experiments to record the subjects' video watching activities under various scenarios. We describe the experiment setup in this section and the results in the next.

6.2.1 Experiment setup

Figure 6.3 shows the experiment setup which is a simple video delivery system. The video server listening on multiple TCP ports responds with the same content. The Click modular router [131], placed in front of the video server, introduces delay and packet loss to the TCP flows. Table 6.2 lists the path metrics setting used by the router. The router emulates one of the 9 (3×3) combinations of path metrics on each TCP port. Therefore, connecting the video server via different ports results diverse network path performance. A mild level of cross-traffic with Pareto distributed inter-departure time and fixed size packets is generated by the distributed Internet traffic generator [62]. A workstation installed with a Endace DAG card [76] captures all the traffic between the click router and the video server. The logging server is responsible for recording the information reported by the customized video player (*cf.* Section 6.2.2) during the experiment. The traffic between the logging server and the video player, however, will not be captured and manipulated by the click router.

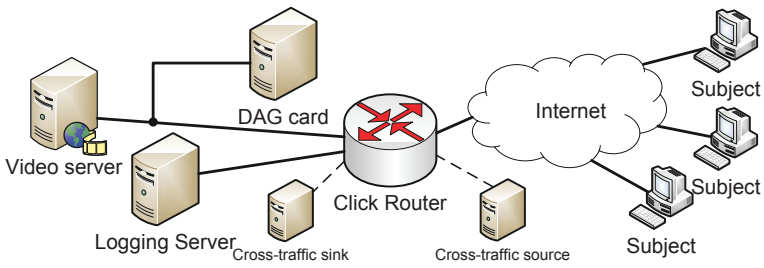


Figure 6.3: Experiment setup for the subjective experiment.

Table 6.2: Path quality settings used in the Click modular router.

Path quality metrics	Settings
Additional delay (ms)	0, 40, 80
Round-trip packet loss rate	0%, 2%, 4%

6.2.2 Client software

We have enhanced FlashTrack [165], a customized Adobe Flash video player, to record the application events. The events include the current position of video playback, the buffer status, the number of bytes loaded, buffer-full events, and buffer-empty events. In addition to those events, we record user-viewing activities listed in Table 6.1. Besides using Flash, we have employed Javascript to capture the cursor coordinates and browser's focus. All the events are logged periodically every 0.25 seconds. The logs are then aggregated and sent to a logging server every three seconds.

The video player provides all the basic functionalities offered by commodity video sharing websites. Users can pause and resume the video playback, changing the video quality, watch in full screen mode and forward/backward shift along the buffered video. Moreover, the downloading progress, current video position, and the length of the video are visible to the subjects. When a buffer-empty event occurs, a small screen is shown over the video displaying the percentage of buffer filled until the buffer is filled up. These visual components help the subjects understand the status of the playback.

6.2.3 Video materials

Three videos clips are chosen for the experiments. They are labeled as *speech*, *TVshow*, and *sports*, in an ascending order of temporal complexity. Video *speech* shows a person delivering a speech with static background. Video *TVshow* is a portion of a local TV comedy show. Video *sports* is a highlight of basketball games. Table 6.3 shows the detailed profiles of the video clips used in this experiment. The quality of the source videos is equivalent to 720p. The source videos are then down-sampled into three lower bit rate versions with H.264 codec according to the profiles of 480p, 360p, and 240p.

Table 6.3: Profiles of the video clips.

Parameters	720p	480p	360p	240p
Video width (pixel)	1280	854	640	400
Video height (pixel)	720	480	360	226
Video bit rate (Mbps)	2	1	0.5	0.25
Video frame rate (fps)	29.97	29.97	29.97	29.97
Audio bit rate (kbps)	128	96	80	32

6.2.4 Subjective assessment

After filling some basic information and answering questions on video-watching habits, each subject was given a list of four videos to watch. They were first instructed to access to a dedicated video to try the platform before starting the experiment. Through this training process, the subjects became more familiar with the testing environment and understood clearly about the functionality provided by the video player. After that, the subjects could freely select the watching sequence of the remaining 3 videos. To mitigate the order effects, the display order and the choice of network path performance of the remaining three videos were randomized, and the initial quality of all the videos was 480p.

The subjects were first informed that they might experience dissimilar performance for different links. Besides, they were also reminded to behave as usual and watch the entire video clips. At the end of each video clip, the subjects could immediately rate their perceived video-watching experience. A 7-point Likert scale of MOS was adopted, from 1 (“Bad”) to 7 (“Excellent”), for obtaining a higher granularity.

6.3 Results and analysis

6.3.1 User-viewing activities and network path quality

A total of 22 subjects, 16 male and 6 female, participated in the subjective assessment. All of them were non-experts in assessing the video quality. Nine of them carried out the experiment through the campus network, and others accessed the experiment platform through the public Internet. All the subjects reported that they spent at least one hour on surfing the Internet, and 20 of them watched at least one video on the web in the week before performing the experiment. Therefore, it is reasonable to assume that they are familiar with video-watching applications.

Figure 6.4 shows the frequency distribution of the ratings collected in the assessment. Although we observe very low frequency for the two extreme ratings (1 and 7), unrealistically poor or good performance usually contain less information. Another possible reason is that the subjects, who are all Chinese, avoid giving extreme ratings because of stronger central tendency bias [45]. However, we believe that the frequency of the rating between 2 and 6 can already provide enough variance.

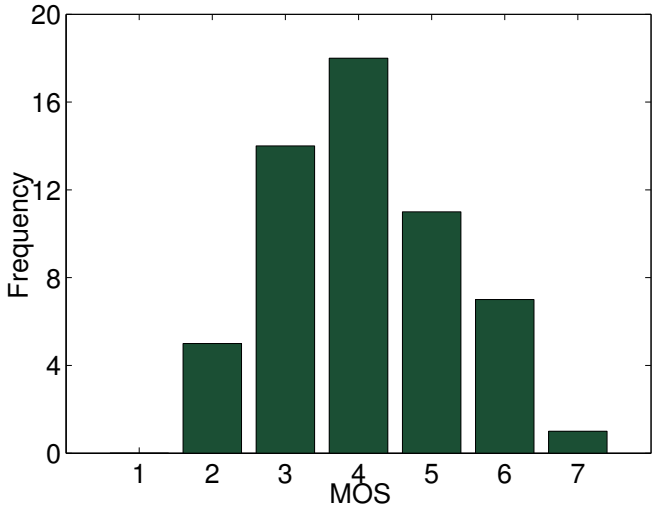


Figure 6.4: The overall distribution of MOS.

Figure 6.5 depicts the user-viewing activities recorded from one of the experiments. The bars shown along the x-axis are the cursor speed. The "$\times$"s and "$\circ$"s are the points when the buffer-empty and buffer-full events are triggered, respectively. The time that the subject pressed the pause button and the resume button are denoted by the "$+$"s and the "$\square$"s, respectively. The "$\triangle$"s and the "$\triangledown$"s are the respective times that the subject switched to full screen mode and normal screen mode. The quality switching events are indicated by the "$\triangleleft$"s (for switching down) and "$\triangleright$"s (for switching up). For a clearer illustration, the events are plotted in different levels.

Figure 6.5 shows that the application-level events, such as cursor movement, correlate with the user-viewing activities to some extent. In the first 50 seconds, the subject was still adapting to the network condition, and therefore the viewing activities are relatively few. After feeling that the playback is acceptable, she switched to a higher quality and watched in full screen mode just before the second buffer-empty event. However, after the second buffer-empty event occurs, the subject seemed to be annoyed by the event, switching back to the original quality quickly and returning to the normal screen size. She clicked the pause button and returned to normal screen mode until the end of the playback. After 200 seconds, she paused the playback for every buffer-empty event. Moreover, the cursor speed sharply increased whenever some user-viewing activities were captured, because the subject had to move the mouse cursor to press the pause buttons.

Figure 6.6, on the other hand, shows the RTT and packet loss rate (aggregated every second) measured from the same viewing session. To facilitate the comparison, the two time axes are perfectly aligned. The median RTT is about 80ms and the average packet loss rate is 4%. Loss busts are occasionally observed, but the network path was generally unchanged during the whole experiment session. The video was completely downloaded about 10 seconds before the video playback was finished. By comparing

Figures 6.5 and 6.6, we observe that using only network path measurement could fail to capture the important information about the user perceived QoE of the video, such as her dissatisfaction after the second rebuffering event.

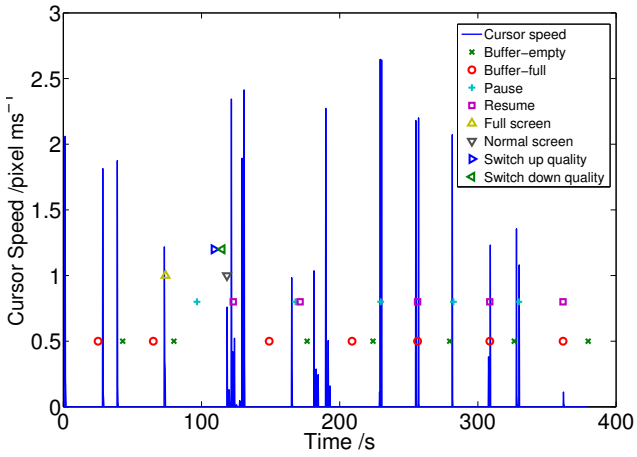


Figure 6.5: Time series of application-level events and user-viewing activities.

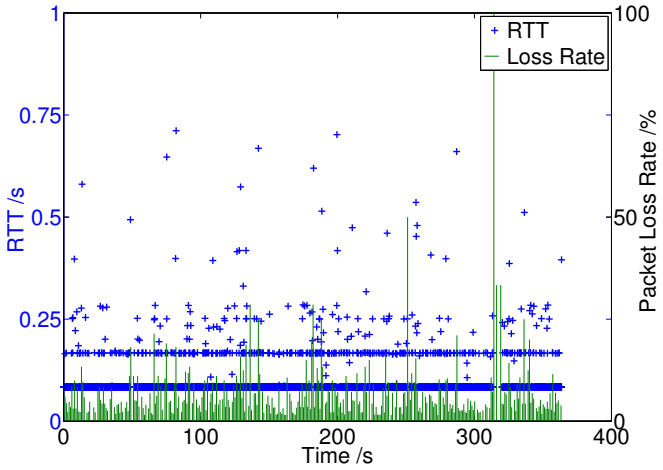


Figure 6.6: Time series of the RTT and packet loss rate.

6.3.2 Hypothesis testing for user-viewing activities

To give a generalized view of the activities, we formulate hypotheses for the user-viewing activities from Hypothesis 1 and test each hypothesis through statistical tests.

Using one-sample *t* test, we can obtain probability p that the null hypothesis is true for given mean and standard error [209]. If a user-viewing activity occurs after the impairment events, the average time distance is positive (i.e., $\overline{D_i} > 0$). We choose rebuffering (buffer-empty) events as the impairment event, because it is the main factor affecting the perceived quality. However, our method can also be applied to other impairment events.

Table 6.4 shows the average time distance, $\overline{D_i}$, of three user-viewing activities. The results show that the average time distance for the pause activity is significantly larger than zero, which means that users pause the video playback around two seconds after she encounters rebuffering events. Similarly, users change back to the normal-size screen from full screen about three seconds after the occurrence of impairment events.

The few seconds of delay between the impairment events and the activities can be regarded as user's *reaction time*. The average time distance for reducing the screen size is about one second more than pausing, implying that users usually pause the playback before reducing the screen size. As users know that pausing is functional, they regard it as a more critical action than reducing the screen size.

Although the activities of switching to a lower quality have a positive mean, it is not statistically significant due to small sample size ($N = 3$). A small activity count reflects that the subjects prefer pausing instead of switching the quality. The results for mouse movement are also not significant in our analysis, indicating that the average cursor speeds before and after the impairments are very similar.

6.3.3 Correlating user-viewing activities with QoE

In [165], we have showed that the rebuffering frequency is the main factor affecting the QoE, in terms of the MOS. Our results show that all three application performance metrics (APMs) impacted the QoE, and we have also found that using log transforma-

Table 6.4: Average time distance for different user-viewing activities and impairment events.

Activities	$\overline{D_i}$ (seconds)
Pause	1.94***
Switch to lower quality	2.19
Reduce the screen size	3.10**
	$\overline{v_t}$ (pixel/ms)
Mouse movement	-0.070

Note: $*p < 0.05$, $**p < 0.01$, $***p < 0.001$.

tion to correct the functional form of the rebuffering frequency and the initial buffering time can obtain a better model fit. As the MOS is ordinal in nature, the ordinary least-square regression cannot be applied. We have therefore adopted the ordinal logistic regression [34, 172] using SPSS [2] in the analysis below.

The left column of βs in Table 6.5 shows the regression results of solely using the APMs proposed in [165], where f_{rebuf} is the rebuffering frequency, T_{init} is the initial buffering time, and $\overline{T_{rebuf}}$ is the mean rebuffering duration. The model is significant with a χ^2 of 14.3 on 3 $d.f.$, meaning that the original model better explains the variations in the MOS than an intercept-only model. Among the three APMs, the rebuffering frequency and the initial buffering time are significant, while the mean rebuffering duration is marginally significant. The negative β means that the odds (probability) of obtaining higher MOS categories decrease with the rebuffering frequency and/or the initial buffering time. This implies that a higher rebuffering frequency or a longer initial buffering time has a lesser chance of obtaining higher MOS categories. However, an increase in mean rebuffering duration has a slightly higher chance of obtaining higher MOS categories. We adopt one of the pseudo-R^2 metrics, Nagelkerke R^2 [175], which ranges from 0 to 1, to represent the goodness of fit of the model. The explanatory power is moderate with a Nagelkerke R^2 of 0.24.

We further incorporate two of the user-viewing activities which show significant results in section 6.3.1 (i.e., pausing and reducing the screen size). We count the num-

Table 6.5: Regression results for the APM model and modified model.

Predictors	Estimates, βs	
	APM model	Modified model
$\ln(f_{rebuf})$	-0.54*	-0.62*
$\ln(T_{init})$	-0.60*	-0.54*
$\overline{T_{rebuf}}$	0.04 †	0.045*
N_{pause}	-	-0.68†
N_{screen}	-	0.94
Nagelkerke R^2	0.24	0.32
χ^2	14.3**	19.5**
$d.f.$	3	5
Valid N	54	54

Note: †p <0.1, *p <0.05, **p <0.01.

ber of pause and screen size reducing activities which are probably triggered by the impairments, denoted by N_{pause} and N_{screen}, respectively. The right column of βs in Table 6.5 shows the regression results of the modified model which is also significant with a χ^2 of 19.5 on 5 $d.f.$ The rebuffering frequency which shows significant result in the original model is still significant. For the new factors, we have obtained a marginal significance for the pause activities. By adding these two factors, the explanatory power, measured by the Nagelkerke R^2, increases from 0.24 to 0.32. The negative β of N_{pause} means that the probability of obtaining a higher MOS category increases when less pauses are triggered by impairment events. On the other hand, switching the screen size has no effect to the odds of MOS categories.

6.4 Summary

In this chapter, we studied how the user-viewing activities help evaluate the QoE of HTTP video streaming. We proposed a new methodology to examine the user-viewing activities around the occurrences of impairment events. From our subjective measurement results, we found that the impairments can trigger pause and screen size switching events after two and three seconds, respectively. We then incorporated these triggered

activities into the prediction model of the QoE to improve its prediction power by 8%. We also found that the pause activities are responsible for the variation of the MOS.

Chapter 7

Improving the Reliability of QoE Crowdtesting

Conducting QoE assessments in the crowd, also known as QoE crowdtesting, is one of the recent trend, because crowdsourcing platforms can provide diverse and large amount of human subjects. However, the workers in the crowd lack supervision. Some of them may intend to cheat and give unreliable and random responses, while they can also be distracted or unsuitable for the task. Both kinds of workers can lead to unreliable measurements. Therefore, identifying these workers can help improve the reliability of crowd-based assessments.

To address this problem, we analyze the worker behavior to infer the quality of workers. A worker behavior-based mechanism has three main advantages over existing anti-cheater methods, which was reviewed in Section 2.6. First, as the monitoring of worker behavior runs in the background, it is almost invisible to the workers. Therefore, fraudsters' attempts to evade the anti-cheating checks will be foiled. Second, the time and cost for conducting experiments can be reduced because the monitoring will not induce extra or redundant questions in the assessment. Finally, our mechanism is independent of the assessment results, so that the test items or stimulus are not required

to have any implicit ranking or absolute answers.

In this study, we propose a novel method for detecting low-quality workers. Our method constructs a predictive model based on the worker behavior. After collecting raw behavior data and the assessment results from the QoE crowdtesting, the first step is to estimate the quality of workers and assign a label for each worker by analyzing the assessment results. The second step is to extract useful features from the behavior data. The final step is to leverage supervised machine learning algorithms to train a model which can predict the quality of workers from the metrics. There are three major issues to be resolved. First, systematically analyzing the worker's behavior is hard. The second issue is how to obtain the ground truth (i.e., the quality of workers) as a label from the assessment results. The last issue is that implementing the assessment system has to be carefully designed to mitigate the performance impact caused by the capturing of the behavior.

We tackle the first issue by proposing a novel set of ten *worker behavior metrics*. This set of metrics can effectively extract information from the worker's behavior data captured from the browser. In the design of the metrics, the most challenging part is to systematically analyze the cursor trajectory. We quantify the cursor trajectory by its micro-movement and timing information. The second problem is alleviated by carefully designing the assessment task. Multiple existing anti-cheating techniques, such as reserved code items, are employed. Furthermore, we include human inspection. Hence, we can compose a *quality score* to reveal the quality of workers. The third issue is mitigated by carefully adjust the parameters in the system to balance between the performance and the frequency of feedback.

In our evaluation, we collect worker behavior datasets from our adaptive video quality assessments crowdsourced through MTurk and CrowdFlower. The worker behavior traces are used to compute the worker behavior metrics, while the assessment

results are used to compute the quality score. A mulitclass Naïve Bayes classifier [200] is employed to build a predictive model for estimating the quality of workers from the worker behavior metrics. Furthermore, four rating methods suitable for rating Likert scale, are investigated, including radio buttons, stars, slide bar, and number steps.

Our results show that four out of ten metrics can effectively infer the workers' quality. The error rates of the trained model for all rating methods are around 30%. These metrics include the submovement count, the time delay, the cursor's speed, and the number of extra clicks. We also find that, among four rating methods, stars and radio buttons are more effective than the other two to be used in detecting low-quality workers. By combining multiple rating methods, the accuracy of detecting low-quality workers can reach about 80%. Extending from our previous work [167], we further compare the performance with CrowdMOS [202]. Our method shows better precision and recall than CrowdMOS.

The outline of this chapter is structured as follows. An overview of our mechanism is provided in Section 7.1. After that, we describe the composition of quality score and worker behavior metrics in Section 7.2 and Section 7.3, respectively. The experiment setup, building of predictive model, and the results are presented in Section 7.4. We then further compare the cursor traces from different rating methods in Section 7.5.

7.1 Overview

Our approach is to analyze the behavior generated by the workers when they answer the questions in our QoE crowdtesting task. After the completion of the QoE crowdtesting campaign, we process the behavior traces to detect low-quality workers. Our method is enclosed by the solid blue box in Figure 7.1. The basic idea of the detection mechanism is to quantify the worker's behavior captured during the assessment into worker behavior metrics. After that, we apply a multiclass Naïve Bayes classifier to build a

predictive model to correlate with the quality of workers. A *quality score* is used to estimate the quality of the workers by the assessment results they submitted in the task and ratings from human raters. Finally, we can obtain a trained model which can be used to predict the quality of worker by the worker's behavior.

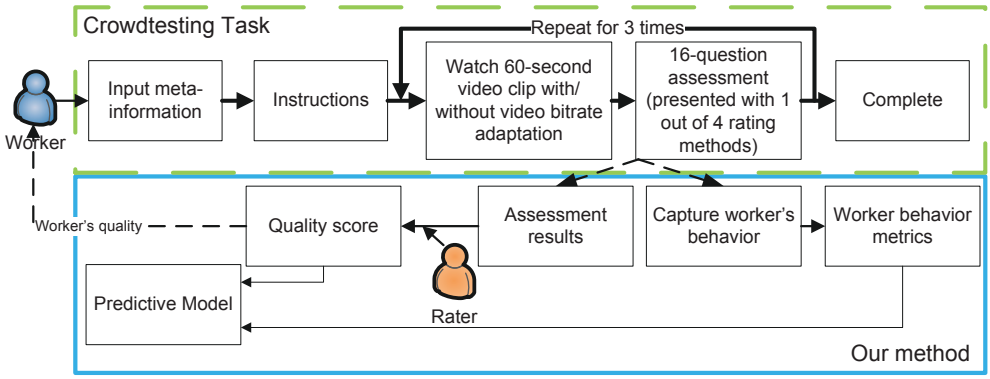


Figure 7.1: The overview of the workflow.

Employing workers' behavior has three main advantages over other cheater detection approaches. First, the capturing of worker behavior is almost invisible to the workers. This feature is important, because anti-cheating checks, such as plausible questions or consistency tests, can be easily located and evaded by individual sophisticated spammers. The checks may also fail to group attacks, which share the answers of a task among cheaters [68]. Second, the monitoring of worker's behavior does not incur any extra work load to workers. After obtaining the trained model, the anti-cheating checks can be reduced/removed. Thus, the length of of tasks can be decreased. As the cost for conducting experiments is often directly proportional to the task's length, our method can save experimenter's cost. Finally, our approach is independent to the actual ratings. This is particular useful for QoE assessments, because some subjective measures, such as expectation or enjoyment level, may not have an absolute answers or rankings. Discrepancy between workers may not be able to infer the worker's quality.

There are three main challenges in using worker's behavior to detect low-quality workers.

- Systematically analyzing the worker's behavior and identifying the factors enabling the detecting of low-quality workers are hard, because their behavior can also be affected by their responses or their own usage behavior. We tackle this challenge by designing ten *worker behavior metrics* to quantify relevant behavior, such as the micro-movements of the cursor trajectories and timing information throughout the assessment. In particular, we apply submovement analysis [159], which is common in the human-computer interface area to investigate the performance and accuracy of pointing devices, to extract the micro-movement information from the cursor traces. In addition, we adopt part of the cursor measures proposed in Hwang et al. [107] to quantify the cursor trajectories, such as the velocity and the acceleration of the cursor (*cf.* Section 7.3). In our analysis, we apply statistical tools to select four of the metrics to be included in our model.

- The second challenge is to obtain the ground truth about the quality of workers in order to correlate it with the worker's behavior. Even though the crowdsourcing platforms provide the historical acceptance rate of individual worker, we find that the workers' quality can be diverse. For example, a group of workers or bots can use the Shared Question Answer Dictionary (SQAD) method to boost their acceptance rate [68]. Therefore, we carefully design the questions in our QoE crowdtesting task, which includes reverse-coded questions and skip logic. We also analyze the open-ended responses and assessment results. By examining the violation of these logic, complexity of the responses and human inspection, we can deduce a *quality score* to estimate the quality of the workers (*cf.* Section 7.2).

- The final issue is related to a practical design of capturing the workers' behavior. The implementation of the assessment system has to be carefully designed to mitigate the performance impact caused by the capturing of the behavior which

could interrupt the normal behavior. Therefore, we carefully adjust the frequency of the feedback of the worker behavior. Therefore, the number of entries buffered at the browser can be reduced. We also utilize the cloud to improve the video streaming performance.

In this paper, we evaluate our method by an QoE crowdtesting task as shown in the green dashed box in Figure 7.1. Our crowdsourcing task is similar to typical QoE crowdtesting tasks requiring workers to rate according to their quality of experience. Our task is to assess the QoE of different video bitrate adaptation schemes, and is implemented as a simple web site similar to [196]. First, each worker was asked to provide some meta-information, such as gender and frequency of watching online videos. An instruction was then presented to the worker about the task, the questions to be asked, and the rating methods to be used. Each worker was required to rate the QoE of four 60-second video clips. Our customized video player adjusted the video bitrate in three of the video clips according to a pre-defined scheme, while the remaining one, serving as a control, was kept at a constant highest/lowest bitrate. After watching each video, the worker was then prompted to answer 16 questions. We expect a normal worker can finish the entire task using less than 20 minutes. The worker was given a unique validation code to return to the crowdsourcing platform for their payment. The implementation details will be discussed in Section 7.4.1.

7.2 Assessing the workers' quality

The subjective nature of QoE crowdtesting makes it unfeasible to measure the accuracy or precision of the workers. Unlike the previous works (e.g., [206] and [236]), our task does not have model answers. We tackle this problem by imbedding multiple cheater-detection tactics in our assessment, such that we can infer the worker's quality with some confidence. In addition to the single-item measure used in [114], we used 15

multiple choice questions and 1 open-ended question to measure the QoE. The workers were asked to rate the QoE they just watched from different aspects to improve the robustness of the measurement, including picture/sound quality, video content, and the smoothness of the playback, and so on. The workers were required to indicate whether they noticed any video quality adaptation. Three reverse-coded questions were set to measure the worker's reliability. We also implemented four rating methods commonly used in rating 5-point Likert scales (*cf.* Section 7.4.1). One of the methods was randomly chosen for workers in each assessment.

A *quality score* is used to quantify the quality of workers. It is composed of seven measures which are computed from the responses of all four video assessments and manual inspections. We assume the quality of worker does not change throughout the whole task. Therefore, we assign the quality score per worker instead of per assessment. The set of measures can be classified into three categories.

7.2.1 Complexity in text input response

There is an open-ended question in our assessment requiring the workers to input three words separated by commas about the content of the video they have just watched. This question is similar to the image/video annotation tasks. Based on their answers, we can examine whether the worker has paid sufficient attention to the video and the question.

We analyze the response by three metrics (q_{wc}, q_{ww}, and q_{wf}), which are related to the number of unique characters used, the number of unique words used, and the format of the response. We also manually inspect the content of the response and give a rating to each, q_{ct}. To compute q_{wc}, as shown in Equation (7.1), we first convert the response to small capital letters and then count the number of unique characters used. We normalise this index by dividing it by 26, which is the total number of English

alphabets.

$$q_{wc} = \frac{\text{No. of unique character used}}{26}. \tag{7.1}$$

Another measure, q_{ww}, considers the ratio of unique words to the total number of words in the response as shown in Equation (7.2). The unique words are found by grouping the same sub-string slitted by non-alphabet characters. We observe that some workers gave similar responses to all four videos with different content, such as "good" and "interesting".

$$q_{ww} = \frac{\text{No. of unique words used}}{\text{No. of total words used}}. \tag{7.2}$$

As the counting of characters or words cannot inspect the content of the responses, we also rate the responses (from 1 to 5) by a human rater as a measure, q_{ct}. Table 7.1 lists the rating criteria and examples of the respective score taken from our dataset. The responses are related to a NBA basketball match. Because our tasks only require the workers to input three words per assessment, the rating criteria mainly focus on the *accuracy* of the responses rather than the descriptiveness. The scores are then normalized as shown in Equation (7.3).

Table 7.1: The rating schemes and examples for rating the responses.

Score	Description	Example
5	All words describe different objects.	Basketball, Lakers, shot, ...
4	One or two two-word nouns broken into two words or some typos are found.	Basket, ball, match, ...
3	Some words describe the video content, but irrelevant words are found.	basketball, game, boring, ...
2	Descriptive words are used, but they mostly are not related to the video content.	entertaining, active, memorable,...
1	Simple words are used and they are not related to the video content.	Good, Nice, Impressive, ...

$$q_{ct} = \frac{\text{Rating of responses}}{5}. \tag{7.3}$$

7.2.2 Violation of soft rules

To ensure the workers reach a certain quality, some rules are stated in the instructions. For example, the workers have to watch the whole video without fast forwarding. Violating these "hard" rules can lead to a rejection of their work or stopping them from proceeding to the next assessment. On the other hand, "soft" rules do not lead to rejection, but they can reflect the workers' awareness to the instructions. There are two soft rules in our assessment. One of them is implemented in a question requiring the workers to indicate whether they notice any video quality adaptation. The workers were instructed to skip the next question if they did not notice any quality changes. However, we find that some workers did not skip the question as instructed. q_{jp} is the average count of correctly following the rules across the four assessments in the whole task as computed in Equation (7.4).

$$q_{jp} = \frac{\text{Count of correctly skipping/answering the questions}}{4}. \tag{7.4}$$

Another soft rule is about the formatting of the text responses, which requires the workers to input three comma separated words. Although it is feasible to enforce the formatting policy at the browser before the workers submit the answer, we do not limit the input. Therefore, we can capture the low-quality workers who input casually. We find that around 18% of workers did not input the correct format in all four assessments. Similar to q_{jp}, this measure, q_{wf}, is computed using the average number of correctly formatted input in Equation (7.5).

$$q_{wf} = \frac{\text{No. of correctly formatted inputs}}{4}. \tag{7.5}$$

7.2.3 Contradictory responses

Some low-quality workers tend to provide random ratings or the same rating for all questions. These workers can be easily screened out by applying reverse-coded questions. These questions are phrased in the semantically opposite direction to another one. For example, "The initial picture quality is too low." vs. "The initial picture quality meets my expectation." In our assessments, three questions are reverse-coded. We compose a measure, q_{rc}, which computes the average differences in ratings between the positively and negatively coded questions in all four assessments as shown in Equation (7.6).

$$q_{rc} = 1 - \frac{\sum_{j=1}^{4}\sum_{k=1}^{3}\Delta r_{jk}}{3 \times 4 \times 5}, \tag{7.6}$$

where Δr_{jk} is the difference in ratings of the k^{th} in the j^{th} assessment.

The last measure, q_{cn}, checks whether the workers can correctly identify whether the video streaming has any video bitrate adaptation. Because it is easy for workers to determine whether the video quality has changed or kept constant, we believe that this measure can reveal the level of concentration to the assessment. This measure averages the number of correctly identified assessments in the task as shown in Equation (7.7).

$$q_{cn} = \frac{\text{No. of assessments correctly identified constant/changing video bitrate.}}{4}. \tag{7.7}$$

Finally, the quality score, q, is the summation of the seven measures. Therefore, the score is between 0.4 to 7. A higher score means a better worker's quality.

7.3 Quantifying workers' behavior

In this session, we present our methodology on systematically quantifying the workers' behavior using *worker behavior metrics*. Before introducing the details of the metrics, we first provide some intuition for using mouse cursor trajectory to infer worker's quality by showing a typical case collected in our QoE crowdtesting task.

7.3.1 Observation

We believe that the information hidden in the mouse cursor trajectory can also help infer the quality of workers, because cursor movements strongly correlate with eye movement [50]. We manually select two workers from our experiment, an honest worker *R* and a low-quality worker *C*, to illustrate the importance of cursor trajectory. Figure 7.2 shows the cursor trajectories of the two workers. The red dotted rectangle is the area for answering the questions. We can see that worker *C* takes a much direct pathway to answer all the questions, but worker *R* shows more zig-zag paths in between questions. Besides the pathway, the velocity of the cursor also carries useful information. Figure 7.3 plots the velocity of the two traces in Figure 7.2. We can see that worker *C* moves the cursor with a much higher velocity and shorter-period pauses than worker *R*.

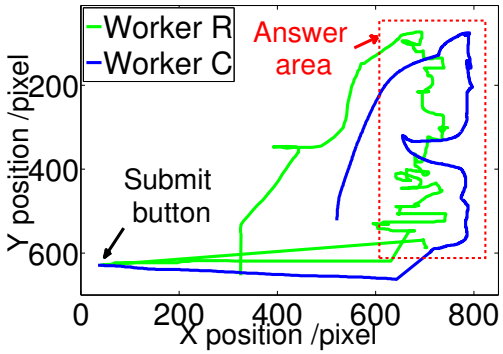


Figure 7.2: Mouse cursor trajectories of an honest worker *R* and a low-quality worker *C*

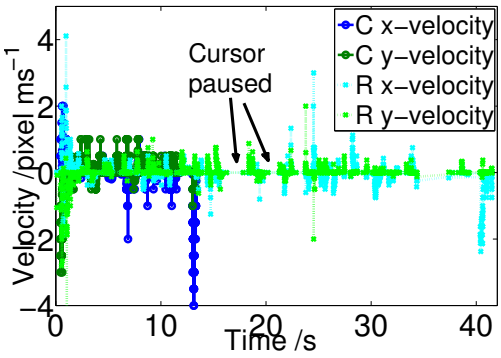


Figure 7.3: Cursor velocity of the data in Figure 7.2

7.3.2 Worker behavior metrics

Our raw behavior trace consists of a comprehensive set of cursor and mouse-related events. Table 7.2 lists the worker behavior collected from our experiments. Furthermore, we install callback functions for each rating objects, such as radio buttons or text fields, to distinguish between random clicks and clicks on the rating objects. Other browser events, such as the sizing or loss of focus, are also recorded. Each record is timestamped at the user side with time resolution of 1ms.

We propose a set of ten *worker behavior metrics* which extracts information from the start-up period, inter-question periods, and the overall time period. We believe that these three types of metrics can capture workers' cognitive process from different aspects. In the following sections, we will use the timeline in Figure 7.4 as an example. The assessment page finishes rendering and starts capturing worker behavior at time t_0. The worker clicks on the a^{th} question at time $t_c^{(a)}$. The *start-up period* is defined as the time period from the page rendered to the first click on the answer, whereas the inter-question time periods are defined as the time period between the worker answering a question and the next one.

Each cursor movement record contains the coordinates, x_j and y_j, and its timestamp t_j, where j is the j^{th} cursor movement record in the trace. The shaded area with solid color is the time period in which continuous mouse cursor movement with inter-

Table 7.2: A list of collected user behavior.

Event/Behavior	Description
Cursor coordinates	The mouse coordinates are recorded when the worker moves the mouse cursor over the page. The coordinates are relative positions to the top-left corner of the browser window containing the page and are only available when the window of the page is in focus.
Mouse clicking	The click position relative to the top-left corner of the page is recorded. We can distinguish the click events triggered by rating and random clicks.
Mouse over	A mouse-over event will be fired when the mouse cursor is moving over the stars in the star rating method.
Mouse up/down	Besides clicking, the slider bar may also be used by dragging the slider. We can identify the sliding time by recording the mouse up and down events separately.
Mouse scrolling	The amount of pixel scrolled by the worker is recorded by this event. We can use this information to recover the absolute coordinate of the page the worker is on.
Key strokes	The last question in our assessment is an open-ended question. We record how the workers key in the answers into the text field.
Window focus	This event can detect whether the window is in or out of focus.
Window resizing	The height and width of the browser window are recorded when the browser window is resized.

cursor movements less than 50 ms (i.e., $t_j - t_{j-1} < 50$ ms) is recorded. Otherwise, we treat the movement as a pause which shaded with vertical strokes. We use $t_p^{(k)}$ and $\tau_p^{(k)}$ to denote the beginning and the end of the k^{th} pause event, respectively. We let the total number of cursor movement records and clicks to be N and C, respectively. In the rest of the chapter, we use these notations to introduce the computation of the worker behavior metrics.

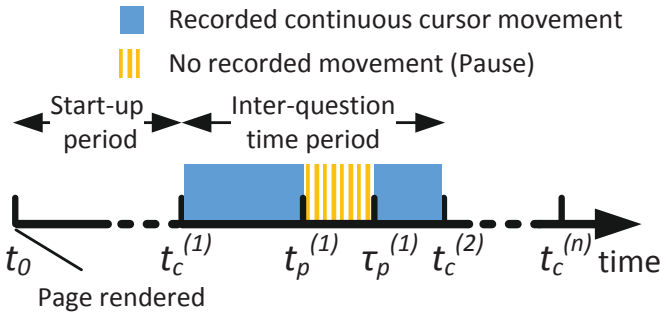


Figure 7.4: A timeline of events.

Another challenge is to systematically analyze the trajectory, because the trajectory can be affected by many factors, such as the responses and the worker's habit. In this work, we apply submovement analysis [159] to capture the micro-structures in the trajectories. Figure 7.5 shows an example of a cursor trajectory consisting of two submovements. The horizontal dotted line connects the start and end points. We assume the origin (i.e., $(x, y) = (0, 0)$) is at the top left corner. The first submovement is in upward direction away from the horizontal line (i.e., $y-$velocity<0) until the red dotted line. The second submovement changes to downward direction until reaching the end point (i.e., $y-$velocity>0). Furthermore, Hwang et al. [107] propose a set of cursor measures, which are mainly based on submovement analysis, to analyze the performance and accuracy of pointing devices for different kinds of users. We also adopt the cursor measures to infer workers' quality.

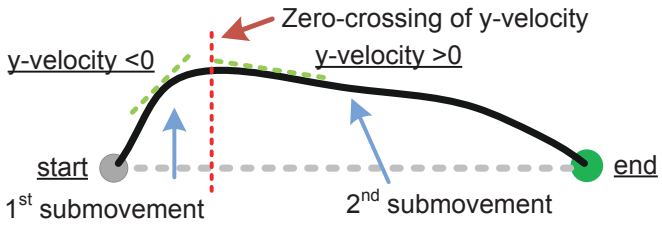


Figure 7.5: A submovement example.

We compute the submovements from cursor trajectories using the following steps. We define function $\mathcal{S}(a, b)$ to be the number of submovements between time epoches a and b. The collected cursor trajectories are represented by a series of points $\{t_j, x_j, y_j\}, \forall j = 0, 1, ..., N-1$, where x_j and y_j are the x and y coordinates, t_j is the timestamp of this datum, and N is the total number of records. To compute $\mathcal{S}(t_0, t_{N-1})$, we compute the x and y components of the cursor velocity by $(x_{j+1} - x_j)/(t_{j+1} - t_j)$ and $(y_{j+1} - y_j)/(t_{j+1} - t_j)$, respectively. Finally we count the number of zero-crossings (i.e., from positive to negative value or vice versa) in either $x-$ or $y-$ velocity components as the number of submovement.

Overall submovement count, number of pause and median pause duration

The first three metrics quantify the worker behavior throughout the assessment task. The total number of submovement as computed in Equation (7.8) can quantify the micro-movement generated by the workers during the assessments and reveal whether the worker takes a very straight forward pathway to complete the task.

$$m_{tc} = \mathcal{S}(t_0, t_N). \tag{7.8}$$

Submovement can only reveal the direction of movement. To obtain the temporal measures, we consider the number of pause, P, and the median pause duration, m_{td}. Equation (7.9) shows the computation of m_{td}. We consider there is a pause event whenever the cursor stays at the same position for longer than 50 ms. We employ a shorter time than the one used in [206], because our task is relatively simple and the workers can answer quickly.

$$m_{td} = Md(\{\tau_p^{(i)} - t_p^{(i)} | i = 2, ..., P\}), \tag{7.9}$$

where $Md(\cdot)$ returns the median value of the input set.

Start-up time and submovement count

The following two metrics particularly focus on the start-up period when the workers may skim through the questions before answering the questions and move the mouse cursor. We quantify this behavior by measuring the length and counting the submovement of the start-up period (denoted by m_{st} and m_{sc}, respectively). These two metrics

can be computed by Equations (7.10) and (7.11).

$$m_{st} = t_c^{(1)} - t_0. \tag{7.10}$$

$$m_{sc} = \mathcal{S}(t_0, t_c^{(1)}). \tag{7.11}$$

Number of extra clicks

We count the number of *extra* clicks generated by the workers, denoted by m_{tk}. We subtract the minimum number of clicks required to complete the task from the number of clicks recorded by the trace. For example, we assume that only one click is required for answering a multiple choice question with radio buttons. We consider this metrics because low-quality workers are tend to finish the task with minimum effort. However, extra mouse clicks is additional to the task. Therefore, we believe that this metrics can help screen out low-quality workers.

Median inter-question time and submovement

In addition to the overall and start-up period statistic, we consider the behavior during the inter-question period. The callback function installed in each rating object allows us to identify the questions the worker answered. We can easily slice the trace by the time the worker answering the questions. After that, we can compute the median length of time, m_{it}, and the number of submovement generated, m_{is}, in between answering each question by Equations (7.12) and (7.13), respectively. By quantifying the inter-question behaviors, we can mitigate those spammers who attempts to evade the timing or submovement check by generating extra movement after finishing the task.

$$m_{it} = Md(\{(t_c^{(i)} - t_c^{(i-1)}) | i = 2, 3, ..., C\}), \tag{7.12}$$

$$m_{is} = Md(\{\mathcal{S}(t_c^{(i-1)}, t_c^{(i)}) | i = 2, 3, ..., C\}). \tag{7.13}$$

Median cursor speed and acceleration

From our observation, we find that the dynamics of the mouse movement is also important in detecting low-quality workers. Therefore, kinematic analysis [190] is employed as a part of our metrics. The median cursor speed, m_{cs}, and acceleration, m_{ca}, are the first and second derivatives of the coordinates which can be computed in Equations (7.16) and (7.17), respectively. These two metrics are important measures in characterising the cursor trajectories [190, 107].

$$\delta D(j) = \sqrt{(x_j - x_{j-1})^2 + (y_j - y_{j-1})^2}, \tag{7.14}$$

$$\delta t(j) = t_j - t_{j-1}, \tag{7.15}$$

$$m_{cs} = Md(\{\frac{\delta D(i)}{\delta t(i)} | i = 2, 3, ..., N\}). \tag{7.16}$$

$$m_{ca} = Md(\{\frac{\delta^2 D(i)}{\delta^2 t(i)} | i = 3, 4, ..., N\}). \tag{7.17}$$

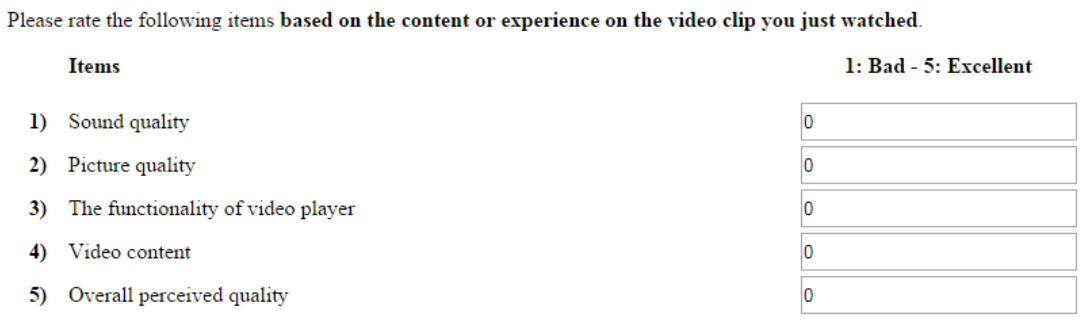
7.4 Evaluation and Results

Our aim of the evaluations is to investigate the relationships between the worker behavior metrics and the quality of workers. In our evaluation, a QoE crowdtesting task is published to evaluate the perceived quality of different video bitrate adaptation schemes as we described in Section 3.5.2. We then analyze the results for training the predictive model.

7.4.1 Implementation of the QoE crowdtesting task

Because the worker's behavior can be influenced by the layout of the page, our implementation has considered the spacing and size of the objects on the question page. The upper part of the question page using number field is shown in Figure 7.6. The width of the page is 800 px, which modern PCs should be able to display without

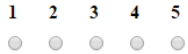
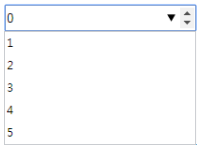
horizontal scrolling. Furthermore, we have implemented four common methods for rating questions in 5-point Likert scale . They are radio buttons, slider bar, number field, and stars. Table 7.3 shows the outputs of the rating methods rendered by Chrome and Firefox. We used the `RateIt` [6] jQuery library to implement the stars. Although the other three methods are the input types natively supported in HTML5, we can see that the rendering of the slider and number field method is slightly different in the two browsers. Chrome provides more feature to the two methods. It supports tags above the slider to indicate the positions of the values, and provides drop-down menu showing all available options when users click on the number field. The size of all rating methods are unified as 200 px (width) $\times$ 20 px (height).


Please rate the following items **based on the content or experience on the video clip you just watched**.

Items	**1: Bad - 5: Excellent**
1) Sound quality	0
2) Picture quality	0
3) The functionality of video player	0
4) Video content	0
5) Overall perceived quality	0

Figure 7.6: The layout of the question page.

Table 7.3: The sample display of the four rating methods used in our task.

Rating methods	Implementation	Chrome	Firefox
Radio buttons	`<input type="radio">`		
Slider	`<input type="range">`		
Number field	`<input type="number">`		
Stars	`RateIt` plugin [6]		

We have implemented a jQuery-based library to collect the worker's behavior as

listed in Table 7.2 from the browser. The library is deployed in the question page and executed upon the rendering of the page. Our library is run at the background, and it sends back the worker behavior to our server using AJAX every second. We have considered several time interval for sending back the worker behavior. Sending back worker behavior frequently could generate excessive network traffic and overheads, whereas the performance of the workers' browsers could be slowed down when there was a large amount of data in the browser's buffer. At the server side, we used `php` and `MySQL` to receive and store the data. We believe that the library can be easily intergraded into other web-based QoE crowdtesting frameworks (e.g., [126, 132, 82]).

Since we cannot control the access time of workers, our assessment platform is required to support multiple workers to simultaneously watching HD video, which are quite bandwidth demanding. Therefore, we utilized two separate servers and the cloud [126]. Figure 7.7 depicts the overall architecture of the assessment platform. The web server, which is hosted by a web hosting company in the US, is responsible for hosting the assessment interface and logging the worker behavior recorded at the browsers, while the video streaming server is an Amazon EC2 virtual machine. We also used the CloudFlare CDN to cache the video data to ensure the server side is not the bottleneck.

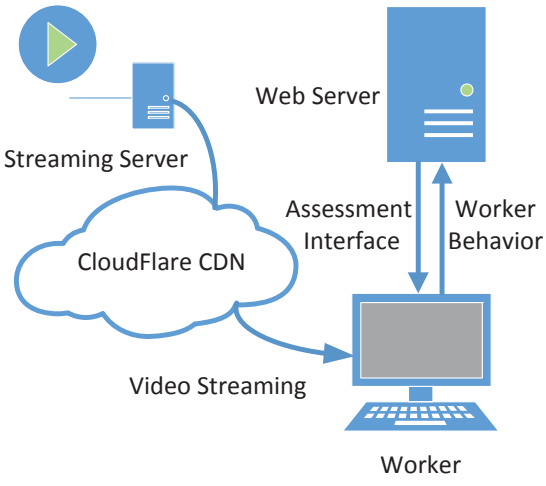


Figure 7.7: The architecture of our assessment platform.

We published our QoE crowdtesting task to two major crowdsourcing platforms, Amazon Mechanical Turk (MTurk) and CrowdFlower. Only US workers were able to perform our tasks of evaluating the video steaming performance. We chose the workers with acceptance rate higher than 90% in MTurk and the "highest quality" in all available channels in CrowdFlower, because we targeted to capture the behavior of more intelligent cheaters rather than those who could be easily detected. Each completed worker was awarded $0.5 US.

7.4.2 Results

Descriptive statistics

We successfully collected results from 172 workers (42 MTurk workers and 130 CrowdFlower workers from 20 channels). As each worker was required to rate four video clips, we collected a total of 688 (=172 $\times$ 4) samples of worker behavior data. Nearly half of the workers (49.3%) are male. Due to the compatibility issue for our Flash-based video player, our platform does not support Apple and Internet Explorer users. Moveover, 63.4% of the workers used Google Chrome browser to perform our tasks, while others used Mozilla Firefox. Almost all workers used Windows. Only one worker used Linux. Figure 7.8 shows the CDF of the completion time of each assessment. The median time for {MTurk/CrowdFlower} workers to complete one assessment is {67.8s/71.3s}, respectively. About 90% of the workers submitted the assessment in 150s. However, we also found a few workers taking longer than 5 minutes. 58 CrowdFlower workers also submitted post-task feedback through the CrowdFlower platform. The average satisfaction to our award is 4.1 out of 5. We believe that the workers generally satisfied the pay and were not motivated to provide low-quality work because of underpayment.

The CDFs of the quality score of all workers is plotted in Figure 7.9. The dotted

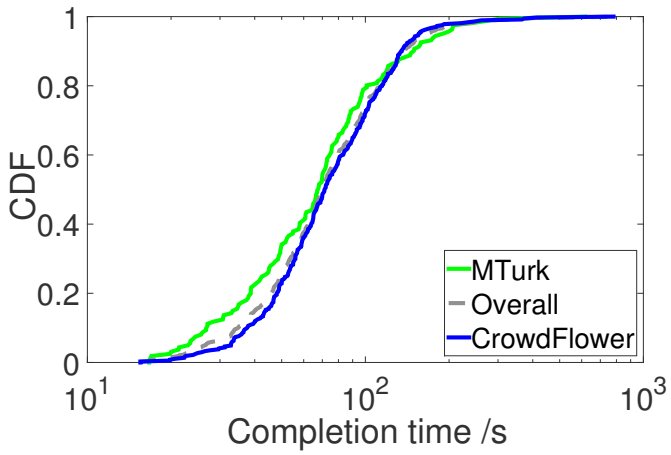


Figure 7.8: The CDF of completion time of each assessment (x-axis in log scale).

grey line, light green line and the dark blue line are the CDFs of the overall quality score, and the quality score from MTurk and CrowdFlower workers, respectively. We can see that workers from MTurk generally have a lower score than those from CrowdFlower. The median score for {MTurk,CrowdFlower} workers is {4.6,5.2}, which shows that the workers from CrowdFlower has a better quality than those from MTurk.

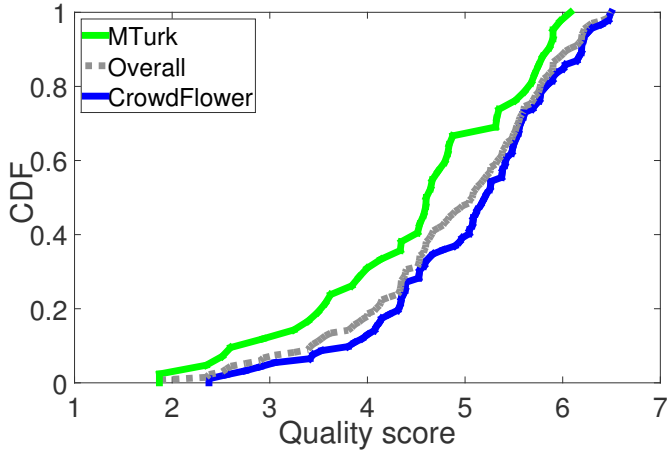


Figure 7.9: The CDFs of the quality scores.

We also study the characteristics of the quality score and the measures using correlation analysis and principal component analysis before employing them in our training. Table 7.4 shows the correlation table for the seven measures used to compose the

quality score. All the measures are expectedly positively correlated with each other.

Table 7.4: Correlation analysis among metrics of quality score.

	q_{wc}	q_{ww}	q_{rc}	q_{jp}	q_{cn}	q_{wf}
q_{ww}	.55***					
q_{rc}	.13	.082				
q_{jp}	.07	.084	.72***			
q_{cn}	.32***	.068	.069	.009		
q_{wf}	.39***	.15†	.055	.059	.41***	
q_{ct}	.57***	.68***	.033	.016	.16†	.31***

Note: †$p < 0.1$, *** $p < 0.001$

The measure, q_{wc}, is not only significantly correlated with other measures related to the word input (i.e., q_{ww}, q_{wf}, q_{ct}). It is also significantly correlated with a consistency measure, q_{cn}. Besides, we find that the workers with less contradictory responses, q_{rc}, have better performance in following the soft rule, q_{jp}. The workers obtained a high value in identifying bitrate changes also tend to provide correctly formatted text responses.

We observe that there are two distinct sets of variables which are highly correlated (i.e., $\{q_{wc}, q_{ww}, q_{wf}, q_{ct}\}$ and $\{q_{rc}, q_{jp}\}$). Hence, we apply the principal component analysis to the measures to understand the underlying factors. Figure 7.10 plots the percentage of variance explained of the components. We find that there are three major components explaining over 87% of variances. We believe that the three components are related to the three categories of the measures.

Model training

To train the predictive model, we include all or some of the worker behavior metrics and compare their error rates for the best model. Multinomial distribution is applied to count data, such as m_{sc} and m_{tk}, denoted by ‡ in Table 7.5. In the following, we adopt a 10-fold cross validation method to estimate the error rate of the classifiers. Because

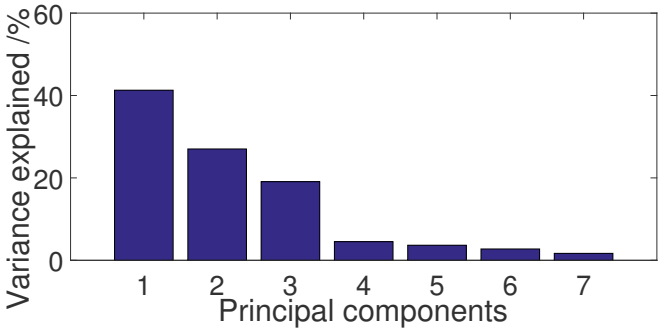


Figure 7.10: The result of a principal component analysis of the seven measures.

of the randomness in the cross validation process, the error rate of each model is the average error rate computed from 100 times of training.

We first apply all the ten worker behavior metrics into the model as shown in the first row of Table 7.5. Figure 7.11 plots the average error rates of the model with 95% confidence intervals. In model 1, stars and slide bar rating methods can achieve error rates of 31.8% and 33.1%, respectively. However, the error rates for radio buttons and number field are large (i.e., 38.3% and 59.0%, respectively).

The poor results are due to some metrics failing to fit the model well. To select the metrics to be included in the model, we use a simple linear regression model to identify the metrics which can better predict the quality score. It is found that four of the metrics are statistically significant in at least one of the rating methods: m_{sc}, m_{it}, m_{cs}, and m_{tk}. Guided by the results from the linear regression analysis, we selected these four metrics to form models 2 to 5 as listed in the second to fifth row in Table 7.5 and re-train them using the multiclass Naïve Bayes classifier. Subsequently, the average error rates are decreased to around 30%. Among the four rating methods, the radio button and stars are better other two methods with an error of 27.2% in model 5. All rating methods except the number field method also obtained lowest error rates in model 5, whereas the number field method achieved the lowest value in model 3. Therefore, we suggest that different sets of worker behavior metrics should be used

when the rating method is different.

Table 7.5: Worker behavior metrics used in various models.

Model	Worker behavior metrics									
	m_{st}	m_{sc}‡	m_{tc}‡	P‡	m_{td}	m_{it}	m_{is}‡	m_{cs}	m_{ca}	m_{tk}‡
1	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
2		✓				✓		✓		✓
3		✓						✓		✓
4		✓				✓		✓		
5		✓				✓				✓

Note: ‡the variables applied multinomial distribution.

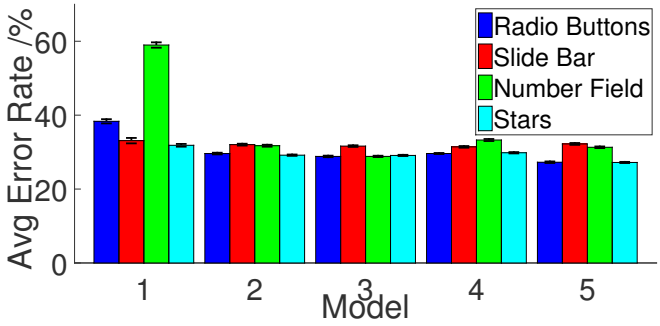


Figure 7.11: The average error rate of different models.

We further analyze the performance of the models using the model with the lowest error rate for each rating method to predict the worker's category. We then compute the differences between the actual and the predicted worker's category, ΔL, with Equation (7.18).

$$\Delta L = l - \check{l}, \tag{7.18}$$

where l is the actual category, and $\check{l}$ is the predicted category.

Table 7.6 shows the percentage of workers in each category. The percentage of correctly predicting the categories for all four rating methods ($\Delta L = 0$) is quite high, which is ranged from 73.3% to 82.6%. The quality of less than 2% of workers is under-estimated by two categories ($\Delta L = 2$). This can cause the false alarm that the worker is wrongly classified as a low-quality one. However, it is still acceptable in practice, because the honest workers can complaint to the experimenter on his decision. The

experimenter can also manually check those workers labeled as low-quality. On the other hand, there are around 16% and 6% cases which are over-estimated by one and two categories, respectively. The false negative rate is barely satisfactory.

However, we found that different sets of workers are labeled as low-quality in different rating methods. We hence label the workers using the lowest category predicted by all the rating methods. The fifth row in Table 7.6 shows the combined results. The accuracy can be retained, whereas the percentage of over-estimating the categories ($\Delta L < 0$) is significantly reduced to 9.3% (=0.58%+8.72%).

Furthermore, as the number of low-quality workers are often less than good ones, the accuracy can be easily inflated by overestimating their quality. Therefore, we also examine whether the model can successfully identify low-quality workers. We manually inspect the low-quality workers reported by the four rating methods. Both stars and number field methods report the same set of six workers. Radio buttons and slide bar can detect five and three low-quality workers, respectively. However, only two workers can be found by at least two rating methods. This is probably due to the different behavior induced by the rating methods. We will further discuss this issue in §7.5. By combining four rating methods, 78.6% (11 out of 14) of low-quality workers are correctly identified.

Table 7.6: The percentage of workers showing differences between the actual and predicted worker's category.

Rating methods	ΔL				
	-2	-1	0	1	2
Radio buttons	4.07%	12.8%	82.6%	0.58%	0%
Slide bar	5.23%	17.4%	75.6%	1.74%	0%
Number field	4.65%	15.1%	73.8%	4.65%	1.74%
Stars	6.98%	16.3%	73.3%	2.91%	0.58%
Combined	0.58%	8.72%	82.6%	6.40%	1.74%

Comparison with CrowdMOS

We further compare our method with the post-screening procedure in the CrowdMOS [202]. The main idea of the procedure is to compute the correlation between submitted ratings and the mean rating of the same cases. The worker will be rejected if the correlation coefficient is lower than a threshold, r_{th}. The sample correlation coefficient of worker n, r_n, can be computed by Equation (7.19).

$$r_n = \frac{\text{cov}(\hat{\mu}^1_{(n,1)}, ..., \hat{\mu}^7_{(n,4)}; \hat{v}^1_{(n,1)}, ..., \hat{v}^7_{(n,4)})}{\text{std}(\hat{\mu}^1_{(n,1)}, ..., \hat{\mu}^7_{(n,4)})\text{std}(\hat{v}^1_{(n,1)}, ..., \hat{v}^7_{(n,4)})}, \tag{7.19}$$

where $\text{cov}(\cdot)$ and $\text{std}(\cdot)$ return the covariance and standard deviation of the values, respectively. $\hat{\mu}^c_{(a,b)}$ represents the rating for question c given by worker a in the b^{th} assessment and $\hat{\mu}^c_{(a,b)}$ represents the mean rating from all the workers in question c under the scenario experienced by worker a in the b^{th} assessment.

However, our task only required the workers to rate on four video clips, which is insufficient to compute a meaningful correlation coefficient. Hence, we consider seven ratings from each assessment, such as the rating on sound quality, playback smoothness, and video content, instead of only examining the overall perceived quality (MOS).

Because the threshold, r_{th}, is an arbitrary chosen value [202], we evaluate a range of threshold value from 0.2 to 0.5. We label those workers with $\{r_n < r_{th}/r_n \geq r_{th}\}$ as {'reject'/'accept'}. Figure 7.12 shows the recall, precision, and accuracy of CrowdMOS against different correlation coefficient threshold. The recall, precision, and accuracy can be computed by Equations (7.22), (7.21), and (7.20), respectively. The accuracy and precision reached the maximum at $r_{th} = 0.25$ with the value 98.3% and 38.5%, respectively. This result also matches the value of r_{th} recommended in [202]. Both measures decrease when r_{th} further increases, because more workers are

incorrectly classified as 'cheaters'. However, the recall increases from 14.3% to 64.3% as r_{th} increases from 0.2 to 0.5, because more cheaters are spotted as the criteria of labeling cheaters releases.

$$\text{Accuracy} = (tp + tn)/W, \tag{7.20}$$

$$\text{Precision} = tp/(tp + fp), \tag{7.21}$$

$$\text{Recall} = tp/(tp + fn), \tag{7.22}$$

where tp, tn, fp, and fn are the number of true positives, true negatives, false positives, false negatives, respectively. W is the total number of workers.

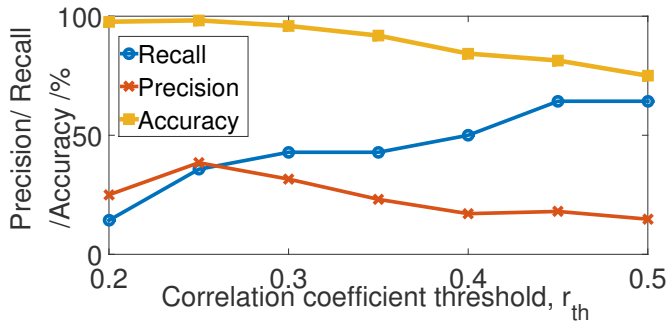


Figure 7.12: Recall and precision of CrowdMOS approach against different correlation coefficient, r_{th}.

Table 7.7 shows a comparison of the accuracy, precision, and recall CrowdMOS and our method. For CrowdMOS, we set to threshold to be 0.25 (i.e., $r_{th} = 0.25$). For our method, we use the results combined by all rating methods. CrowdMOS only supports making binary decision. Therefore, the 'low-quality' workers labeled by our method are equivalent to the 'reject' category in CrowdMOS. Other workers are regarded as 'accept'. CrowdMOS has a slightly higher accuracy than our method, because it selects less workers as low-quality ones than our method and results in more true negatives. However, our method has significantly higher precision and recall, because it can correctly detect more low-quality workers.

Table 7.7: A comparison of results of CrowdMOS and our method.

	Accuracy	Precision	Recall
CrowdMOS ($r_{th} = 0.25$)	98.3%	38.5%	35.7%
Our Method	93.0%	55.0%	78.6%

7.5 Discussion

In this section, we will discuss how different rating methods can affect the worker behavior and the limitation of our method.

7.5.1 The influence of rating methods

Figures 7.13(a) – 7.13(d) show the fragments of the cursor traces and mouse events from a typical and acceptable worker rating with radio buttons, number field, slide bar, and stars methods, respectively. All the figures are in the same scale. The degree of horizontal movement for radio buttons and number field is lower than slide bar and stars. Moreover, the number field can trigger more vertically up and down movement, which is probably due to the menu in Chrome. Interestingly, even though both slide bar and stars can be used by directly clicking on the position corresponding to the rating or the number of stars, the worker still controls them by drag-and-release or by moving over the stars. We believe that, due to these differences in the behavior, we cannot apply the same set of worker behavior metrics to the model for all the rating methods.

7.5.2 Limitation

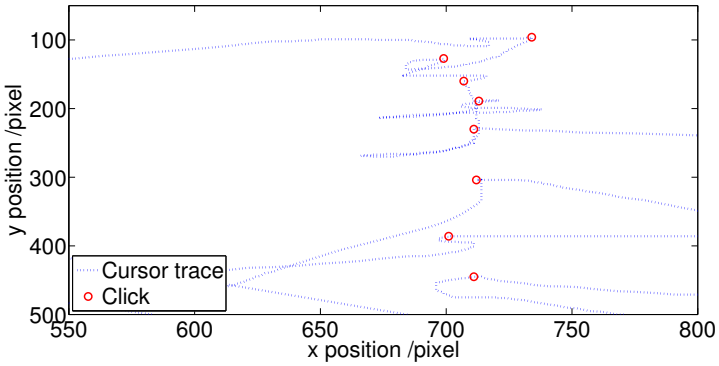
The popularity of tablet devices and laptops with touch screens is increasing. The workers can directly tap on the rating objects without using mouse. In this case, we cannot record any cursor trajectory. In our tasks, we used javascript to check the type of operating system and browser to filter the workers using tablet devices. However,

this approach cannot apply to the workers using laptops equipped with touch screens. We suggest that the experimenters can require the workers to use a mouse in the instructions. In our assessments, no worker used touch screen devices.

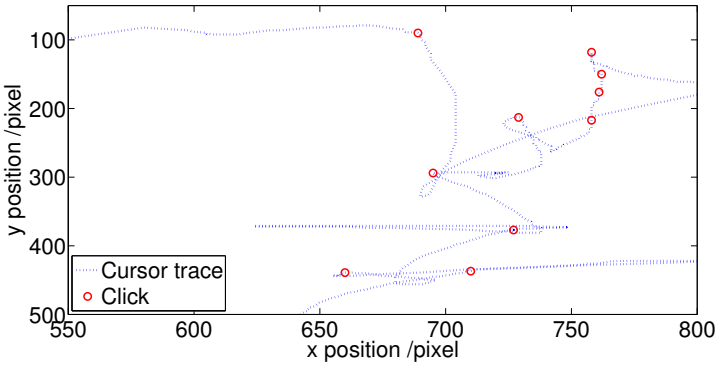
7.6 Summary

This chapter presented a novel worker-behavior based method which could be used to infer the quality of workers. We proposed to extract information from the cursor trajectory and quantify them using a set of worker behavior metrics. In our experiment, we carefully designed a QoE crowdtesting task, such that we could estimate the quality of workers with a quality score. We then correlated the worker behavior metrics with the score using multiclass Naïve Bayes model.

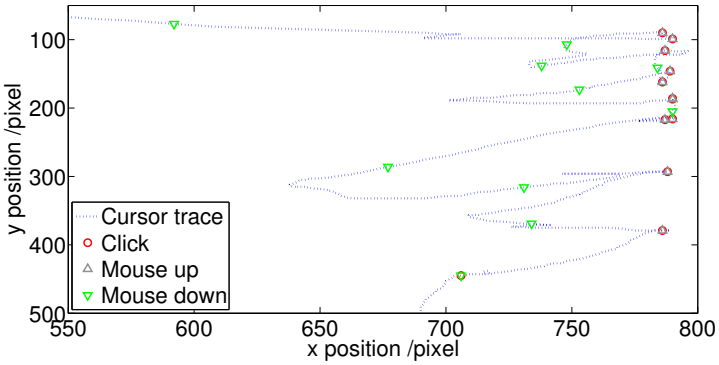
Our results showed that the error rate for the models with three metrics is less than 30%. We also found that different set of metrics should be used for different rating methods. By combining the predictions from four rating methods, the success rate in detecting low-quality workers is around 80%. We further showed that our method outperforms CrowdMOS in the precision and recall.



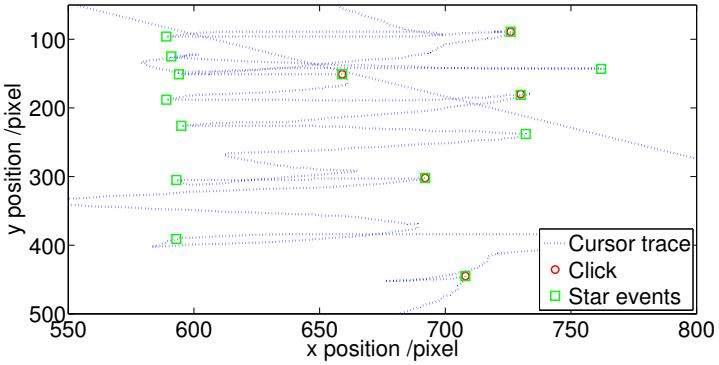
(a) Radio button



(b) Number field



(c) Slide bar



(d) Stars

Figure 7.13: Cursor trace of different rating methods from a typical worker.

Chapter 8

Conclusions and Future Work

This thesis advances HTTP video streaming systems in three directions—QoE measurement, QoE improvement, and QoE assessment.

QoE Measurement We proposed a set of application performance metrics (APM) to quantify the application QoS of HTTP streaming and link the network QoS with the QoE. Through comprehensive testbed experiments and subjective assessments, we revealed that the frequency of the occurrence of rebuffering events impacts the QoE the most. We further investigated the impact of video quality adaptation on the user perceived quality by conducting laboratory-based and crowdsourcing-based subjective assessments. In particular, we found that a sudden drop of quality level can severely degrade the QoE. Instead, switching down to an intermediate level before reaching the target (lower) quality level is preferred. Another finding is that starting from low initial video bitrate impacts on the QoE even if the quality can ramp up to a high quality. We believe that this is because low initial video quality cannot meet user expectation.

QoE Improvement To improve the QoE of HTTP-based video streaming system, we designed and implemented two practical systems `IRate` and `QDASH`. Both systems

enhanced the HAS systems by a measurement middlebox, which can facilitate the implementation of probe optimized network measurement methods on the server side. `IRate` improved the selection of initial video bitrate by conducting pre-stream network measurement before the onset of the video playback. Our design enables browsers to perform lightweight packet-pair based measurement to collect the network path quality. We demonstrated that `IRate` can estimate the BIBR with 80% accuracy in 10s of measurement with a trained decision tree. `IRate` can significantly reduce the rebuffering frequency in HTTP streaming and improve the efficiency and stability in DASH. Our user QoE experiment further showed that the perceived initial quality and the MOS increase by 20.8% and 6.4%, respectively.

After the video starts, the network throughput can be changed. The BIBR may be no longer suitable. Therefore we proposed `QDASH` to improve the QoE of quality adaptation in DASH after the initial playback. First, the `QDASH` middlebox can improve the available bandwidth measurement in HAS. Our novel design allows the measurement middlebox to buffer and dispatch trains of video data packets with different packet sending rates to measure the available bandwidth. We further increased the probing performance by reordering the packets to increase the number of RTT samples and selecting discrete sending rates corresponding to video bitrates. We showed that our method can detect throughput changes rapidly, so that the quality adaptation algorithm can have more time to mitigate steep quality changes. We further proposed a buffer-aware quality adaptation algorithm to mitigate abrupt change in quality level.

QoE Assessment We proposed to apply user-behavior analytics to enhance the video streaming QoE assessments and detect low-quality QoE crowdtesting workers. We first studied how the user-viewing behavior can be useful for inferring the QoE. A novel methodology was proposed to select and analyze the user viewing behavior which is relevant to the impairments in HTTP streaming. The results from subjective as-

sessments showed that including user viewing behavior can significantly improve the explanatory power of predicting the QoE from using only the application layer information.

QoE crowdtesting can compensate the lack of scalability in laboratory based subjective assessments. However, the reliability of workers can be questionable. We therefore proposed a novel cheater detection mechanism to screen out low-quality workers. We proposed a set of worker behavior metrics to systematically quantify the behavior traces collected from the question pages. The trained prediction model can classify low-quality workers without processing the submitted ratings. We also found that using different rating methods the accuracy of our method was about 80%. In our comparison to CrowdMOS, our method reported a higher precision and recall by around 20% and 40%, respectively.

8.1 Future work

Although we believe that our works presented in this thesis have shown the feasibility of a QoE-aware HTTP-based video streaming system, there are still a few issues to explore.

First, in the QoE measurement, we consider the impact of application QoS on the QoE. We will further extend our measurement to the video content. It is because the degradation of QoE may vary for different genre of video under the same set of application QoS. For example, rebuffering events may hurt the QoE more in action movies than news. We will include other factors into the QoE model which are related to the characteristics of video clips, such as Structural Similarity Index (SSIM) [174]. Furthermore, we will construct a model based on the QoE measurement results to improve the bitrate adaptation of the video streaming.

Second, we will investigate the interaction of competing HAS video flows with

QDASH. Because QDASH middlebox bypasses the TCP congestion control, the characteristics of QDASH video flows can be different from other DASH video flows. We will also examine the fairness and stability of QDASH video flows. Furthermore, the QDASH middlebox can manipulate the video flows sending to the same client, so that the competing flows can fairly share the network bandwidth.

Third, video streaming through mobile networks is increasing popular. It is very common nowadays that users watch videos using their smartphones or tablets. The characteristics of the network and the user behavior can be very different from desktop users. Therefore, we will extend our studies to the mobile context in the following two aspects.

1. IRate and QDASH were only evaluated in wired network environment. However, in mobile networks, the probe packets may experience a higher delay overhead in the device [144], which can affect the RTT measurement. Furthermore, throughput estimation in cellular networks can be more challenging, because the timing of packets can be regulated by schedulers in the base station and/or the queuing policy [240, 160, 161]. We will validate the accuracy of the network measurement components of both systems under mobile networks. Other factors include the signal strength, interference in WiFi, network congestion, and so on [69].

2. The user behavior analytics we proposed was focused on desktop environment. However, some types of behavior are not available in the mobile. For example, smartphones and tablets, which are touch screen operated, lack mouse cursor trajectory traces. Instead, new types of user behavior can be triggered, including user interactivity with virtual keyboards [187]. The sensors equipped on smartphones, such as accelerometer, gyroscope, and GPS, can provide rich information about the user behavior. Therefore, we will extend our QoE evaluation by implementing a mobile app [108] and collecting user behavior data.

We will also investigate incorporating historical data to further improve the accuracy and reduce the overhead of IRate. It is common to record the throughput and video streaming performance in the video player. These collected data can validate the BIBR predicted by IRate. Similar to MLASH [56], the corresponding IRate measurement results can be used as a feedback to the model, such that the quality oracle can learn the characteristics of the clients and improve the prediction accuracy.

Finally, we will seek for opportunities to deploy our systems in large-scale video streaming systems. In the meantime, we will explore other possible paradigms and improvements to implement `IRate` and `QDASH`. For example, we can directly modify the video servers to support the network measurement. We can further improve the accuracy of the network measurement by employing new network primitives (e.g., `OMware` [168]). For the middlebox paradigm, we will research the SDN, which is one of the future Internet initiatives. The software running on SDN controllers can measure and intercept the traffic flowing through the SDN switches. This design shares some similarities with our middlebox paradigm. We will investigate embedding network measurement components into the SDN architecture for improving the QoE of video streaming.

Bibliography

[1] Crowdflower. http://www.crowdflower.com/.

[2] IBM SPSS Statistics. http://www.spss.com.

[3] Microworkers. https://microworkers.com/.

[4] Netfilter. http://www.netfilter.org.

[5] Open Source Media Framework (OSMF). http://www.osmf.org.

[6] RateIt plugin for jQuery. http://rateit.codeplex.com/.

[7] Tcpdump/libpcap. http://www.tcpdump.org.

[8] YouTube MySpeed. https://www.youtube.com/my_speed.

[9] YouTube TrueView Video Ads. http://www.youtube.com/advertise/trueview.html.

[10] I. Abdeljaouad, H. Rachidi, S. Fernandes, and A. Karmouch. Performance analysis of modern TCP variants: A comparison of Cubic, Compound and New Reno. In *Proc. QBSC*, 2010.

[11] Abobe. Strobe Media Playback. http://www.osmf.org/strobe_mediaplayback.html.

[12] V. Adhikari, Y. Guo, F. Hao, V. Hilt, Z.-L. Zhang, M. Varvello, and M. Steiner. Measurement study of Netflix, Hulu, and a tale of three CDNs. *IEEE/ACM Trans. Netw.*, 23(6):1984–1997, 2015.

[13] V. K. Adhikari, S. Jain, Y. Chen, and Z.-L. Zhang. Vivisecting YouTube: An active measurement study. In *Proc. IEEE INFOCOM Mini-conference*, 2012.

[14] Adobe. Adobe HTTP Dynamic Streaming. http://help.adobe.com/en_US/HTTPStreaming/1.0/Using/index.html.

[15] Adobe. HTTP Dynamic Streaming. http://www.adobe.com/products/hds-dynamic-streaming/features.html.

[16] S. Ahsan, V. Bajpai, J. Ott, and J. Schönwälder. Measuring YouTube from dual-stacked hosts. In *Proc. PAM*, 2015.

[17] S. Akhshabi, L. Anantakrishnan, C. Dovrolis, and A. Begen. Server-based traffic shaping for stabilizing oscillating adaptive streaming players. In *Proc. ACM NOSSDAV*, 2013.

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

[19] S. Akhshabi, A. C. Begen, and C. Dovrolis. An experimental evaluation of rate-adaptation algorithms in adaptive streaming over HTTP. In *Proc. ACM MMSys*, 2011.

[20] C. Alberti, D. Renzi, C. Timmerer, C. Mueller, S. Lederer, S. Battista, and M. Mattavelli. Automated QoE evaluation of dynamic adaptive streaming over HTTP. In *Proc. QoMEX*, 2013.

[21] Amazon. Mechanical turk. https://www.mturk.com.

[22] P. Ameigeiras, J. J. Ramos-Munoz, J. Navarro-Ortiz, and J. Lopez-Soler. Analysis and modelling of YouTube traffic. *Trans. on Emerging Telecommun. Technologies*, 23(4):360–377, 2012.

[23] D. Antoniades, M. Athanatos, A. Papadogiannakis, E. P. Markatos, and C. Dovrolis. Available bandwidth measurement as simple as running wget. In *Proc. PAM*, 2006.

[24] Apple. HTTP Live Streaming. https://developer.apple.com/streaming/.

[25] T. Arsan. Review of bandwidth estimation tools and application to bandwidth adaptive video streaming. In *Proc. IEEE HONET*, 2012.

[26] S. Avallone, S. Guadagno, D. Emma, A. Pescape, and G. Ventre. D-itg distributed internet traffic generator. In *Proc. QEST*, 2004.

[27] A. Balachandran, V. Sekar, A. Akella, S. Seshan, I. Stoica, and H. Zhang. Developing a predictive model of quality of experience fro Internet video. In *Proc. ACM SIGCOMM*, 2013.

[28] H. Balakrishnan, V. Padmanabhan, G. Fairhurst, and M. Sooriyabandara. RFC3449: TCP performance implications of network path asymmetry, 2002.

[29] W. Bao, V. W. Wong, and V. C. Leung. A model for steady state throughput of TCP CUBIC. In *Proc. IEEE Globecom*, 2010.

[30] S. Benno, J. O. Esteban, and I. Rimac. Adaptive streaming: The network HAS to help. *Beel Labs Technical Journal*, 16(2):101–114, 2011.

[31] A. Biernacki and K. Tutschku. Performance of HTTP video streaming under different network conditions. *Multimed Tools Appl*, 72(2):1143–1166, 2014.

[32] N. Bouten, R. de O. Schmidt, J. Famaey, S. Latré, A. Pras, and F. D. Turck. QoE-driven in-network optimization for adaptive video streaming based on packet sampling measurements. *Computer Networks*, 81:96–115, 2015.

[33] R. Braden. RFC1122: Requirements for Internet Hosts – Communication Layers, 1989.

[34] R. Brant. Assessing proportionality in the proportional odds model for ordinal logistic regression. *Biometrics*, 46(4):1171–1178, 1990.

[35] K. V. Bruno A. A. Nunes and, W. Ballenthin, S. Lukin, and K. Obraczka. A machine learning approach to end-to-end RTT estimation and its application to TCP. In *Proc. IEEE ICCCN*, 2011.

[36] S. Buchholz and J. Latorre. Crowdsourcing preference tests, and how to detect cheating. In *Proc. ISCA INTERSPEECH*, 2011.

[37] P. L. Callet, S. Möller, and A. Perkis. Qualinet white paper on definitions of quality of experience. European Network on Quality of Experience in Multimedia Systems and Services, 2012.

[38] B. Carbunar, R. Potharaju, M. Pearce, and V. Vasudevan. Network aware caching for video on demand systems. In *Proc. IEEE WoWMoM*, 2012.

[39] P. Casas, P. Fiadino, A. Sackl, and A. D'Alconzo. YouTube in the move: Understanding the performance of YouTube in cellular networks. In *Proc. IFIP WD*, 2014.

[40] P. Casas, R. Schatz, and T. Hoßfeld. Monitoring YouTube QoE: Is your mobile network delivering the right experience to your customers? In *Proc. IEEE WCNC*, 2013.

[41] J. Case, M. Fedor, M. Schoffstall, and J. Davin. RFC1157: Simple Network Management Protocol (SNMP), May 1990.

[42] E. Chan, A. Chen, X. Luo, R. Mok, W. Li, and R. Chang. TRIO: Measuring asymmetric capacity with three minimum round-trip times. In *Proc. ACM CoNEXT*, 2011.

[43] Y. Chang, C. Chang, K. Chen, and C. Lei. Radar chart: Scanning for high QoE in QoS dimensions. In *Proc. IEEE CQR*, June 2010.

[44] C. Chen, L. K. Choi, G. de Veciana, C. Caramanis, R. W. H. Jr., and A. C. Bovik. Modeling the time – Varying subjective quality of HTTP video streams with rate adaptations. *IEEE Trans. Image Process.*, 23(5):2206–2221, 2014.

[45] C. Chen, S.-Y. Lee, and H. W. Stevenson. Response style and cross-cultural comparisons of rating scales among East Asian and North American students. *Psychological Science*, 6(3):170–175, 1995.

[46] K.-T. Chen, C.-J. Chang, C.-C. Wu, Y.-C. Chang, and C.-L. Lei. Quadrant of euphoria: a crowdsourcing platform for QoE assessment. *IEEE Netw.*, 24(2):28–35, 2010.

[47] K.-T. Chen, C. C. Tu, and W.-C. Xiao. OneClick: A framework for measuring network quality of experience. In *Proc. IEEE INFOCOM*, 2009.

[48] L. Chen, Y. Zhou, and D. M. Chiu. Video browsing – A study of user behavior in online VoD services. In *Proc. ICCCN*, 2013.

[49] L. Chen, Y. Zhou, and D. M. Chiu. A study of user behavior in online VoD services. *Computer Communications*, 46:66–75, 2014.

[50] M. C. Chen, J. R. Anderson, and M. H. Sohn. What can a mouse cursor tell us more?: Correlation of eye/mouse movements on web browsing. In *Proc. ACM CHI*, 2001.

[51] Y. Chen, Q. Chen, F. Zhang, Q. Zhang, K. Wu, R. Huang, and L. Zhou. Understanding viewer engagement of video service in Wi-Fi network. *Computer Networks*, 91, 2015.

[52] Y. Chen, K. Wu, and Q. Zhang. From QoS to QoE: A tutorial on video quality assessment. *IEEE Commun. Surveys Tuts*, 17(2):1126–1165, 2015.

[53] Y. Chen, B. Zhang, Y. Liu, and W. Zhu. On distribution of user movie watching time in a large-scale video streaming system. In *Proc. IEEE ICC*, 2014.

[54] X. Cheng and M. Fatourechi. Insight data of YouTube from a partner's view. In *Proc. ACM NOSSDAV*, 2014.

[55] X. Cheng, J. Liu, and C. Dale. Understanding the characteristics of internet short video sharing: A YouTube-based measurement study. *IEEE Trans. Multimedia*, 15(5):1184–1194, 2013.

[56] 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 ICME*, 2015.

[57] S. Chikkerur, V. Sundaram, M. Reisslein, and L. J. Karam. Objective video quality assessment methods: A classification, review, and performance comparison. *IEEE Trans. Broadcast.*, 57(2):165–182, 2011.

[58] B. Choi, S. Moon, R. Cruz, Z. Zhang, and C. Diot. Practical delay monitoring for ISPs. In *Proc. ACM CoNEXT*, 2005.

[59] Cisco. Cisco visual networking index: Global mobile data traffic forecast update 2014-2019 white paper. `http://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/white_paper_c11-520862.html`.

[60] G. Costagliola, V. Fuccella, M. Giordano, and G. Polese. Logging and visualization of learner behaviour in web-based e-testing. In *Proc. ICWL*, 2007.

[61] N. Cranley, P. Perry, and L. Murphy. User perception of adapting video quality. *International Journal of Human-Computer Studies*, 64(8):637 – 647, 2006.

[62] A. Dainotti, A. Botta, and A. Pescapè. A tool for the generation of realistic network workload for emerging networking scenarios. *Computer Networks*, 56(15):3531–3547, 2012.

[63] L. De Cicco and S. Mascolo. An experimental investigation of the Akamai adaptive video streaming. In *Proc. USAB*, 2010.

[64] L. De Cicco, S. Mascolo, and V. Palmisano. Feedback control for adaptive live video streaming. In *Proc. ACM MMSys*, 2011.

[65] E. Dedu, W. Ramadan, and J. Bourgeois. A taxonomy of the parameters used by decision methods for adaptive video transmission. *Multimed Tools Appl*, 74(9):2963–2989, 2015.

[66] G. Demartini, D. E. Difallah, and P. Cudré-Mauroux. ZenCrowd: Leveraging probabilistic reasoning and crowdsourcing techniques for large-scale entity linking. In *Proc. ACM WWW*, 2012.

[67] M. Dhawan, J. Samuel, R. Teixeira, C. Kreibich, M. Allman, N. Weaver, and V. Paxson. Fathom: A browser-based network measurement platform. In *Proc. ACM/USENIX IMC*, 2012.

[68] D. E. Difallah, G. Demartini, and P. Cudré-Mauroux. Mechanical cheat: Spamming schemes and adversarial techniques on crowdsourcing platforms. In *Proc. ACM CrowdSearch*, 2012.

[69] G. Dimopoulos, I. Leontiadis, P. Barlet-Ros, K. Papagiannaki, and P. Steenkiste. Identifying the root cause of video streaming issues on mobile devices. In *Proc. ACM CoNEXT*, 2015.

[70] F. Dobrian, A. Awan, D. Joseph, A. Ganjam, J. Zhan, V. Sekar, I. Stoica, and H. Zhang. Understanding the impact of video quality on user engagement. In *Proc. ACM SIGCOMM*, 2011.

[71] C. Dovrolis, P. Ramanathan, and D. Moore. Packet dispersion techniques and a capacity-estimation methodology. *IEEE/ACM Trans. Netw.*, 12(6):963–977, 2004.

[72] J. S. Downs, M. B. Holbrook, S. Sheng, and L. F. Cranor. Are your participants gaming the system?: Screening mechanical turk workers. In *Proc. ACM CHI*, 2010.

[73] S. Egger, B. Gardlo, M. Seufert, and R. Schatz. The impact of adaptation strategies on perceived quality of HTTP adaptive streaming. In *Proc. ACM VideoNext*, 2014.

[74] C. Eickhoff and A. P. de Vries. Increasing cheat robustness of crowdsourcing tasks. *Inf Retrieval*, 16(2):121–137, 2013.

[75] C. Eickhoff, C. G. Harris, P. Srinivasan, and A. P. de Vries. Quality through flow and immersion: Gamifying crowdsourced relevance assessments. In *Proc. ACM SIGIR*, 2012.

[76] Endace. DAG packet capture cards. `http://www.endace.com`.

[77] A. E. Essaili, D. Schroeder, D. Staehle, M. Shehada, W. Kellerer, and E. Steinbach. Quality-of-Experience driven adaptive HTTP media delivery. In *Proc. IEEE ICC*, 2013.

[78] K. Evensen, D. Kaspar, C. Griwodz, P. Halvorsen, A. F. Hansen, and P. Engelstad. Using bandwidth aggregation to improve the performance of quality-adaptive streaming. *Signal Processing: Image Communication*, 27(4):312–312, 2012.

[79] J. Fardous and S. Kanhere. On the use of location window in geo-intelligent HTTP adaptive video streaming. In *Proc. IEEE ICON*, 2012.

[80] A. Finamore, M. Mellia, M. M. M. abd Ruben Torres, and S. G. Rao. YouTube Everywhere: Impact of device and infrastructure synergies on user experience. In *Proc. ACM/USENIX IMC*, 2011.

[81] A. Ganjam, J. Jiang, X. Liu, V. Sekar, F. Siddiqui, I. Stoica, J. Zhan, and H. Zhang. C3: Internet-scale control plane for video quality optimization. In *Proc. USENIX NSDI*, 2015.

[82] B. Gardlo, S. Egger, M. Seufert, and R. Schatz. Crowdsourcing 2.0: Enhancing execution speed and reliability of web-based QoE testing. In *Proc. IEEE ICC*, 2014.

[83] P. Georgopoulos, M. Broadbent, A. Farshad, B. Plattner, and N. Race. Using software defined networking to enhance the delivery of Video-on-Demand. *Computer Communications*, 69:79–87, 2015.

[84] P. Georgopoulos, Y. Elkhatib, M. Broadbent, M. Mu, and N. Race. Towards network-wide QoE fairness using openflow-assisted adaptive video streaming. In *Proc. ACM FhMN*, 2013.

[85] M. Ghobadi, Y. Cheng, A. Jain, and M. Mathis. Trickle: Rate limiting YouTube video streaming. In *Proc. USENIX ATC*, 2012.

[86] P. Gill, M. Arlitt, Z. Li, and A. Mahanti. YouTube traffic characterization: A view from the edge. In *Proc. ACM IMC*, 2007.

[87] P. Gill, M. Arlitt, Z. Li, and A. Mahanti. Characterizing user sessions on YouTube. In *Proc. MMCN*, 2008.

[88] O. Goga and R. Teixeira. Speed measurements of residential Internet access. In *Proc. PAM*, 2012.

[89] Google. Video quality report. `https://www.google.com/get/videoqualityreport/`.

[90] S. Ha, I. Rhee, and L. Xu. CUBIC: A new TCP-friendly high-speed TCP variant. *SIGOPS Oper. Syst. Rev.*, 42(5):64–74, 2008.

[91] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten. The WEKA data mining software: An update. *ACM SIGKDD Explorations*, 11(1):10–18, 2009.

[92] J. Hao, R. Zimmermann, and H. Ma. GTube: Geo-predictive video streaming over HTTP in mobile environment. In *Proc. ACM MMSys*, 2014.

[93] M. Hirth, S. Scheuring, T. Hoßfeld, C. Schwartz, and P. Tran-Gia. Predicting result quality in crowdsourcing using application layer monitoring. In *Proc. IEEE ICCE*, 2014.

[94] 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. QoMEX*, 2012.

[95] T. Hoßfeld, C. Keimel, M. Hirth, B. Gardlo, J. Habigt, K. Diepold, and P. Tran-Gia. Best practices for QoE crowdtesting: QoE assessment with crowdsourcing. *IEEE Trans. Multimedia*, 16(2):541–558, 2014.

[96] T. Hoßfeld, R. Schatz, and U. R. Krieger. QoE of YouTube video streaming for current Internet transport protocol. In *Proc. MMB&DFT*, 2014.

[97] T. Hoßfeld, M. Seufert, M. Hirth, T. Zinner, P. Tran-Gia, and R. Schatz. Quantification of YouTube QoE via crowdsourcing. In *Proc. IEEE ISM*, 2011.

[98] T. Hoßfeld, M. Seufert, C. Sieber, and T. Zinner. Assessing effect sizes of influence factors towards a QoE model for HTTP adaptive streaming. In *Proc. QoMEX*, 2014.

[99] T. Hoßfeld, L. Skorin-Kapov, Y. Haddad, P. Pocta, V. A. Siris, A. Zgank, and H. Melvin. Can context monitoring improve QoE? a case study of video flash crowds in the internet of services. In *Proc. IFIP/IEEE QCMan*, 2015.

[100] R. Houdaille and S. Gouache. Shaping HTTP adaptive streams for a better user experience. In *Proc. ACM MMSys*, 2012.

[101] J. K. Hsu. Startup bitrate in adaptive bitrate streaming, August 2013.

[102] N. Hu and P. Steenkiste. Evaluation and characterization of available bandwidth probing techniques. *IEEE JSAC*, 21(6):879–894, 2003.

[103] N. Hu and P. Steenkiste. Evaluation and characterization of available bandwidth techniques. *IEEE JSAC*, 21(6):879–894, 2003.

[104] C. Huang, J. Li, and K. W. Ross. Can Internet Video-on-Demand be profitable? In *Proc. ACM SIGCOMM*, 2007.

[105] T.-Y. Huang, N. Handigol, B. Heller, N. McKeown, and R. Johari. Confused, timid, and unstable: Picking a video streaming rate is hard. In *Proc. ACM/USENIX IMC*, 2012.

[106] 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 SIGCOMM*, 2014.

[107] F. Hwang, S. Keates, P. Langdon, and J. Clarkson. A submovement analysis of cursor trajectories. *Behaviour & Information Technology*, 24(3):205–217, 2005.

[108] S. Ickin, M. Fiedler, K. Wac, P. Arlos, C. Temiz, and K. Mkocha. VLQoE: Video QoE instrumentation on the smartphone. *Multimed Tools Appl*, 74(2):381–411, 2014.

[109] Interactive Advertising Bureau. Digital video in-stream Ad format guidelines and best practices. `http://www.iab.net/media/file/IAB-Video-Ad-Format-Standards.pdf`, May 2008.

[110] ITU-R Recommendation BT.500-12. *Methodology for the subjective assessment of the quality of television pictures*, January 2012.

[111] ITU-T Recommendation P.10/G.100. *Vocabulary for performance and quality of service*, amendment 1 edition, 2006.

[112] ITU-T Recommendation P.800. *Methods for subjective determination of transmission quality*, August 1996.

[113] ITU-T Recommendation P.910. *Subjective video quality assessment methods for multimedia applications*, April 2008.

[114] ITU-T Recommendation P.911. *Subjective audiovisual quality assessment methods for multimedia applications*, December 1998.

[115] M. Jain and C. Dovrolis. End-to-end available bandwidth: measurement methodology, dynamics, and relation with TCP throughput. *IEEE/ACM Trans. Netw.*, 11:537–549, 2003.

[116] D. Jarnikov and T. Özçelebi. Client intelligence for adaptive streaming solutions. *Signal Processing: Image Communication*, 2011.

[117] J. Jiang, V. Sekar, I. Stoica, and H. Zhang. Shedding light on the structure of Internet video quality problems in the wild. In *Proc. ACM CoNEXT*, 2013.

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

[119] M. Joglekar, H. Garcia-Molina, and A. Parameswaran. Evaluating the crowd with confidence. In *Proc. ACM KDD*, 2013.

[120] P. Juluri, V. Tamarapalli, and D. Medhi. Measurement of Quality of Experience of Video-on-Demand services: A survey. *IEEE Commun. Surveys Tuts*, 18(1):401–418, 2015.

[121] P. Juluri, V. Tamarapalli, and D. Medhi. SARA: Segment aware rate adaptation algorithm for dynamic adaptive streaming over HTTP. In *Proc. IEEE QoE-FI*, 2015.

[122] M. Kaplan, M. Zeljkovic, M. Claypool, and C. Wills. How's My Network? Predicting performance from within a web browser sandbox. In *Proc. IEEE LCN*, 2012.

[123] R. Karki, T. Seenivasan, M. Claypool, and R. Kinicki. Performance analysis of home streaming video using Orb. In *Proc. ACM NOSSDAV*, 2010.

[124] G. Kazai, J. Kamps, and N. Milic-Frayling. An analysis of human factors and label accuracy in crowdsourcing relevance judgments. *Inf Retrieval*, 16(2):138–178, 2013.

[125] C. Keimel, J. Habigt, and C. Horch. Video quality evaluation in the cloud. In *Proc. IEEE PV*, 2012.

[126] C. Keimel, J. Habigt, C. Horch, and K. Diepold. QualityCrowd - A framework for crowd-based quality evaluation. In *Proc. IEEE Picture Coding Symposium*, 2012.

[127] I. E. Khayat, P. Geurts, and G. Leduc. Machine-learnt versus analytical models of TCP throughput. *Computer Networks*, 51(10):2631–2644, 2006.

[128] T. Kim and M. Ammar. Receiver buffer requirement for video streaming over TCP. In *Proc. SPIE Visual Communications and Image Processing*, 2006.

[129] L. Kleinrock. *Queueing Systems. Volume 1: Theory*. Wiley-Interscience, 1975.

[130] J. W. Kleinrouweler, S. Cabrero, R. van der Mei, and P. Cesar. Modeling stability and bitrate of network-assisted HTTP adaptive streaming players. In *Proc. ITC*, 2015.

[131] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. F. Kaashoek. The click modular router. *ACM Trans. Comput. Syst.*, 18(3):263–297, 2000.

[132] S. Kraft and U. Zölzer. BeaqleJS: HTML5 and javascript based framework for the subjective evaluation of audio quality. In *Proc. Linux Audio Conference*, 2014.

[133] C. Kreibich, N. Weaver, B. Nechaev, and V. Paxson. Netalyzr: Illuminating the edge network. In *Proc. ACM/USENIX IMC*, 2010.

[134] V. Krishnamoorthi, N. Carlsson, D. Eager, A. Mahanti, and N. Shahmehri. Helping hand or hidden hurdle: Proxy-assisted HTTP-based adaptive streaming performance. In *Proc. IEEE MASCOTS*, 2013.

[135] V. Krishnamoorthi, N. Carlsson, D. Eager, A. Mahanti, and N. Shahmehri. Quality-adaptive prefetching for interactive branched video using HTTP-based adaptive streaming. In *Proc. ACM MM*, 2014.

[136] V. Krishnamoorthi, N. Carlsson, D. Eager, A. Mahanti, and N. Shahmehri. Bandwidth-aware prefetching for proactive multi-video preloading and improved HAS performance. In *Proc. ACM MM*, 2015.

[137] S. S. Krishnan and R. K. Sitaraman. Video stream quality impacts viewer behavior: Inferring causality using quasi-experimental designs. In *Proc. ACM IMC*, 2012.

[138] S. S. Krishnan and R. K. Sitaraman. Understanding the effectiveness of video ads: A measurement study. In *Proc. ACM/USENIX IMC*, 2013.

[139] S. S. Krishnan and R. K. Sitaraman. Video stream quality impacts viewer behavior: Inferring causality using quasi-experimental designs. *IEEE Trans. Netw.*, 21(6):2001–2014, 2013.

[140] R. Kuschnig, I. Kofler, and H. Hellwagner. Improving internet video streaming performance by parallel TCP-based request-response streams. In *Proc. IEEE CCNC*, 2010.

[141] D. H. Lee, C. Dovrolis, and A. C. Begen. Caching in HTTP adaptive streaming: Friend or foe? In *Proc. ACM NOSSDAV*, 2014.

[142] M. Levkov. *VIdeo encoding and transcoding recommendations for HTTP Dynamic Streaming on the Adobe Flash Platform*. Adobe, Oct 2010.

[143] W. Li, R. Mok, R. Chang, and W. Fok. Appraising the delay accuracy in browser-based network measurement. In *Proc. ACM/USENIX IMC*, 2013.

[144] W. Li, R. Mok, D. Wu, and R. Chang. On the accuracy of smartphone-based mobile network measurement. In *Proc. IEEE INFOCOM*, 2015.

[145] Z. Li, A. C. Begen, J. Gahm, Y. Shan, B. Osler, and D. Oran. Streaming video over HTTP with consistent quality. In *Proc. ACM MMSys*, 2014.

[146] Z. Li, J. Lin, M.-I. Akodjenou, G. Xie, M. A. Kaafar, Y. Jin, and G. Peng. Watching videos from everywhere: A study of the PPTV mobile VoD system. In *Proc. ACM IMC*, 2012.

[147] Z. Li, Q. Wu, K. Salamatian, and G. Xie. Video delivery performance of a large-scale VoD system and the implications on content delivery. *IEEE Trans. Multimedia*, 17(6):880–892, 2015.

[148] Z. Li, G. Xie, J. Lin, Y. Jin, M.-A. Kaafar, and K. Salamatian. On the geographic patterns of a large-scale mobile Video-on-Demand system. In *Proc. IEEE INFOCOM*, 2014.

[149] 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 JSAC*, 32(4):719–733, 2014.

[150] K. Liang, J. Hao, R. Zimmermann, and D. K. Yau. Integrated prefetching and caching for adaptive video streaming over HTTP: An online approach. In *Proc. ACM MMSys*, 2015.

[151] A. Lie and J. Klaue. Evalvid-RA: trace driven simulation of rate adaptive MPEG-4 VBR video. *ACM Multimedia Systems Journal*, 14:33–50, 2008.

[152] C. Liu, I. Bouazizi, and M. Gabbouj. Rate adaptation for adaptive HTTP streaming. In *Proc. ACM MMSys*, 2011.

[153] X. Liu, F. Dobrian, H. Milner, J. Jiang, V. Sekar, I. Stoica, and H. Zhang. A case for a coordinated Internet video control plane. In *Proc. ACM SIGCOMM*, 2012.

[154] X. Liu and A. Men. QoE-aware traffic shaping for HTTP adaptive streaming. *IJMUE*, 9(2):33–44, 2014.

[155] Y. Liu, Y. Shen, Y. Mao, J. Liu, Q. Lin, and D. Yang. A study on quality of experience for adaptive streaming service. In *Proc. IEEE IIMC*, 2013.

[156] Y. Liu, Q. Wei, L. Guo, B. Shen, S. Chen, and Y. Lan. Investigating redundant Internet video streaming traffic on iOS devices: Causes and solutions. *IEEE Trans. Multimedia*, 16(2):510–520, 2014.

[157] X. Luo, E. Chan, and R. Chang. Design and implementation of TCP data probes for reliable and metric-rich network path monitoring. In *Proc. USENIX ATC*, 2009.

[158] J. Martin, Y. Fu, N. Wourms, and T. Shaw. Characterizing Netflix bandwidth consumption. In *Proc. IEEE CCNC*, 2013.

[159] D. E. Meyer, R. A. Abrams, S. Kornblum, C. E. Wright, and J. E. Keith Smith. Optimality in human motor performance: Ideal control of rapid aimed movements. *Psychological Review*, 95(3):340–370, 1988.

[160] F. Michelinakis, N. Bui, G. Fioravantti, J. Widmer, F. Kaup, and D. Hausheer. Lightweight mobile bandwidth availability measurement. In *Proc. IFIP Networking*, 2015.

[161] F. Michelinakis, G. Kreitz, R. Petrocco, B. Zhang, and J. Widmer. Passive mobile bandwidth classification using short lived TCP connections. In *Proc. IFIP WMNC*, 2015.

[162] Microsoft. Smooth streaming. `http://www.iis.net/downloads/microsoft/smooth-streaming`.

[163] Mikkel. Tips to enhance Dailymotion HQ/HD video viewing. `http://geekfp.com/tips-to-enhance-dailymotion-hqhd-video-viewing/`.

[164] M. Mirza, J. Sommers, P. Barford, and X. Zhu. A machine learning approach to TCP throughput prediction. *IEEE/ACM Trans. Netw.*, 18(4):1026–1039, Aug. 2010.

[165] R. Mok, E. Chan, and R. Chang. Measuring the quality of experience of HTTP video streaming. In *Proc. IFIP/IEEE IM (pre-conf)*, 2011.

[166] R. Mok, E. Chan, X. Luo, and R. Chang. Inferring the QoE of HTTP video streaming from user-viewing activities. In *Proc. of ACM SIGCOMM W-MUST*, 2011.

[167] R. Mok, W. Li, and R. Chang. Detecting low-quality crowdtesting workers. In *Proc. IEEE/ACM IWQoS*, 2015.

[168] R. Mok, W. Li, and R. Chang. Improving the packet send-time accuracy in embedded devices. In *Proc. PAM*, 2015.

[169] R. Mok, W. Li, and R. Chang. IRate: Initial video bitrate selection system for HTTP streaming. *IEEE JSAC*, In press, 2016.

[170] R. Mok, X. Luo, E. Chan, and R. Chang. QDASH: A QoE-aware DASH system. In *Proc. ACM MMSys*, 2012.

[171] H. G. Msakni and H. Youssef. Impact of non-QoS related parameters on video QoE. In *Proc. PQS*, 2013.

[172] M. Mu, A. Mauthe, G. Tyson, and E. Cerqueira. Statistical analysis of ordinal user opinion scores. In *Proc. IEEE CCNC*, 2012.

[173] C. Müller, S. Lederer, and C. Timmerer. An evaluation of dynamic adaptive streaming over HTTP in vehicular environments. In *Proc. ACM MoViD*, 2012.

[174] D. Munaretto, D. Zucchetto, A. Zanella, and M. Zorzi. Data-driven QoE optimization techniques for multi-user wireless networks. In *Proc. IEEE ICNC*, 2015.

[175] N. J. D. Nagelkerke. A note on a general definition of the coefficient of determination. *Biometrika*, 78 Issue 3:691–692, 1991.

[176] Netflix. The ISP speed index from Netflix. `http://ispspeedindex.netflix.com/`.

[177] Netflix. Overview. `http://ir.netflix.com/`.

[178] I. Netflix. Manage bandwidth usage. `https://support.netflix.com/en/node/87`.

[179] P. Ni, R. Eg, A. Eichhorn, C. Griwodz, and P. Halvorsen. Spatial flicker effect in video scaling. In *Proc. IEEE QoMEX*, 2011.

[180] O. Ognenoski, M. G. Martini, and P. Amon. Segment-based teletraffic model for MPEG-DASH. In *Proc. IEEE MMSP*, 2013.

[181] Ookla. Speedtest. `http://www.speedtest.net/`.

[182] Ookla. Speedtest.net mini. `http://www.speedtest.net/mini.php`.

[183] Óscar Figuerola Salas, V. Adzic, A. Shah, and H. Kalva. Assessing Internet video quality using crowdsourcing. In *Proc. ACM CrowdMM*, 2013.

[184] J. Padhye, V. Firoiu, D. Towsley, and J. Kurose. Modeling TCP Reno performance: A simple model and its empirical validation. *IEEE/ACM Trans. Netw.*, 8(2):133–145, 2000.

[185] G. Paolacci, J. Chandler, and P. Ipeirotis. Running experiments on Amazon Mechanical Turk. *Judgment and Decision Making*, 5(5):411–419, 2010.

[186] P. Papageorge, J. McCann, and M. Hicks. Passive aggressive measurement with MGRP. In *Proc. ACM SIGCOMM*, 2009.

[187] A. Patro, S. Rayanchu, M. Griepentrog, Y. Ma, and S. Banerjee. Capturing mobile experience in the wild: A tale of two apps. In *Proc. ACM CoNEXT*, 2013.

[188] V. Paxson. End-to-end routing behavior in the Internet. *IEEE/ACM Trans. Netw.*, 5(5):601–615, 1997.

[189] S. Petrangeli, T. Wauters, R. Huysegems, T. Bostoen, and F. D. Turck. Network-based dynamic prioritization of HTTP adaptive streams to avoid video freezes. In *Proc. IFIP/IEEE QCMan*, 2015.

[190] J. G. Phillips and T. J. Triggs. Characteristics of cursor trajectories controlled by the computer mouse. *Ergonomics*, 44(5):527–536, 2001.

[191] M. H. Pinson and S. Wolf. Comparing subjective video quality testing methodologies. *Visual Communications and Image Processing 2003*, 5150(1):573–582, 2003.

[192] L. Plissonneau and E. Biersack. A longitudinal view of HTTP video streaming performance. In *Proc. ACM MMSys*, 2012.

[193] J. R. Quinlan. *C4.5: programs for machine learning*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1993.

[194] B. Rainer and C. Timmerer. Quality of experience of Web-based adaptive HTTP streaming clients in real-world environments using crowdsourcing. In *Proc. ACM VideoNext*, 2014.

[195] B. Rainer and C. Timmerer. A subjective evaluation using crowdsourcing of adaptive media playout utilizing audio-visual content features. In *IEEE/IFIP NOMS*, 2014.

[196] B. Rainer, M. Waltl, and C. Timmerer. A Web based subjective evaluation platform. In *Proc. QoMEX*, 2013.

[197] J. J. Ramos-munoz, J. Prados-Garzon, P. Ameigeiras, J. Navarro-Ortiz, and J. M. Lopez-soler. Characteristics of mobile youTube traffic. *IEEE Wireless Commun.*, 21(1):18–25, 2014.

[198] A. Rao, Y. sup Lim, C. Barakat, A. Legout, D. Towsley, and W. Dabbous. Network characteristics of video streaming traffic. In *Proc. ACM CoNEXT*, 2011.

[199] J. Redi, T. Hoßfeld, P. Korshunov, F. Mazza, I. Povoa, and C. Keimel. Crowdsourcing-based multimedia subjective evaluations: A case study on image recognizability and aesthetic appeal. In *Proc. ACM CrowdMM*, 2013.

[200] J. Rennie, L. Shih, J. Teevan, and D. Karger. Tackling the poor assumptions of Naive Bayes text classifiers. In *Proc. IMLS ICML*, 2003.

[201] F. Ribeiro, D. Florencio, and V. Nascimento. Crowdsourcing subjective image quality evaluation. In *Proc. IEEE ICIP*, 2011.

[202] F. Ribeiro, D. Florêncio, C. Zhang, and M. Seltzer. CrowdMOS: An approach for crowdsourcing mean opinion score studies. In *Proc. IEEE ICASSP*, 2011.

[203] V. Ribeiro, R. Riedi, R. Baraniuk, J. Navratil, and L. Cottrell. pathChirp: Efficient available bandwidth estimation for network paths. In *Proc. PAM*, 2003.

[204] H. Riiser, Håkon, S. Bergsaker, P. Vigmostad, P. Halvorsen, and C. Griwodz. A comparison of quality scheduling in commercial adaptive HTTP streaming solutions on a 3G network. In *Proc. ACM MoVid*, 2012.

[205] H. Riiser, P. Vigmostad, C. Griwodz, and P. Halvorsen. Bitrate and video quality planning for mobile streaming scenarios using a GPS-based bandwidth lookup service. In *IEEE ICME*, 2011.

[206] J. M. Rzeszotarski and A. Kittur. Instrumenting the crowd: Using implicit behavioral measures to predict task performance. In *Proc. UIST*, 2011.

[207] J. M. Rzeszotarski and A. Kittur. CrowdScape: Interactively visualizing user behavior and output. In *Proc. UIST*, 2012.

[208] sandvine. The global Internet phenomena report: 2H 2014. `https://www.sandvine.com/downloads/general/global-internet-phenomena/2014/2h-2014-global-internet-phenomena-report.pdf`, 2014.

[209] F. Sani and J. Todman. *Experimental Design and Statistics For Psychology: A First Course*. Blackwell, 2006.

[210] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson. RFC3550: RTP: A transport protocol for real-time applications, July 2003.

[211] H. Schulzrinne, A. Rao, and R. Lanphier. RFC2326: Real time streaming protocol (RTSP), April 1998.

[212] H. Schwarz, D. Marpe, and T. Wiegand. Overview of the scalable video coding extension of the H.264/AVC standard. *IEEE Trans. on Circuits and Systems for Video Technology*, 17(9):1103 – 1120, 2007.

[213] V. Sekar, N. Egi, S. Ratnasamy, M. K. Reiter, and G. Shi. Design and implementation of a consolidated middlebox architecture. In *Proc. USENIX NSDI*, 2012.

[214] 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 Commun. Surveys Tuts*, 17(1):469–492, 2015.

[215] M. Z. Shafiq, J. Erman, L. Ji, A. X. Liu, J. Pang, and J. Wang. Understanding the impact of network dynamics on mobile video user engagement. In *Proc. ACM SIGMETRICS*, 2014.

[216] 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 CCNC*, 2012.

[217] I. Sodagar. The MPEG-DASH standard for multimedia streaming over the Internet. *IEEE Multimedia*, 18(4):62–67, 2011.

[218] M. Soleymani and M. Larson. Crowdsourcing for affective annotation of video: Development of a viewer-reported boredom corpus. In *Proc. ACM CSE*, 2010.

[219] J.-H. Song and K. Nakayama. Hidden cognitive states revealed in choice reaching tasks. *Trends in Cognitive Sciences*, 13(8):360–366, 2009.

[220] N. Staelens, J. D. Meulenaere, M. Claeys, G. V. Wallendael, W. V. den Broeck, J. D. Cock, R. V. de Walle, P. Demeester, and F. D. Turck. Subjective quality assessment of longer duration video sequences delivered over HTTP adaptive streaming to tablet devices. *IEEE Trans. Broadcast.*, 60(4):707–714, 2014.

[221] T. Stockhammer. Dynamic adaptive streaming over HTTP: Standards and design principles. In *Proc. ACM MMSys*, 2011.

[222] J. Strauss, D. Katabi, and F. Kaashoek. A measurement study of available bandwidth estimation tools. In *Proc. ACM IMC*, 2003.

[223] D. Suh, I. Jang, and S. Pack. QoE-enhanced adaptation algorithm over DASH for multimedia streaming. In *Proc. ICOIN*, 2014.

[224] S. Sundaresan, W. de Donato, N. Feamster, R. Teixeira, S. Crawford, and A. Pescapé. Broadband Internet performance: a view from the gateway. In *Proc. ACM SIGCOMM*, 2011.

[225] K. Tan and J. Song. Compound TCP: A scalable and TCP-friendly congestion control for high-speed networks. In *Proc. PFLDNet*, 2006.

[226] H. Tani and T. Nunome. QoE assessment of a web-based streaming service in seeking operation. In *Proc. ICOIN*, 2014.

[227] S. Tasaka and Y. Ishibashi. Mutually compensatory property of multimedia QoS. In *Proc. IEEE ICC*, 2002.

[228] T. C. Thang, Q.-D. Ho, J. W. Kang, and A. T. Pham. Adaptive streaming of audiovisual content using MPEG DASH. *IEEE Trans. Consum. Electron.*, 58(1):78–85, 2012.

[229] T. C. Thang, A. T. Pham, H. X. Nguyen, P. L. Cuong, and J. W. Kang. Video streaming over HTTP with dynamic resource estimation. *KICS JCN*, 15(6):635–644, 2013.

[230] G. Tian and Y. Liu. Towards Agile and Smooth Video Adaptation in Dynamic HTTP Streaming. In *Proc. ACM CoNEXT*, 2012.

[231] B. J. Villa, P. E. Heegaard, and A. Instefjord. Improving fairness for adaptive HTTP video streaming. In *Proc. IFIP EUNICE*, 2012.

[232] J. D. Vriendt, D. D. Vleeschauwer, and D. Robinson. Model for estimating QoE of video delivered using HTTP adaptive streaming. In *Proc. IFIP/IEEE IM*, 2013.

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

[234] C. Wang and M. Zink. On the feasibility of DASH streaming in the cloud. In *Proc. ACM NOSSDAV*, 2014.

[235] Z. Wang, L. Sun, C. Wu, W. Zhu, and S. Yang. Joint online transcoding and geo-distributed delivery for dynamic adaptive streaming. In *Proc. IEEE INFOCOM*, 2014.

[236] C.-C. Wu, K.-T. Chen, Y.-C. Chang, and C.-L. Lei. Crowdsourcing multimedia QoE evaluation: A trusted framework. *IEEE Trans. Multimedia*, 15(5):1121–1137, 2013.

[237] W. Wu, A. Arefin, R. Rivas, K. Nahrstedt, R. Sheppard, and Z. Yang. Quality of experience in distributed interactive multimedia environments: Toward a theoretical framework. In *Proc. ACM Multimedia*, 2009.

[238] M. Xing, S. Xiang, and L. Cai. A real-time adaptive algorithm for video streaming over multiple wireless access networks. *IEEE JSAC*, 32(4):795–805, 2014.

[239] X. Xing, J. Dang, S. Mishra, and X. Liu. A highly scalable bandwidth estimation of commercial hotspot access points. In *Proc. IEEE INFOCOM*, 2011.

[240] Y. Xu, Z. Wang, W. K. Leong, and B. Leong. An end-to-end measurement study of modern cellular data networks. In *Proc. PAM*, 2014.

[241] J. Yao, S. S. Kanhere, I. Hossain, and M. Hassan. Empirical evaluation of HTTP adaptive streaming under vehicular mobility. In *Proc. IFIP Networking*, 2011.

[242] Y.-C. Yen, C.-Y. Chu, S.-L. Yeh, H.-H. Chu, and P. Huang. Lab experiment vs. crowdsourcing: A comparative user study on Skype call quality. In *Proc. ACM AINTEX*, 2013.

[243] X. Yin, A. Jindal, V. Sekar, and B. Sinopoli. A control-theoretric approach for dynamic adaptive video streaming over HTTP. In *Proc. ACM SIGCOMM*, 2015.

[244] YouTube. Change video quality. `http://support.google.com/youtube/bin/answer.py?hl=en&answer=91449`.

[245] YouTube. Statistics. `https://www.youtube.com/yt/press/statistics.html`.

[246] L. Zhan, D. M. Chiu, Y. Hua, and Z. Zhu. A measurement study of mobile video streaming by different types of devices. In *Proc. IEEE COMSNETS*, 2015.

[247] W. Zhang, Y. Wen, Z. Chen, and A. Khisti. QoE-driven cache management for HTTP adaptive bit rate (ABR) streaming over wireless networks. In *Proc. IEEE Globecom*, 2012.

[248] Y. Zhang, V. Paxson, and S. Shenker. The stationarity of internet path properties: Routing, loss, and throughput. Technical report, ACIRI, 2000.

[249] M. Zink, O. Künzel, J. Schmitt, and R. Steinmetz. Subjective impression of variations in layer encoded videos. In *Proc. IEEE IWQoS*, 2003.

[250] M. Zink, K. Suh, Y. Gu, and J. Kurose. Watch global, cache local: YouTube network traffic at a campus network: measurements and implications. In *Proc. SPIE*, 2008.

[251] X. K. Zou, J. Erman, V. Gopalakrishnan, E. Halepovic, R. Jana, X. Jin, J. Rexford, and R. K. Sinha. Can accurate predictions improve video streaming in cellular networks? In *Proc. ACM HotMobile*, 2015.