# MEASUREMENT AND IMPROVEMENT OF QUALITY-OF-EXPERIENCE FOR ONLINE VIDEO STREAMING SERVICES

A Dissertation
IN
Telecommunications and Computer Networking
and
Electrical and Computer Engineering

Presented to the Faculty of the University
of Missouri–Kansas City in partial fulfillment of
the requirements for the degree

DOCTOR OF PHILOSOPHY

by
PARIKSHIT JULURI

M. S., University of Missouri–Kansas City, Kansas City, USA, 2008
B. Tech., Jawaharlal Nehru Technological University, Hyderabad, India, 2007

Kansas City, Missouri
2015

# MEASUREMENT AND IMPROVEMENT OF QUALITY-OF-EXPERIENCE FOR ONLINE VIDEO STREAMING SERVICES

Parikshit Juluri, Candidate for the Doctor of Philosophy Degree

University of Missouri–Kansas City, 2015

## ABSTRACT

HTTP based online video streaming services have been consistently dominating the online traffic for the past few years. Measuring and improving the performance of these services is an important challenge. Traditional Quality-of-Service (QoS) metrics such as packet loss, jitter and delay which were used for networked services are not easily understood by the users. Instead, Quality-of-Experience (QoE) metrics which capture the overall satisfaction are more suitable for measuring the quality as perceived by the users. However, these QoE metrics have not yet been standardized and their measurement and improvement poses unique challenges. In this work we first present a comprehensive survey of the different set of QoE metrics and the measurement methodologies suitable for HTTP based online video streaming services.

We then present our active QoE measurement tool *Pytomo* that measures the QoE of YouTube videos. A case study on the measurement of QoE of YouTube videos when accessed by residential users from three different Internet Service Providers (ISP) in a metropolitan area is discussed. This is the first work that has collected QoE data from actual residential users using active measurements for YouTube videos. Based on these measurements we were able to study and compare the QoE of YouTube videos across multiple ISPs. We also were able to correlate the QoE observed with the server clusters used for the different users. Based on this correlation we were able to identify the server clusters that were experiencing diminished QoE.

Dynamic Adaptive Streaming over HTTP (DASH) is an HTTP based video streaming that enables the video players to adapt the video quality based on the network conditions. We next present a rate adaptation algorithm that improves the QoE of DASH video streaming services that selects the most optimum video quality. With DASH the video server hosts multiple representation of the same video and each representation is divided into small segments of constant playback duration. The DASH player downloads the appropriate representation based on the network conditions, thus, adapting the video quality to match the conditions. Currently deployed Adaptive Bitrate (ABR) algorithms use throughput and buffer occupancy to predict segment fetch times. These algorithms assume that the segments are of equal size. However, due to the encoding schemes employed this assumption does not hold. In order to overcome these limitations, we propose a novel Segment Aware Rate Adaptation algorithm (SARA) that leverages the knowledge of the segment size variations to improve the prediction of segment fetch times. Using an emulated player in a geographically distributed virtual network setup, we compare the performance of SARA with existing ABR algorithms. We demonstrate that SARA helps to improve the QoE of the DASH video streaming with improved convergence time, better bitrate switching performance and better video quality. We also show that unlike the existing adaptation schemes, SARA provides a consistent QoE irrespective of the segment size distributions.

APPROVAL PAGE

The faculty listed below, appointed by the Dean of the School of Graduate Studies, have examined a dissertation titled “Measurement And Improvement of Quality-of-Experience For Online Video Streaming Services,” presented by Parikshit Juluri, candidate for the Doctor of Philosophy degree, and hereby certify that in their opinion it is worthy of acceptance.

CONTENTS

ILLUSTRATIONS

TABLES

ACRONYMS

**ABR** Adaptive Bitrate

**AS** Autonomous System

**BBA** Buffer Based Adaptation

**CDN** Content Distribution Network

**DASH** Dynamic Adaptive Streaming over HTTP

**DNS** Domain Name System

**FLV** Flash Video

**GENI** Global Environment for Networking Innovation

**HTML** Hyper Text Markup Language

**HTTP** Hyper Text Transfer Protocol

**IATA** International Air Transportation Association

**ISP** Internet Service Provider

**ITU** International Telecommunication Union

**MOS** Mean Opinion Score

**MPD** Media Presentation Description

**NAT** Network Address Translation

**QoE** Quality of Experience

**QoS** Quality of Service

**RTT** Round Trip Time

**SARA** Segment Aware Rate Adaptation

**SeRViTR** Secure and Resilient Virtual Trust Routing

| | |
|---|---|
| **TBA** | Throughput Based Adaptation |
| **TCP** | Transmission Control Protocol |
| **URL** | Uniform Resource Locator |
| **VoD** | Video on Demand |
| **WHM** | Weighted Harmonic Mean |

ACKNOWLEDGEMENTS

Pursuing a PhD is an adventurous journey, which would not have been as fruitful or fulfilling without the guidance and support of several people.

First and foremost, I would like to thank my advisor Dr. Deep Medhi for all his guidance and enthusiastic support throughout my PhD. Dr. Medhi has taught me to appreciate the good in everything while at the same time being inquisitive and challenging the status quo. Dr. Medhi always emphasized on gaining a well rounded experience as a PhD student. This has enabled me explore opportunities not only in academic research and teaching but also in the industry and work with Orange Labs for part of my work.

I sincerely thank all of the other members of my committee, Dr. GhulamChaudhry, Dr. Kenneth Mitchell, Dr. Baek-Young Choi, Dr. Masud Chowdhury and Dr. T. Venkatesh for their support and feedback.

I have been very fortunate to work with Dr. Louis Plissonneau at Ornage Labs as an intern. Louis taught me the importance of giving attention to detail while at the same time having a clear picture of the final goal. Louis also introduced me to Python programming and this theses would not have been possible without the skills I learnt from him.

I would to thank Dr. T. Vekatesh for his extensive feedback and comments which have significantly helped me in my research.

I would also like to thank Dr. Yong Zeng for his help in collecting the data for my YouTube case study and his guidance during the analysis.

I would like to thank all my lab mates, Xuan Liu, Haiyang Qian, Shuai Zhao, Shahriar Maswood, Rohit Abhishek, and Sheyda Kiani Mehr for adding cheers to every day of this journey, all my friends for their unconditional support, the office staff and colleagues at UMKC for their help.

Most importantly, I would like to thank my parents Dr. J. Manohar Rao and Dr. J. Namratha Manohar and my younger brother Dr. J. Dushyanth for their patience, encouragement, and support over the years.

# CHAPTER 1

# INTRODUCTION

Online video streaming services have become the most popular form of entertainment in the recent years. With Video On Demand (VoD) services, the users are able to watch varied contents: user-generated videos, movies, TV shows, sports etc., from multiple different platforms-laptop, PC, televisions, game consoles, mobile phones and tablets. Over the last few years, the video streaming services have accounted for most of the prime time Internet download traffic. In North America alone, the top video streaming services, Netflix and YouTube, accounted for 43% of the total peak download traffic in 2014. If other VoD services such as Hulu and Amazon video services are considered, this percentage rises to as high as 63%. A similar trend could be found in the global Internet traffic. Globally, video streaming traffic is expected to account for 80% of the consumer traffic on the Internet by 2019, up from 64% in 2014 [24].

The increasing dominance of video traffic over other Internet traffic could soon make it a prime online service for the average user. More and more users are relying on web-based services for their video entertainment needs. The trend of increasing cord-cutters (users that cancel cable or satellite TV subscription) could also be correlated to these trends. The number of devices that are connected to the Internet has also been increasing. Most consumer devices are now connected to the Internet and have the capabilities for media playback. These devices are not limited to the mobile phones, televisions, game consoles, tablets, or watches. With the advent of HTTP based video streaming services, any device with a web browser can be used for video streaming.

In order to ensure the satisfaction of the existing users there has been an increasing interest from the video service providers, Internet Service Providers (ISPs) and even wireless network providers to develop tools and techniques that could measure and improve the user's satisfaction. Measurement of existing quality of services is an important factor in understanding the current system's capabilities and also this assists the engineers in the network and system forecasting.

Traditionally, the Quality of Service (QoS) metrics have been used to study the performance of online services and networked elements. Quality of Service(QoS) as defined by ITU [45] reflects the performance of the network and its components. It measures the network's ability to satisfy the needs of the service and is thus, a network-centric metric. The common QoS metrics used are throughput, bandwidth, packet loss, delay, or jitter. QoS metrics have been found to be more suitable to measure the performance and reliability of the network elements. However, QoS metrics do not capture the quality of service perceived by the user. The service perceived by the user is subjective and is influenced by not only the network components but also several other confounding end-to-end factors. These confounding factors include the effects of the service infrastructure, terminal, client hardware and operating system, system load, and the user's psychological and environmental settings. In order to understand the user's satisfaction with a service it, is necessary to consider the effects of these end-to-end confounding factors on the end user's perception of quality.

## 1.1 Quality of Experience (QoE)

Quality-of-Experience (QoE) as defined by [44, 85] is a user-centric metric that captures the overall acceptability of the service and includes the end-to-end factors. QoE has also been defined as the degree of delight or annoyance experienced by a user of an application or service [22]. QoE, thus, measures the performance as subjectively perceived by the user. However, QoE and QoS are not mutually exclusive; rather, QoE could be considered as an extension to QoS. QoE tries to capture the performance of a service with metrics that could be directly communicated by the user.

Typically, VoD streaming sessions last from several minutes to hours. If there is a noticeable delay after a video link is clicked, or if there is an interruption during the playback or any perceivable drop in the visual quality of the video, these can affect the perceived quality of the user. The users of video streaming services usually have a different set of expectations as compared to other types of web services.

The factors that affect the QoE for video steaming services are not easy to predict or measure. There has been significant work in identifying these factors and developing tools to measure the metrics that could used to estimate the QoE. We present a comprehensive survey of the various factors and the methods that are being used to measure the QoE in Chapter 2.

## 1.2 HTTP Based Video Streaming Techniques

Compared to an average HTML web-page, the size of a typical video file is significantly larger. The average web page size is found to be 1.9 MB in October, 2014 [5]. In

2007, based on their measurements, the authors in [37] found that the average YouTube video size was 10 MB. Since then, YouTube has increased the maximum video file size to 2 GB from 100 MB and also started hosting HD videos. The HD videos hosted by YouTube have an average size of 32 MB per minute [60]. Using these numbers as reference, a user would need to wait for significantly longer durations if the users' video players were to download the entire file before starting the playback. This clearly would affect the users' QoE. Instead, online video streaming services are set up in such a way that users can start viewing the content once the initial part of the video is downloaded, without having to wait for the video to be completely downloaded.

The use of Hyper Text Transfer Protocol (HTTP) for video streaming has become very popular over the last few years. Unlike the traditional streaming protocols, viz., Real-Time Streaming Protocol, Real-Time Media Streaming Protocol etc. [54], HTTP is a stateless protocol. For HTTP based video services it is not required to have dedicated video servers. With HTTP the content providers are able to reuse the existing web-servers, caches and content-delivery architectures. The stateless approach of delivery also reduces the load on the servers by avoiding persistent feedback loops with the client and enables the system to scale better. HTTP natively and easily supports mirroring and edge caching, thus enabling large-scale expansion if necessary. HTTP is also widely enabled across Network Address Translation (NAT) devices and firewalls. The ubiquitous nature of devices compatible with HTTP makes it easier to port the video player onto multiple client platforms. HTTP based video streaming techniques are of two types: progressive download and Dynamic Adaptive Streaming over HTTP (DASH).

### 1.2.1 Progressive Download

The progressive download streaming technique refers to the continued download of a video file, while the video player plays out the video content received so far. In *Progressive Download Streaming* techniques, the multimedia file is downloaded like a regular file using HTTP over TCP from a web server hosting the content. With progressive download, there is no direct feedback loop between the client and the server. On the client-side, the entire video file is being downloaded as quickly as possible and stored on the local disk. However, in order to reduce the amount of data delivered to the user and to reduce the network resources utilized, the server-side could implement certain flow control mechanisms. The video player (typically an embedded Flash player) starts playing the video (for the data received so far) while it is still being downloaded.

Using HTTP over TCP for video transfers ensures reliable data delivery, but in the case of low bandwidth availability or excessive packet losses during the transmission, the playback may be interrupted. By using HTTP, the complexity on the server-side is reduced and the clients are not affected by a firewall or a proxy. However, downloading the media as fast as fast as possible, could lead to waste in bandwidth if the user decides to quit before completely viewing the entire video.

With progressive download, an embedded *Flash Player* is used to play the video directly from any suitable web browser [92]. The player downloads the video as a regular file and stores it in a temporary folder. The downloaded video file is usually an interleaved stream of audio and video blocks with tags to indicate the time at which they are supposed to be played. These tags help the Flash Player to skip backward or forward within the video file already downloaded. Since the entire video is stored locally, the backward skip is simply a local operation. In case of a forward skip to a position in the video that has not yet been downloaded, the Flash Player sends a new HTTP request with the byte-range indicating the new position to the server. Although the progressive download technique has been traditionally used for VoD, it could also be used for live content as well.

Since the server does not maintain any state, the player handles the video playback in two stages to ensure smooth playback. During the initial stage, the player waits for a certain period of time known as the *Initial Buffering* period, during which the content that can play the first 30 to 40 seconds of playback is downloaded as fast as possible from the server [36]. Once the *initial buffer* is filled, the video playback starts. By delaying the start of the playback, the player avoids the effect of packet delay and jitter on the playback. The content is continuously downloaded and buffered while it is played by the Flash player. If the difference between the amount of content played and the content downloaded falls below a certain threshold, the video playback is paused resulting in an interruption. The video playback remains interrupted until the threshold is crossed again after which the playback resumes. The threshold values for different stages of buffering is not standardized and could vary across various implementations of the progressive download video player.

The servers tend to employ flow control mechanisms, to improve the initial buffering time and also reduce the amount of unnecessary data transferred in case the user decides to quit before watching the entire video [15]. Progressive download servers tend to send the video as fast as possible during the initial stages before settling down to a constant sending rate. The sending rate during this steady phase is maintained at or slightly above the playback speed to ensure smooth playback.

Since the video could be accessed by users from different hardware platforms, it is necessary for the servers to be able to support various formats. Each of these different formats are stored as separate files with a different URLs [34]. Before initializing the video streaming, the default format and the resolution are determined by the player and based on the hardware platform and screen resolution settings (such as full-screen and wide screen). The selected format is typically fixed for the entire duration of the playback. However, the player supports the ability to automatically change the format/resolution of the playback depending on the network conditions. This shift is initiated by the player at the user-end and results in the new video format file being downloaded. The popular progressive download video streaming services include YouTube, Dailymotion, Vimeo etc. At the time of writing this paper, YouTube has started supporting most of its videos using MPEG-DASH (discussed in Section 1.2.2) in addition to progressive download [63].

### 1.2.2 Dynamic Adaptive Streaming Over HTTP (DASH)

Progressive download has the advantages of HTTP-based streaming, however, it does not have the capability to adapt the video streaming rate during playback. Any perturbations in the network during playback leads to buffer starvation which results in playback interruptions. The Dynamic Adaptive Streaming Over HTTP (DASH) technique was proposed to support adaptive streaming over HTTP [91]. A DASH server does not maintain any state information during the session. The rate adaptation is handled at the client-end. By offloading the decision making process onto the client-end, the system is able to scale well while still providing dynamic adaptive streaming. It was observed in [102] that the above advantages of DASH resulted in a better QoE for the users when compared to the progressive download techniques.

DASH considers a media file to be a collection of several components: audio, video, and subtitles that are stored separately on the DASH server. These components are delivered to the user independently but combined at the time of the playback [91]. Each of these components is further divided into smaller chunks called *segments* and each *segment* could be encoded into multiple versions. Every *segment* is identified by a unique URL. The different versions of the same media file are called *representations*. Different *representations* vary in bit rate, resolution, format, language, and other characteristics.

When the user starts playing a video, the client first retrieves a Media Presentation Description (MPD) file that is a manifest with the list of all the available *representations*, suggested bandwidth for each *representation*, video dimensions, the digital rights management (DRM) information, the location of each *segment* on the network (URL) and other attributes of the media file. A sample MPD file can be found in Appendix A. In order to play the video, not all representations need to be supported by the client device.

At the beginning of a session after downloading the MPD file, the video player sends an HTTP request for the first *segment* of the video using its URL. Typically, the *segment* with the lowest resolution is requested first. While the first *segment* is being downloaded, the player monitors factors such as throughput, delay, and client-buffer status. [18] reviews several tools that could be used by the DASH streaming services to estimate the bandwidth. Based on these factors, end-device capabilities, and the user's preferences, the DASH player employs adaptation algorithms to determine the most suitable representation for the next segment [67, 73]. The player, thus, decides to either stay with the same representation or shift to a higher or lower representation. The representation is selected to play the best possible quality of the video with the least number of interruptions. Once the decision is made, a new HTTP request is sent for the next *segment*.

Thus, the DASH player is able to adapt the quality of the video at the boundary of each segment in order to ensure uninterrupted playback. Since, DASH retains the advantages of HTTP services while providing adaptive streaming it has gained tremendous popularity over the last few years. Currently the top video streaming services such as Netflix, YouTube, and Hulu use DASH streaming [10].

## 1.3 Scope and Contribution of this Dissertation

In this dissertation, we focus on two aspects of QoE management for online video streaming services. First, is the measurement of QoE for online videos and the second is the improvement of QoE for online video streaming services. The contributions of this dissertation are as follows:

- Although the importance of QoE as a measure has been generally agreed upon, the factors affecting QoE and the metrics used to capture the effects of these factors has not been standardized yet. There have been several tools and methodologies used to measure the QoE for online video services. As an effort to unify these approaches we have conducted an extensive survey of the existing literature on measurement of QoE. This work has been published in [54] and a part of this survey is presented as related work in Chapter 2

- We contributed to the development of a user-end active measurement tool, *Pytomo* to measure the QoE of the popular video streaming service, YouTube. A paper describing this tool was published in [51] and is briefly described in Chapter 3.

- We used *Pytomo* to study the QoE and the content delivery policies for users across different ISPs in a metropolitan area. We present this data as a case study in Chapter 4. This work was published in [52].

- We propose the Segment Aware Rate Adaptation (SARA) algorithm for DASH streaming services to improve the QoE in terms of bitrate switching events, video quality and convergence times. This algorithm and the preliminary evaluations have been published in [53, 56]. The algorithm is presented in Chapter 6.

- We also present a study of the comparison of performance of SARA with existing throughput based and buffer based rate adaptation algorithms is published in [55] and is is presented in Chapter 7.

## 1.4 Additional Contributions

In addition to our work on QoE as presented in this dissertation we have collaborated on a number of research projects that resulted in the following publications:

- A comparative study on the server delivery policies and the perceived QoE of YouTube video in US and Europe was published in [79].

- Apart from the measurement and improvement of QoE at the user-end, we have also proposed an in-network efficient caching framework aimed at increasing the cache-hits and reducing the cache-misses, which was published in [50].

- An experimental study on dynamic network reconfiguration in a virtualized network environment using autonomic management was performed in collaboration with Xuan Liu and was published in [70].

- In [19, 71], the Secure and Resilient Virtual Trust Routing (SeRViTR) framework design, implementation, and and a testbed that enables us to demonstrate SeRViTR was presented.

## 1.5 Organization

The rest of this dissertation is organized as follows: Chapter 2 presents an extensive literature survey of different categories of QoE metrics and the various types of measurement techniques proposed in literature. In Chapter 3, we present our client-end active measurement tool designed to measure the QoE of YouTube videos. A measurement study that uses *Pytomo* to study the QoE and server policies across different ISPs in the same metropolitan area is presented in Chapter 4. Later, we propose a novel Segment Aware Rate Adaptation (SARA) algorithm for DASH video streaming in Chapter 6 followed by an comparative evaluation of SARA in Chapter 7. Chapter 8 concludes this dissertation.

# CHAPTER 2

# MEASUREMENT OF QOE FOR ONLINE VIDEO STREAMING SERVICES: A LITERATURE SURVEY

In traditional video broadcast services QoE metrics were measured by comparing the reference with the outcome at the user-end. Reference reflects the undistorted content in its original form at the server end. The outcome received at the client-end could be potentially distorted or delayed. Based on the amount of information available about the reference the QoE metrics for video services have been classified into the following categories [33]:

- **Full reference(FR) metrics**: Complete copies of the reference and outcome are available to evaluate the quality of the video received at the user in comparison with the original content. This enables detailed subjective and objective comparisons of the videos. These metrics are best suited for traditional broadcasting and television systems which have dedicated delivery networks. A few examples of FR metrics are Peak Signal to Noise Ratio (PSNR) [86], Structural Similarity (SSIM) [99], and Video Quality Metric (VQM) [86].

- **No reference(NR) metrics**: Only the outcome is available and the quality is to be estimated without the reference stream. These types of metrics are more applicable to online services, where the delivery network is shared by other services. In video streaming services, it is hard to determine if the discrepancy in the quality is due to

Table 1: Classification of QoE Metrics for Online Videos

| **Objective Metrics** | **Subjective Metrics** |
|---|---|
| Playback Start Time | Mean Opinion Score (MOS) |
| Number of Interruptions | |
| Duration of Interruptions | |
| Quality of Video File | |
| Bitrate Switching Events | |
| User Engagement | |

the quality of the reference or due to the intermediate elements.

- **Reduced reference(RR) metrics**: The same set of parameters are derived from both the reference and the outcome. These parameters could be at the application layer: bit-rate, frame-rate or at the network layer: packet-loss.

The Full reference and Reduced Reference metrics are not suitable for online media streaming services. This is due to the distance between the client and the server and also due to lack of separate feedback channels. Hence, No Reference metrics are most suitable online streaming services.

## 2.1 Types of QoE Metrics for Online Video Streaming Services

Depending on the type of measurement mechanisms employed, the QoE metrics could also be classified into objective and subjective metrics [85](Table 1).

### 2.1.1 Objective QoE Metrics

Objective QoE metrics are metrics that can be quantified with an automated measurement tool. The following are the common objective QoE metrics that capture the factors that influence the user's QoE.

1. **Playback Start Time** The *playback start time* or the *initial delay* is the time taken from the moment a user clicks the video link until the video playback begins. The *playback start time* typically includes the time taken to download the HTML page and the related objects, download and load the embedded video player plugin, and buffer the initial part of the video. In the case of streaming videos, the player starts playing the video only after a part of the file is downloaded (called *Initial Buffering*). By doing so, the player overcomes the effect of the delay and jitter incurred during the data transfer on the video playback. The *playback start time* is an important factor that affects the QoE, and hence, certain VoD services (such as YouTube) usually tend to push data at higher rates initially and settle down for a lower rate later [83]. It was observed in [59] that playback start time had a significant influence on user retainment. If the playback start time extends by more than 2 seconds it could result in the viewer abandoning the video completely [59]. Playback start time can be used to quantify the QoE in any type of video streaming.

2. **Number of Interruptions** An interruption occurs when the playback of the video is temporarily stalled. A streaming player downloads the initial parts of the multimedia content into a playout buffer before the video has started playing. As long as the rate at which the buffer is being filled is greater than or equal to the rate at which the video is played, the playback is not interrupted. If the download rate falls below the playback rate, the buffer gets depleted and the player waits for the buffer

to be partially filled before resuming the playback; this wait time results in an interruption of the playback. The interruption is thus a direct consequence of video player's *buffer starvation*. The interruptions are also referred to as *(re)buffering events* and the frequency with which the buffering events occur is called the *buffering frequency*. The re-buffering could also be a consequence of user interaction. When a user skips to a different part of the video or changes the quality of the video during the playback, the player needs to fetch the requested content from the server and then continue the playback from the desired position. In this case, the player again waits for a certain buffering period before resuming the playback.

In case of DASH, if the player observes any drop in the receiving rate, it may automatically switch to a lower bitrate so that the video playback is not interrupted. However, if the network condition becomes significantly bad that even the lowest bitrate content cannot be downloaded in time, then the user would see interruptions. The interruption lasts until the playout buffer is partially filled with the content of the desired bitrate. These interruptions or stall events during the playback lead to a poor user experience [39]. It is found that the number of interruptions have a significant impact on the QoE [33]. Users who experienced more interruptions in the video tend to watch the video for shorter durations [59] and are likely to be dissatisfied in the case of four or more interruptions for videos [40].

3. **Duration of Interruptions** Apart from the number of times the playback is interrupted, the duration of each interruption ("buffering duration") could also effect the perceived QoE. If the interruption duration is one second, the users are less

dissatisfied when compared to 3 seconds of interruption while watching YouTube videos [39, 40]. In [81], it was concluded that the viewers prefer a single but long stall event instead of several short stall events. Hence, the effect of the duration of the interruptions and the number of interruptions on the QoE could vary.

4. **Quality of the Video File** The quality of a video stream is based on the encoding rate. The encoding rate is the average data required to play one second of the video. The encoding rate of the video does affect the QoE of the users [64]. A higher quality video (HD) would usually require a larger amount of data for each frame and hence, results in a higher encoding rate. Progressive download streams typically stick to the same encoding rate throughout the duration of the playback irrespective of the change in the network quality. Adaptive streaming techniques, like Real-Time video streaming and DASH, vary the encoding rate depending on the network parameters.

   There are other video characteristics that have been used to represent the quality of the video such as the contrast, blurring [77], and blockiness [32, 65, 100, 103]. Blockiness manifests as a block appearing in the video. It is caused by the block-based coding schemes such as H.261, H.263, MPEG-1. These other video characteristics have been mainly used in the traditional video broadcasting.

5. **Bitrate Switching Events** The Bitrate switching events are related to the Dynamic Adaptive Streaming over HTTP technique (DASH). For DASH videos, the player tends to pick a lower initial startup and gradually keeps increasing the quality before

settling at a suitable bitrate. The bitrate could later be reduced when the rate of the playback exceeds the buffering rate due to degraded network conditions. By lowering the quality of the video streaming, the player minimizes the interruptions during the playback. On the other hand, when the network conditions improve, the bitrate is increased. Thus bitrate switching events could be of positive or negative polarity. A bitrate switching event where the level of bitrate is increased is called *Positive switch*, whereas a decrease in the existing bitrate is called *Negative switch*. However, frequent switching in bitrates can degrade the users QoE [102]. Hence, it is necessary to ensure that the number of bitrate switching events are reduced. The startup bitrate, number of bitrate switching events, and the average bitrate affect the QoE [64, 102]. Different polarities of bitrate switching events can have varying effects on the QoE. Users were found to be more critical towards *negative switching* events when compared to *Positive switch* events [26].

6. **Convergence Time** is another objective metric related to DASH video streaming services. As discussed in Chapter 1.2.2, the DASH player starts the playback with lowest video quality and then gradually changes to the optimum quality (preferably highest quality). The bitrate switching occurs at the boundary of each segment. Depending on the rate adaptation scheme used and the network conditions, the time taken for the player from starting playback to reaching the highest quality could vary and this duration is considered as the *Convergence Time*. The QoE of the users can be improved by minimizing the *Convergence Time* for DASH videos.

7. **User Engagement** User Engagement reflects the user involvement and interaction

with the video. User engagement is measured in terms of the number of views and the play time of the video. However, the play time might not reflect the amount of time the user actually spends watching the video without getting distracted. It is hard to quantify the user's focus which is a subjective metric. The users who are satisfied with the content and the QoE of the streaming session tend to spend more time watching the video [31].

### 2.1.2 Subjective QoE Metrics

Subjective metrics are the QoE metrics based on collecting data directly from the users based on their experience with the service. A limited set of human subjects are exposed to the video service in a controlled environment and are asked to rate them on a linear scale. Due to the use of actual human subjects, the ratings could be affected by several physical and psychological confounding factors and hence, subject to user-bias. The confounding factors can be broadly classified as (1) user-dependent: user interest, purpose (educational, entertainment) (2) content-based: genre, age of the content and popularity; or (3) device-dependent: quality of the Internet connection, screen size, and the capabilities of the device. Such subjective metrics are susceptible to bias and hence, can vary from one subject to another. The best way to measure the effects of these factors is to collect direct feedback from multiple users using robust sampling methodologies and to use statistical analysis techniques to avoid any bias.

1. **Mean Opinion Score(MOS)** The *Mean Opinion Score* (MOS) is the most popular subjective metric measurement scale that is often used to quantify these factors.

Users watch videos and rate them on a five-point discrete scale: 1-bad, 2-poor, 3-fair, 4-good, and 5-excellent. The use of the MOS as a subjective metric has become the de facto standard for subjective assessment. It is, however, not easy to automate the MOS measurement since the influence of the human psychological factors and the user bias needs to be considered. In order to predict the MOS, a good understanding about the psychology of the users to predict the MOS ratings is necessary.

## 2.2 Measurement of QoE Using Client-Side Instrumentation

To precisely capture the QoE as experienced by the user many studies have proposed using tools closer to the client-end to measure the objective metrics. These metrics are collected by using measurement tools that run on the user devices. Depending on how the data is collected by the tool, the user-end measurement studies can be further classified into two categories: measurement based on active and passive analysis

### 2.2.1 Measurements Based on Passive Analysis:

Passive analysis tools collect QoE metrics for the videos that are being watched by the users. These tools run in the background and the QoE metrics are obtained in real-time by analyzing the video streams being played. In this case, the videos for which the QoE is measured is purely dependent on the interest of the user and the tool has no control over the selection or the playback duration of the videos.

In [92], the authors developed a client side passive measurement tool, YOMO that collects the QoE metrics for progressive download videos, specifically YouTube videos.

YOMO estimated the number of interruptions occurring while watching a YouTube video by tracking the status of the playout buffer that reflects the total time, say $\beta$, that the playback can continue in case of an interruption in the download. Here, $\beta$ is the difference between the time in which the content could be played from the playout buffer, $T$ and the current play time of the video, *t*. In case $\beta$ is smaller than a threshold $\beta_0$, the playback is interrupted. In case of YouTube, they found that the API waited for a certain duration from the time the first byte of the video is downloaded before the playback is started. However, this duration was observed to be inconstant and it did not have any correlation with the video characteristics. The accuracy with which the duration of the interruption is estimated depended on the accuracy in estimating *t*, which is also the time since the playback started. The authors proposed two methods to estimate *t*. In *Method 1*, it was assumed that the video playback began as soon as the first Flash Video (FLV) tag was downloaded. The accuracy of this method was found to be directly related to the bandwidth. In this method, the error was sufficiently low for broadband connections. In *Method 2*; YOMO uses a Firefox plugin that retrieved *t* from the YouTube player. This method resulted in an estimation independent of the bandwidth. The maximal error in this method was 0.5 seconds whereas it was 20 seconds in the case of slow connections with *Method 1*.

In [30], the application layer metrics are used to estimate the QoE of *Windows Media Player* users. They developed a wrapper for the player to collect the application layer metrics such as the number of packets lost, recovered, and received, the current data rate of the stream, and buffer starvation, if any. The data collected from the player was sent to a centralized location to analyze and report on the statistics. The current work used a distributed architecture that consisted of assessment servers, media clients, data collection points, and report servers. The data collection was scheduled and collected by the central assessment servers. The playback metrics collected from the application were transferred to the data collection points and sent to the assessment servers, where the data was analyzed. The analyzed results were sent to the report servers where the analyzed data was made available to the customers. They demonstrated that the application layer metric, player buffer starvation, could be used as an indicator to predict the playback interruptions. A similar approach is taken in [28] where the metrics such as the number of packets lost and retransmitted at the application layer are measured and compared against similar metrics for a reference stream to predict the user-perceived quality. The authors in [28] concluded that the re-buffering frequency and initial buffering time are the main factors affecting the QoE. In [29], the authors extended their earlier study by also collecting the *initial buffering period,* which is equal to the *Playback Start Time* along with other application layer metrics to predict the MOS ratings for the videos.

In [74, 75], the authors used client-side instrumentation for HTTP based progressive download video streaming. The instrument kept track of the user initiated actions such as pausing, resuming playback, jumping within a video, and screen switching, etc. The QoE metrics such as initial buffering time, mean buffering duration, re-buffering frequency, and bitrate switching were measured for the video streaming sessions. Apart from these objective metrics, the users were also asked to rate the videos using the MOS scale. The objective QoE metrics and the subjective feedback from the users thus collected were used to correlate the objective metrics with the MOS. Similar to the previous studies, they concluded that the re-buffering frequency is the main factor affecting the QoE in terms of the MOS.

In [69], client-side instrumentation was used to collect data from 50 million viewers for 200 million views of both VoD and live streaming services that were served by 91 different content providers. They used the QoE metrics such as *re-buffering frequency, playback start time, average bitrate, and failure to start video*. Using these metrics, three issues that could result in poor video quality were identified: (1) client-side bandwidth variations during the playback, (2) variation in the CDN performance across the time and geographic regions, and (3) heavily loaded ISPs. In order to overcome these issues, they proposed a video control plane that utilized measurement-driven feedback on the performance. The feedback enabled the control plane to dynamically adapt the video parameters such as the CDN that was used and the bitrate to improve the quality.

In [31], client-side instrumentation was used to measure the *initial playback time, buffering ratio (defined as the ratio between buffering time and play time), buffering frequency, and average bitrate* for different types of videos: long, short, and live. These videos were accessed from multiple different content providers, that used varied streaming techniques. A quantitative analysis was done on the correlation between the three different factors: QoE metrics, content type, and user engagement. Based on the measurements, it was concluded that the buffering ratio impacted the user-engagement for all content types whereas the bitrate mainly affected the user-engagement for live streaming videos. They also determined that the *initial buffering period* was critical for the user engagement.

In the client-side passive measurement tools presented above, the authors developed wrappers around the players for different streaming services to capture different metrics. They either polled the video player or observed the buffer status of the player to estimate the status of the video playback. The data collection in such tools requires the active involvement of the users.

### 2.2.2 Measurements Based on Active Analysis:

In measurements based on active analysis, the requests for the videos are generated artificially and the QoE metrics for these videos are collected. Active measurements circumvent the need for a user to sit and watch the entire set of videos that need to be evaluated. Active measurement approaches typically use crawlers or bots that crawl through the video streaming websites and collect the QoE metrics for a large number of videos. The advantage of using such tools is that they can be easily used to measure the QoE for a large number of videos and do not require any user participation, thus eliminating any subjective bias.

In [83], the latency of three video services, YouTube, Dailymotion, and Metacafe, was measured and compared using active measurements. Automated scripts were run on several PlanetLab nodes to fetch 1 MB of each video in 50 KB increments. They computed the mean service delay for a video by averaging the service delays of three consecutive 1 MB chunks to eliminate the effect of the size of the files. This service delay, referred to as the *incremental service delay*, was independent of the client-side application as it did not account for the flow control or the buffering scheme used. By comparing the incremental service delay for the three services, it was observed that YouTube delivered 1 MB of video content nearly 6 times slower than Dailymotion and Metacafe. However, the incremental service delay is not a direct quantifier of the QoE.

Similar to the studies based on passive analysis, the active measurements are also done as close to the user as possible. These tools also capture the effect of the performance of the user device and the last hop connectivity on the QoE. Since active analysis uses crawlers, it requires minimal involvement of the users.

## 2.3 Measuring QoE Using Direct User Feedback

The QoE of users is a subjective metric and one popular method of evaluating it is to obtain the rating directly form the users. Typically, a set of users (friends, colleagues, or volunteers) are requested to watch a set if videos in a controlled environment and provide their rating on an MOS scale. These studies are found to be good at capturing the effect of confounding factors on the QoE. In such subjective evaluations it is necessary to maintain a consistent viewing conditions for the different users. The standard in [46] specifies the conditions that are needed to be considered to conduct experiments that describe the viewing distance, room lighting conditions, selection of the subjects, video content selection, and assessment and data analysis methods.

In [74], the authors created a platform that played HTTP streaming videos by varying the objective QoE metrics and collected the MOS rating from 10 different subjects who were non-experts in the estimation of video quality. The authors determined that the interruptions in playback was the main factors affecting the MOS rating.

Collecting a sufficient number of unbiased users to provide feedback on the videos is a non-trivial task. This becomes even more difficult if the study targets different geographic areas. Some studies have used a crowd-sourcing model to overcome these limitations. In a crowd-sourcing platform, users are provided with incentives to perform small tasks. Example of such platforms are Mechanical Turk [16] and Microworkers [72]. However, such studies need to ensure the reliability of the results as the users are not monitored or provided with any controlled environments. In order to gather the data that is more reliable, the tasks assigned need to be interleaved with gold standard data tests, consistency tests, usage monitoring, and content questions as suggested in [40]

In [39], a subjective study was performed using both volunteers in a laboratory and crowd-sourcing. This study compared the sensitivity to initial delay and the interruptions while watching YouTube videos. Apart from YouTube videos, they also studied the effect of the authentication time for the social networks and Wireless 3G Internet connection setup time on the QoE. Based on the MOS data collected, the authors found that the users in both the experiments preferred some delay before playback started instead of an interruption. It was also observed that the results from crowd-sourcing were similar to the results from the laboratory tests and the length of the video clip did not have any influence on the MOS.

In [102] the authors developed a framework to obtain the subjective ratings from the volunteers to measure the QoE for an adaptive video streaming service and compare it with fixed bitrate video streaming (progressive download) technique. They used 141 volunteers to rate artificially spliced variable bitrate samples that emulated a DASH session. The volunteers were asked to rate the overall video viewing experience using a MOS rating and also rate their satisfaction with objective metrics such as video-definition (video quality), fluency (interruptions), response speed (initial delay). Based on these studies, they found that for the DASH streams the number of interruptions affects the QoE more than the video quality, the startup bitrate and slow bit-rate ramp-up also affects the QoE, and the users are sensitive to frequent bit-rate switching. They also concluded that in general the QoE of DASH videos was better than the progressive download videos.

The MOS feedback collected from the users either in a controlled lab or crowdsourcing environment can be used to understand the effects of various metrics on the QoE. However, it is difficult and time consuming for the users to watch and provide feedback on every video. This process of collecting data is also expensive and not easily repeatable.

## 2.4 Other Measurement Techniques

The client-end QoE measurement techniques are good at capturing the QoE at a point closest to the user and can capture the end-to-end effects on the video performance. However, it is necessary to gain access to the user machine in order to perform the measurement. For the network operators and service providers this approach might not be the most suitable. They however have the advantage of unique vantage points that have access to the complete data passing through the network. There are different set of studies that collect data from places other than the user-end to estimate the QoE.

### 2.4.1 Estimating QoE from measurements within the network

While the measurements at the user-end estimate the user satisfaction, they cannot be implemented easily by the network providers. A network provider can collect metrics within the network in a faster and easier manner. There has been extensive research on measuring the QoS metrics such as throughput, loss rate, delay, jitter, and packet re-ordering and it was demonstrated that these network-level parameters affect the perceived quality [57], [25], [93]. If the network-level metrics could be used to predict the QoE of the user, it would help the network providers to extend the existing measurement tools to evaluate the QoE.

The authors of [82] used Deep Packet Inspection (DPI) in the network to develop a preliminary model to estimate the QoE in the case of progressive download services. From the TCP headers of the packet traces collected, the timestamps of the TCP acknowledgments at the client were monitored. By tracking the TCP segments and the corresponding acknowledgments (ACKs), they were able to estimate the playout buffer level at the client and predict the QoE metrics such as initial buffering time, number of interruptions, and total duration of the interruptions. However, this requires the complete packet trace and is an offline analysis approach.

In [34], the authors used Tstat [95], a DPI tool that implements a traffic classifier from the flow-level statistics. YouTube traffic was analyzed from vantage points within the ISPs and university campuses across Europe and the U.S.A. It was observed that all YouTube video requests contained `HTTP videoplayback` tags in their HTTP headers. By observing the time difference between the `HTTP videoplayback` request and the reception of the video data in the payload, the start up latency of the video was predicted (the initial buffering period at the user was ignored). The user behavior during the playback was observed from the user-initiated bitrate switching, screen mode (full-screen) switching, and the portion of the video actually watched to correlate with the system performance. It was observed that although progressive download maintains the quality of streaming with aggressive download, the amount of unused data was significantly high as the users typically watched only a part of the video. The behavior of the users across different devices (such as mobiles and PCs) was also studied and it was observed that the device, location, and infrastructure had no effect on the playback quality of the YouTube videos.

An on-line QoE estimation algorithm for progressive download videos was presented in [62]. The packet-level metrics were obtained from the TCP headers and the meta data information of the video. The network layer information was collected using the Access Network TCP Monitoring Algorithm (ANTMA) [98] to predict the number of interruptions and the duration of the interruptions in real-time. However, the algorithm is designed only for an MP4 video format and makes two assumptions to work in real-time. First, the node should see all the TCP packets in both directions. Second, no re-ordering of packets takes place between the monitoring node and the TCP receiver. Based on their analysis, the authors were able to predict the QoE metrics such as the number of interruptions, and the playback start time in real-time by restricting the tool to run in the node in the access network of the user.

The QoE of services using DASH was estimated by using the session logs collected from a node within the network and a server in [42]. The session logs were generated using a packet capture method. These logs consisted of the complete MPD file, and three timestamps for each video segment downloaded: interception time of the `HTTP-GET` request from the client (including the URL), the interception time of the first packet with the video payload (this gives the segment size), and the `OK` time stamp that indicates the end of the last segment. The session log data collected was used to reconstruct the DASH session that reflected the adaptation of the bitrate and the evolution of the buffer-filling (in seconds) over the duration of the session. Based on the reconstructed sessions, the authors were able to estimate the QoE metrics such as the number of interruptions, initial buffering duration, average duration of each interruption, number of bitrate switching events, and average re-buffering time due to user interaction.

In-network methods to measure the QoE of YouTube videos was presented in [84] by estimating the number of interruptions during the playback based on arrival time of TCP ACK messages.

The use of the in-network QoE estimation techniques eliminates the need to modify the existing video players or deploy additional software on the client or the server. These tools can be deployed easily by network providers who have greater control over the network compared to the user devices and are also platform independent. However, due to the limited processing capabilities of the nodes in the network and large packet processing times involved in Deep Packet Inspection (DPI), these techniques are best suited for off-line processing. Packet loss and caching between the network monitoring node and the client can pose significant challenges to the estimation of the playout buffer state using these tools.

### 2.4.2 Estimating QoE using Predictive methods

Assessing QoE based on the feedback of users is good at capturing the effect of the confounding factors but it is a time consuming and expensive process. It is also difficult to be replicated or scaled. In order to retain the advantages of subjective studies while enabling replication and scalability some studies have proposed predictive models that estimate the subjective MOS rating based on objective metrics.

The main challenge with such predictive models is to identify a relationship between the objective QoE or QoS metrics and the subjective MOS rating. This relationship was found to be non-linear. A generic expression that captures the exponential relation between the QoE and QoS parameters was proposed in [33] and is called the IQX hypothesis. Here, the QoE for streaming services is considered in terms of the MOS and is expressed as a function of loss and reordering ratio (caused by jitter). According to the IQX hypothesis, the change of the QoE depends on the current level of the QoE given the same amount of change in the QoS value but with a different sign, as shown in (2.1). If the QoE is already very high, then even a small disturbance can effect the QoE; however, if the QoE was already low to begin with, then a further disturbance will not be perceived.

$$\frac{\partial QoE}{\partial QoS} \sim -(QoE - \gamma). \tag{2.1}$$

It was demonstrated that the proposed relationship provided a better approximation when compared to the original logarithmic approximations presented in [58]. A similar pattern was seen in [59] where users watching the videos with better conditions would have less patience with the initial startup delay and would abandon sooner.

The authors in [58] presented a simple model that defined the relationships between one of the QoS parameters: bandwidth, response times and the QoE. In [87], this model was extended to capture the effect of multiple QoS parameters such as throughput and delay on the QoE. A new discrete scale called the Opinion Score (OS) that ranges from 0 to 5 was introduced to eliminate the constant $\gamma$ from (2.1). Using the Opinion Score values and applying the Multiple Linear Regression Model(MLR) from [17], a linear relationship between the QoE and multiple QoS parameters was defined as:

$$\log(\text{QoE}) = a_0 + a_1\text{QoS}_1 + a_2\text{QoS}_2 + ... + a_n\text{QoS}_n \tag{2.2}$$

where constants $a_i$ were estimated by the least squares method. On applying an exponential transformation on (2.2), the QoE/QoS exponential correlation is modeled as

$$\text{QoE} = e^{a_0} + e^{a_1\text{QoS}_1 + a_2\text{QoS}_2 + ... + a_n\text{QoS}_n} \tag{2.3}$$

The models in (2.1) and (2.3) were defined for web-pages and files. However, based on the following studies, it was found that the aforementioned models were applicable for video streaming services. In [14], the authors presented a similar non-linear model (2.4) using a *psychometric model* to estimate the MOS from the QoS parameters for HTTP-DASH streaming systems using any N parameters.

$$\text{QoE} = \sum_{i=0}^{N-1} a_i\text{QoS}_i^{k_i} \tag{2.4}$$

where $a_i$ are the constants and $k_i$ are the exponents for $N$ metrics.

The *Perceived QoE* model [88] defines a non-linear relationship between the MOS and objective QoE/QoS metrics. This no-reference model extracts parameters such as the bitrate $b$, frame rate $f$, packet loss rate $l$, video jerkiness $j$, and quantization parameter $q$, to approximately map the MOS. These parameters are combined by the following non-linear formula (2.5):

$$\text{MOS} = B \times b^{bb} + F \times f^{ff} + L \times l^{ll} + J \times j^{jj} + .. + Q \times q^{qq}... \tag{2.5}$$

The above formula uses specific weights (capital letters) and exponents that can be estimated according to the service and the device being studied. This composite metric helps capture the effect of multiple parameters on the MOS rating.

A preliminary model with limited scope that predicts the QoE in terms of MOS for progressive download videos has been presented in the ITU-T P.1201 standard [43]. This model considers sequences between 30 to 60 seconds and considers the interruptions, and playback start time.

A cross-layer monitoring architecture was proposed in [89] to monitor the packets at the node and the network level to measure the QoS metrics (packet loss ratio) and to build a physical and network level view. A mapping tool estimates the Decodable Frame Rate ($Q$) at the user for the streaming service based on the QoS measurements. The tool then predicts the degradation in the QoE in terms of the MOS from the Decodable Frame Rate ($Q$).

Pseudo Subjective Quality Assessment (PSQA) is a hybrid approach that combines subjective and objective evaluation [96]. It is used to predict the subjective ratings of videos using automation. In this approach, an initial set of reference videos are evaluated subjectively by users. These reference videos are samples of distorted videos whose objective metrics are known. The result of the initial evaluation is used to train a self-learning tool to predict the subjective rating in real time.

In [96] a framework for measuring the QoE of MPEG videos was presented. The framework consisted of a streaming server, monitoring nodes (called probes), a data collector server, a PSQA learning tool and a web-based reporting application called *Webstat*. The streaming server was used to deliver video content (MPEG) and audio content (MP3) using several protocols, HTTP, RTP, and UDP. The probes collected frame level metrics such as the loss rate (LR) of video frames, and the mean size of loss bursts (MLBS)), defined to be the average length of a consecutive sequence of frames lost but not part of a longer such sequence. The probes transferred the collected metrics to the data collector server using the Simple Network Transfer Protocol (SNMP). The perceptual QoE was calculated by the PSQA tool, and the computed QoE metrics were presented using the *webstat* application. Before the framework was used for real-time measurements, the PSQA tool was passed through a pre-processing stage during which it learned the effect of LR on the QoE. During this stage, a fixed set of videos with different LR and MLBS values were evaluated by a group of five human experts. The five experts provided the MOS values for each of the initial sequences. These MOS values were used as input to train a Random Neural Network (RNN) on the two variables, LR and MLBS, and mapped them into a perceived quality on a $[0, 1]$ range. Once the pre-processing was complete, the framework was ready to predict the MOS for the videos. During a real-time evaluation of the QoE, the probes collected the frame level information of the video streams and transferred the LR and MLBS metrics to the data collector server. Using the trained RNN model, the PSQA tool used the LR and MLBS metrics as input to generate the MOS in real-time.

The authors in [28], used the PSQA tool to predict the perceived quality of the videos streamed using a *Windows Media Player* that uses UDP to transfer the data. Objective metrics, such as *Application Layer Packets Lost* and *Application Layer Packets Retransmitted* were used to predict the MOS rating. Before applying this method in real-time, the MOS ratings of certain sets of videos (called the reference streams) with known objective metrics were evaluated by a group of participants. These volunteers were asked to rate the reference video streams twice, once without loss and once with loss. The MOS and the objective metrics for each stream were fed to a *Dynamic Time Warping* (DTW) predictor during its training stage. On completion of the training stage, the DTW predictor was ready to rate the streams in real-time. During real-time measurements, the application layer metrics were obtained by polling the *Windows Media Player* periodically. It was found that the DTW predictor matched the patterns of the metrics of the active streams with the reference streams to predict the MOS in real-time.

In [27], [35] the authors extended the PSQA to predict the QoE by using some parts of the videos rather than the complete videos. These parts were used in both training as well as in real-time prediction. With a set of preliminary experiments, it was demonstrated that the MOS could be predicted with 70-80% accuracy.

A_PSQA proposed in [23] is a reduced-reference model based on PSQA measures.

This model used the packet loss rate but can be extended to other network metrics such as the packet loss rate, mean loss burst size, end-to-end delay, and jitter. It was observed that the results of the A_PSQA model correlate very well with the subjective scores when compared to the full-reference models and no-reference models.

In [90], the authors used a no-reference PSQA model based on RNN to estimate the QoE for Adaptive HTTP over TCP video streaming. The parameters considered were the interruptions, average interruption duration, and the maximum interruption duration. Another QoE metric unique to Adaptive HTTP streaming, *Quantization Parameter* (QP), which controls the degree of video compression was also considered. The MOS collected from the users viewing an initial set of distorted reference videos was used to train the PSQA-RNN tool that was later used to predict the MOS for other videos.

In [20], the authors proposed a predictive model for the user-engagement as a function of the QoE metrics. The user-engagement is defined as the amount of time spent by the user in watching the video before quitting. In order to capture the effect of the QoE metrics they use a machine learning algorithm. The QoE metrics, such as the average bitrate, join time (playback start time), rate of buffering events, buffering ratio (ratio between buffering and play time), and frequency of buffering were collected using client-side instrumentation for both VoD and live sessions. They concluded that the main confounding factors are the type of video, device, and connectivity.

In [97], the authors presented an online learning QoE management tool that uses machine learning to understand the effects of the objective metrics and application layer metrics on the MOS ratings. The effect of objective metrics such as the video bitrate, audio bitrate, and frame rate on the perceived subjective QoE was studied. In the current model, continuous feedback from the users (MOS) was collected and served as a datapoint for the online learning algorithm. As the number of feedback instances increased so did the accuracy of the model. By using online learning algorithms they avoided the need for complex and expensive a priori subjective tests that use reference streams. They also demonstrated that the learning algorithms can retain the accuracy in estimation of the QoE without the use of a priori subjective studies.

In [12], the authors developed a framework that uses prediction models to realize a QoE-aware QoS management strategy. The framework would predict the MOS rating from the QoS parameters by using a statistical prediction model based on Discriminant Analysis. The QoS parameters considered were the encoding bitrate and the frame rates of the videos. A set of subjective measurements, where several users are presented with reference videos with degrading levels of QoS, were conducted. During these measurements the feedback from the subjects was recorded in the form of a binary response ("acceptable" or "unacceptable"). The main aim of these tests was to determine at what level of the QoS metrics and the perceived quality becomes unacceptable. The data collected from these measurements was then used as the input for the model that correlates the QoS parameters with the QoE perception. Using this model, the degree of influence of those QoS metrics had on the QoE was determined. The prediction model developed was used to realize a management strategy to control the QoS parameters that could guarantee a satisfactory QoE level.

A content-adaptive packet layer model to assess the QoE in terms of the frame rate quality for video services using RTP/UDP was presented in [101]. The bit-rate, packet loss rate, temporal complexity (the acuteness of temporal changes of a video sequence), and the frame type information of the transported video were determined from the packet headers. Using all these factors, a model to predict the MOS was developed.

All the studies done so far attempted to build models to capture the relationship between the QoS metrics, the objective QoE metrics, and the MOS. Due to the non-linear relationship between these metrics, it is not easy to construct a simple model. Considering the work done in the literature to date, it can be concluded that the relation between the QoS metrics and the QoE score follows a non-linear exponential relationship. The use of supervised learning techniques and neural network-based models could be used to predict the QoE in real time. The effect of confounding factors on the subjective QoE metrics also needs deeper investigation and the inter-dependence of the metrics needs to be studied further.

## 2.5 Summary

There has been a significant interest in developing tools to measure the QoE of online video streaming services. Measuring QoE metrics at the user-end is most suitable for capturing the effects of the end-to-end factors on the QoE. Obtaining direct feedback from users could be expensive and difficult to automate. As an effort automate the measurement of QoE, we developed an an active measurement tool - Pytomo (see Chapter 3), that measure the QoE of YouTube videos by crawling the YouTube website and emulating an embedded video player to collect the QoE for each of the crawled videos.

Table 2: QoE Measurement Studies: A Comprehensive View

| Ref. | Collection | | Data Collection Method | | | | | VDT | QoE Metrics | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | U | N | Passive | Active | DUF | PS | CL | | IBT | I | RF | AL | MOS | Other |
| [92] | ✓ | | ✓ | | | | | PD | | | ✓ | | | |
| [75] | ✓ | | ✓ | | ✓ | | | PD | ✓ | ✓ | ✓ | | ✓ | User-Viewing Activities |
| [29] | ✓ | | ✓ | | | | | RTSP | ✓ | ✓ | ✓ | ✓ | | |
| [28] [27] [35] | ✓ | | ✓ | | | ✓ | | RTSP | | | | ✓ | ✓ | |
| [90] | ✓ | | ✓ | | | ✓ | | DASH | | ✓ | ✓ | | | |
| [31] | ✓ | | ✓ | | | | | Multiple | ✓ | ✓ | ✓ | | | AverageBitrate, Rendering Quality |
| [69] | ✓ | | ✓ | | | | | Multiple | ✓ | | ✓ | | | Average Bitrate, FailureRate, & Failure to Start video |
| [51] [80] [52] | ✓ | | | ✓ | | | | PD | ✓ | ✓ | ✓ | | | |
| [83] | ✓ | | | ✓ | | | | PD | | | | ✓ | | |
| [74] | ✓ | | | | ✓ | | ✓ | PD | ✓ | ✓ | ✓ | | ✓ | |
| [40] | ✓ | | | | ✓ | | | PD | | ✓ | ✓ | | ✓ | |
| [39] | ✓ | | | | ✓ | | | PD | ✓ | ✓ | | | ✓ | |
| [96] | ✓ | | ✓ | | | ✓ | | Multiple | | | | | ✓ | LR, MLBS |
| [34] | | ✓ | ✓ | | | | | PD | ✓ | | | | | Bitrate Ratio |
| [82] | | ✓ | ✓ | | | | | PD | ✓ | ✓ | ✓ | | | |
| [62] | | ✓ | ✓ | | | | | PD | ✓ | ✓ | ✓ | | | |
| [84] | | ✓ | ✓ | | | | | PD | ✓ | ✓ | ✓ | | ✓ | |
| [42] | | ✓ | ✓ | | | | | DASH | ✓ | ✓ | ✓ | | | Bitrate switching, Rebuffering (User initiated) |
| [102] | ✓ | | | | ✓ | | | DASH | ✓ | ✓ | | | | Startup and average bitrate, bitrate switching events |

U=User,N=Network,VDT = Video Delivery Technique, PD = Progressive Download, DUF = Direct User Feedback, CL = Controlled Lab, PM = Passive Measurements, AM = Active Measurements, PS = Pseudo Subjective Quality Assessment (PSQA), IBT = Initial Buffering Time, I = Interruption, RF = Rebuffering Frequency, AL = Application Layer Metrics (Packet Loss)

CHAPTER 3

# PYTOMO: A TOOL FOR MEASURING QOE OF YOUTUBE VIDEOS

Measurement of QoE at the client end provides the advantage of capturing the effects of the end-to-end factors. In order to capture the users experience from a point closest to the user, we developed a client-end active measurement tool, Pytomo, that crawls the YouTube website and measures the QoE for YouTube's progressive download videos.

## 3.1 YouTube's Video Delivery Framework

YouTube is a video-sharing website that was founded in 2005 and was acquired by Google in 2006. As of 2015, YouTube is one of the most popular user generated content streaming service with more than 1 billion users [104].

Each YouTube video is associated with a base HTML page with a URL that is usually in the format `http://www.youtube.com/watch?v=video_id`. The base HTML page consists of multiple parts: the video description, embedded flash video player, user comments and a list of related videos. The embedded video player retrieves the URL of the video file from the base HTML page. The video is located on a different set of servers from the HTML pages and are referred to as video servers. A high-level overview of YouTube's video delivery framework and the video request pattern is presented in Fig. 1.

YouTube video streaming mostly employs a progressive download technique and

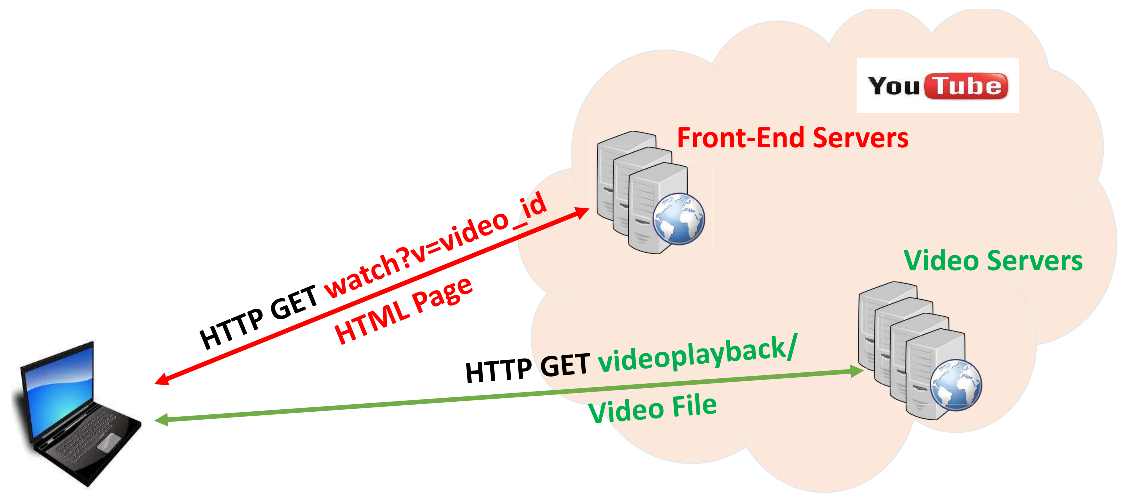


Figure 1: YouTube Video Delivery Framework

recently has upgraded most of its videos to support DASH. At the server end, the video files are delivered as regular HTTP objects. The servers do not maintain any state. The VCR like functionality provided to the user is handled by the embedded video player. In order to minimize the effect of jitter on the playback, the embedded player waits until certain amount of the buffer (*initial_buffer*) is filled before beginning the playback. Once the buffer level reaches *initial_buffer*, the playback is initiated. The data received from the HTTP connection is placed into the buffer, and the player reads the buffered video. As long as the amount of buffered data is more than *minimal_playout_buffer* the playback is maintained. Due to any perturbations in the network, if the buffer falls below the *minimal_playout_buffer*, the playback is interrupted. The playback is resumed when the buffer crosses the *minimal_restart_buffer* level.

The values for the various buffer levels employed by YouTube are not published. Hence, we used reverse engineering techniques and controlled evaluations to determine

Table 3: YouTube Buffer Parameters

| Buffer Level | Value |
|---|---|
| *initial_buffer* | 2 seconds |
| *minimal_playout_buffer* | 0.1 seconds |
| *minimal_restart_buffer* | 1 second |

the different buffer levels. For our evaluation setup we used the base HTML page and the embedded player provided by YouTube to create a test webpage. On the client machine we used a web proxy application [3] to force the client to retrieve the video file from a test sever instead of YouTube's video server. Under this scenario, we played the video multiple times and each time we modified the initial delivery rate, average delivery rate and the duration of interruptions during playback to understand the pattern of response from the embedded video player. Based on our evaluations, we determined the various buffer lengths of the YouTube player to be as listed in the Table 3.

## 3.2 Pytomo: Description

The current tool, Pytomo [51], is a Python based open-source tool that measures QoE as well as QoS of YouTube videos. Pytomo, is a platform agnostic, easily portable tool that emulates a user's video streaming session.

Pytomo is initialized with a bootstrapping phase where we consider an initial set of video URL's. These URLs by default are the most popular videos from the previous week based on YouTube's worldwide rankings. The aim of Pytomo is to collect the QoS metrics, QoE metrics and also information related to the Content Delivery Network (CDN) used by YouTube to serve each of these videos. For each of the videos being considered, Pytomo performs the following steps [51, 79]:

1. **Get video URL:** Pytomo downloads the base HTML page of the video and parses it. In order to support different access platforms, YouTube hosts multiple formats for each video. The URL for each of these is listed in the HTML file. Pytomo parses the HTML file to retrieve: video metadata (video duration, list of video formats, default quality etc.), the list of URLs associated with each video format. The default video format for the current platform is considered by Pytomo for the emulated video playback. The meta-data related to the video is logged for each video crawled.

2. **Domain Name Server(DNS) Resolution:** The hostname from the URL of the video file is resolved to the video server's IP address. In order to study the effect of the DNS resolver's IP resolution on the QoE of the videos, we use three different DNS resolvers: the ISP's default local resolver, Google Public DNS [4], and Open DNS [7]. Each of these IP addresses are logged. If the IP resolved with the three DNS resolvers is found to be different, the video file is downloaded from each distinct IP address.

3. **Collect QoS statistics:** Pytomo starts its video performance analysis by collecting QoS metrics based on Ping statistics (minimum, maximum and average delay) between the client and the video server. By default we use 10 packets, for the ping[1]. If the IP addresses returned by the DNS resolved in Step 2 is different, then the QoS is collected for each of the IP addresses.

[1]The number of packets is a configurable parameter and can be modified, if necessary.

4. **Collect QoE statistics:** Pytomo emulates the YouTube player based on the buffer parameters listed in Table. 3. The video playback session is emulated with the help of two different time scales associated with the playback buffer:

   - *D(t)*: Amount of video (playback duration) content downloaded into the buffer up to time *t*, i.e., the amount of video that is downloaded in terms of playback duration (obtained through the timestamps of video tags). *D(t)* reflects the rate at which the buffer is filled is dependent on the TCP connection between the client and the server.

   - *P(t)*: Amount of the video (playback duration) already played up to time *t*. *P(t)* reflects the rate at which the buffer is being drained. In order to ensure that this time-scale is consistent with an actual video player we parse the flags in the video file to determine which block of video is to played at each time unit.

   Initially, the player waits until *D(t)* reaches *initial_buffer* level, after which the playback is initiated. The duration for which the player waits before starting the playback is considered as the *initial_buffering_period*. Once the playback begins, the player is considered to enter the *playback stage*. During the *playback stage*, if amount of unplayed video $(D(t) - P(t))$ falls below the *minimal_playout_buffer* the playback is interrupted $((D(t) - P(t)) < minimal\_playout\_buffer)$. The playback resumes when the amount of video that has been buffered is greater than *minimal_restart_buffer* $(D(t) - P(t) > minimal - restart - buffer)$. By keeping track of the state of the playback, we are able to measure the QoE metrics: the playback start time, the number of interruptions during video playback, duration of the interruptions. By default, we limit the duration of the video session to the first 30 seconds of playback.

It was observed that YouTube uses HTTP redirection on their video servers (possibly for load balancing). In case of HTTP redirections, the server serving the video could be different from the IP resolved in the Step 2. In such scenarios, Pytomo re-collects the QoS parameters from the previous step with the IP address of the redirected server.

5. **Logging the data:** The QoS and QoE metrics collected from the previous steps are logged into a text-based log file and local SQLite [9] database file.

6. **Continue Crawl with Related Videos:** *Pytomo* continues the crawl by collecting the QoS and QoE metrics for related videos. Once the video playback is completed *Pytomo* parse the related video list to retrieve a subset of the related videos. These videos are added to the Videos related to the current video (obtained through YouTube API) are then added to the list of videos to be crawled.

For each video and the subset of related videos, the 6 steps listed above are executed. Pytomo thus continues crawling YouTube website while collecting the QoS and QoE metrics for each of them. This process is repeated until the maximum number of videos as configured by the user are crawled.

Pytomo is an active QoE and QOS measurement tool that is currently designed for YouTube progressive download videos. Pytomo enables us to automate the QoE measurement without the need for a user to watch a video. The use of automation ensures higher accuracy, repeatability and better scalability. The cross-platform design of Pytomo allows the researchers, network engineers and service providers to deploy the tool onto any client machine to study the user perceived video quality.

# CHAPTER 4

# CASE STUDY: QOE ACROSS THREE INTERNET SERVICE PROVIDERS IN A METROPOLITAN AREA

In Chapter 3, we presented the design of *Pytomo*, an active client-end QoE measurement tool for the YouTube video streaming service. In this chapter, we present a case study that investigates the QoE of the users connected to three different Internet Service Providers (ISPs), located in the Kansas City, USA metropolitan area.

## 4.1 Data Collection

In this study, we identified volunteers located across Kansas City metro area. Kansas City is a metropolitan area that spans the border between the U.S states of Missouri and Kansas. With a population of 2.75 million, Kansas City ranks as the second largest metropolitan area in Missouri (after Greater St. Louis) and is the largest with territory in Kansas.

Based on our preliminary assessment, we found that the local time between 8:00 PM and 10:00 PM, is the most common prime time for home Internet services. In consistent with this trend, we asked the volunteers to run the *Pytomo* tool during this specific time window. The tool was run on their personal machines connected to their respective residential ISPs.

The first set of collection was performed by 19 users during prime time on December 8, 2011. A total of 1,260 videos were downloaded from 442 distinct video servers.

The second collection was performed on the evenings of March 14 and 15, 2012 by 32 different users. This collection consisted of a total of 2,390 videos from 988 distinct video servers. For both sets of data collection, the video downloads were limited to 30 sec for each video. *Pytomo* started each crawl by initializing the playlist with 10 most popular videos of the week, followed by selecting 2 random videos out of the related videos for each video crawled. With this setting, we ensured that at least 10 videos were crawled from each volunteers's computer. *Pytomo* crawled on average 60 videos from each volunteer's computer. At the beginning of each crawl session, the user was prompted to enter the ISP that they subscribe to. The volunteers were connected to three different ISPs, with 5, 9, 4 users in the December crawl and 8, 19, 5 users in the March crawl who were connected to ISP-1, ISP-2, and ISP-3, respectively. Of these three ISPs, two of them (ISP-1 and ISP-3) were cable-based providers while ISP-2 was a DSL-based provider. Based on preliminary tests, we determined that a small time gap before moving from downloading one video to the next was necessary, so that YouTube's servers did not think that it was a bot crawling their servers and therefore, did not block access for any requests from each user's computer. This delay between consecutive requests is even more critical when crawling a large number of videos as YouTube was found to prompt a Captcha [2] message if the delay was not introduced.

## 4.2 Results and Analysis

In this section, we present our analysis and observations on the QoE of the residential users and how the variation in the video server selection dynamics in relation to the ISPs. We determine the relation between the variation in the QoE and video server selection patterns across the two different data collection periods for the individual ISPs.

Before we present our analysis, we present our observations on the naming convention used by YouTube at the time of our study. The actual video file URL was found to be typically of the form: `http://o-o.preferred.SERVER_CODE.v[1-24].lscache[1-8].c.youtube.com`, which identifies the domain name of the actual server to be in the form `o-o.preferred.SERVER_CODE.v[1-24].lscache[1-8].c.youtube.com`. Here, `SERVER_CODE` identifies a server ID, behind which, is likely, a cluster of video servers in a data center environment. The naming convention of `SERVER_CODE` seems to indicate the geographic location of video server clusters, that are commonly named by including IATA's 3-letter airport codes usually at the beginning; for example, the IATA airport code `dfw` refers to Dallas-Forth Worth-Texas, `iad` refers to Washington, DC, `ord` refers to Chicago-Illinois, and so on (see Table 4); this was also noted in [11]. While these are certainly logical domain names, the naming convention seems to indicate that they are located in the geographic region identified via the IATA airport code. The `SERVER_CODE` also gives us an indication of where or how far the video servers are from the users.

Table 4: Access Patterns for The Video-Server from 3 ISPs with QoE Metrics (± denotes 95% confidence interval)

| | **December 2011** | | | | | **March 2012** | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| **Server code** | **No. of Samples** | **Playback Start Time (sec)** | **% Videos Interrupted** | **RTT (ms)** | **Download Rate (Mbps)** | **No. of Samples** | **Playback Start Time (sec)** | **% Videos Interrupted** | **RTT (ms)** | **Download Rate (Mbps)** |
| **ISP-1** | | | | | | | | | | |
| all (ISP-1) | 654 | 1.18±0.09 | 10.24 | 160.39±9.05 | 1.35±0.08 | 1468 | 0.49±0.05 | 1.90 | 68.89±5.61 | 2.60±0.20 |
| `dfw06g01` | 41 | 1.45±0.71 | 9.75 | 100.63±19.67 | 1.30±1.16 | - | - | - | - | - |
| `iad09g05` | 613 | 1.15±0.09 | 10.29 | 164.43±9.48 | 1.35±0.08 | 213 | 0.65±0.09 | 2.81 | 104.92±22.40 | 1.82±0.22 |
| `xo-ord1` | - | - | - | - | - | 1213 | 0.44±0.05 | 0.99 | 62.83±4.68 | 2.82±0.24 |
| `lax04s12` | - | - | - | - | - | 20 | 0.97±0.16 | 0.00 | 80.44±1.52 | 1.69±0.46 |
| `ord12so5` | - | - | - | - | - | 22 | 1.44±0.55 | 45.45 | 43.84±1.10 | 0.55±0.13 |
| **ISP-2** | | | | | | | | | | |
| all (ISP-2) | 347 | 0.77±0.07 | 1.80 | 85.33±2.64 | 1.50±0.08 | 602 | 1.23±0.07 | 16.28 | 65.24±2.52 | 1.78±0.24 |
| `dfw06g01` | 175 | 0.89±0.09 | 2.85 | 90.59±6.26 | 1.35±0.08 | - | - | - | - | - |
| `iad09s12` | 6 | 1.50±0.31 | 0.00 | 77.15±20.78 | 1.14±0.21 | - | - | - | - | - |
| `lga15s20` | 36 | 0.62±0.45 | 5.56 | 72.58±9.46 | 1.76±0.54 | - | - | - | - | - |
| `dfw06s10` | 3 | 0.85±0.74 | 0.00 | 35.33±5.74 | 1.58±2.37 | - | - | - | - | - |
| `dfw06s08` | - | - | - | - | - | 46 | 1.42±0.20 | 4.34 | 63.46±6.13 | 0.96±0.06 |
| `mia05s05` | 65 | 0.42±0.06 | 0.00 | 79.07±15.48 | 1.78±0.23 | 93 | 0.67±0.67 | 1.07 | 69.61±3.41 | 2.52±0.55 |
| `sjc07s11` | 62 | 0.83±0.10 | 0.00 | 87.65±6.90 | 1.50±0.18 | 6 | 2.71±2.63 | 33.33 | 122.15±34.63 | 1.06±0.56 |
| `sjc07s15` | - | - | - | - | - | 147 | 2.91±0.54 | 43.54 | 99.29±7.08 | 0.67±0.07 |
| `atl14s01` | - | - | - | - | - | 88 | 0.34±0.05 | 0.00 | 57.65±2.39 | 2.77±0.64 |
| `atl05s01` | - | - | - | - | - | 28 | 0.34±0.05 | 0.00 | 57.84±1.25 | 2.42±1.06 |
| `ord12s05` | - | - | - | - | - | 44 | 1.30±0.34 | 54.54 | 44.46±1.06 | 0.50±0.04 |
| `ord12s06` | - | - | - | - | - | 109 | 0.52±0.12 | 4.35 | 45.18±1.57 | 1.82±0.26 |
| `xo-ord1` | - | - | - | - | - | 41 | 0.34±0.05 | 0.00 | 66.54±16.37 | 3.86±1.36 |
| **ISP-3** | | | | | | | | | | |
| all (ISP-3) | 259 | 0.78±0.07 | 0.77 | 78.27±2.64 | 1.68±0.17 | 320 | 0.63±0.05 | 0.63 | 109.657.05 | 1.84±0.20 |
| `ISP3-dfw1` | 259 | 0.78±0.06 | 0.77 | 78.27 ± 2.65 | 1.68±0.17 | - | - | - | - | - |
| `ISP3-mia1` | - | | - | - | - | 132 | 0.54±0.05 | 1.53 | 95.07±11.72 | 2.02±0.35 |
| `ISP3-sjc1` | - | | - | - | - | 188 | 0.68±0.07 | 0.00 | 119.90±8.67 | 1.73±0.25 |

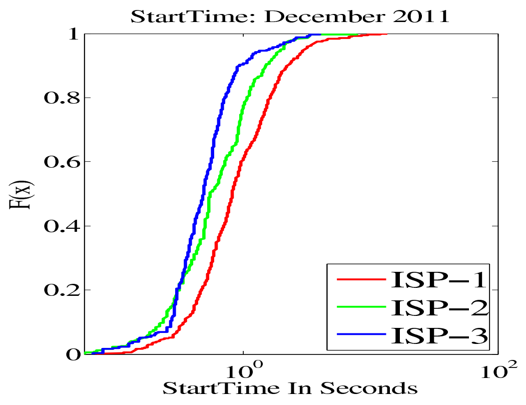


Figure 2: StartTime : December 2011

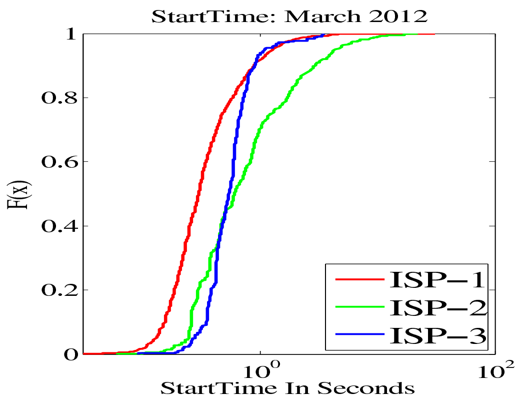


Figure 3: StartTime: March 2012

#### 4.2.1 Analysis of User's QoE

The factors that we considered to compare the QoE of the users were: *EncodingRate*, *StartTime*, *InitialRate*, *DownloadInterruptions*, and *DownloadRate*. Since YouTube supports multiple formats for the same video, the format selected can also influence the QoE of the user.

- ***EncodingRate***: The encoding rate represents the quality of the video and is the average amount of data required to playback one second of the video. We found that the default quality of videos downloaded by users of different ISPs were similar. Since the video format selection is dependent on the type of the machine and YouTube's estimation of the data rate to the user, we can say that the machines and the networks used by the users are homogenous as perceived by YouTube. This confirms that any variation in the QoE dependent on the network or video server performance and not on the video quality.

- ***StartTime and InitialRate***: Table 4 shows the average start time (with 95% confidence interval) observed for users accessing YouTube from different IPSs when video servers are located in different geographic areas (more on server locations will be discussed in. The CDF of *StartTime* (Fig. 2 & 3) reflects the overall distributions for the two time periods. While in most cases, the average start time was less than 1 sec., we found that for ISP-2, there were at least two server locations that were experiencing an average start time of 2 sec or more in the March 2012 crawl. Although the average *StartTime* for all servers for ISP-1 and ISP-3 had reduced from December 2011 to March 2012, for ISP-2 it had increased. For the most popular servers out of these three ISPs, servers used for ISP-3 had significantly faster *StartTimes*.

- ***Interruptions and DownloadRate***: From the data collected, we also calculated the percentage of video interruptions experienced, which is shown in Table 4. During the March 2012 crawl, videos served for ISP-3 customers had the least number of interruptions, while at the same time, the number of interruptions was relatively higher for ISP-2 customers. From the table, we can also observe how each of the video servers contributed to the interruptions. For example, the new server ID `sjc07s15` (i.e., San Jose, California) that served most videos in the March 2012 crawl for ISP-2 had an unusually high number of interruptions at 43.54%. Not surprisingly, this affected the average download rate from this server ID, which was among the lowest. Also, there was a strong correlation between the percentage of interruptions and the download rate.

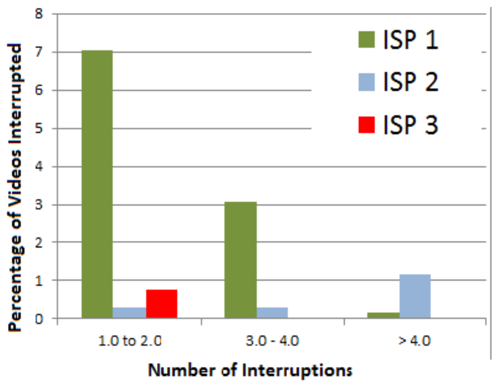


Figure 4: Download Interruptions: December 2011

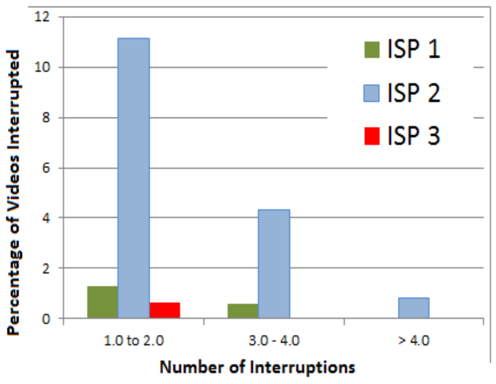


Figure 5: Download Interruptions: March 2012

While we do not have additional data to correlate whether this anomaly is due to the network or servers, we can make a preliminary inference in the case when multiple server IDs are located in the same city. We note two server IDs located in Chicago: `ord12s05` and `ord12s06`; the former one is used by both ISP-1 and ISP-2, while the latter one is used only by ISP-2. Since accessing `ord12s05` shows high interruptions (and a low download rate) for both providers, but low interruptions in case of `ord12s06`, it is very likely that `ord12s05` is experiencing high load or some anomaly. Thus, we were able to isolate the set of servers that were the cause for reduced QoE.

In Figs. 4 and 5, we present severity of interruptions by categorizing the number of interruptions (shown in percentage) into three groups: 1 to 2 (“low”), 3 to 4 (“moderate”), and more than 4 (“high”). We observe that the severity of interruptions was often low, while for a notable number of cases, severity was moderate for ISP-1 in the December crawl and for ISP-2 in the March crawl.

### 4.2.2 Two-Sample $t$-test

In order to determine if the difference in the QoE between December 2011 and March 2012 crawls was significant, we performed the two-sample $t$-test between the two datasets to determine the $p$-values (see Table 5). For the $t$-test the most commonly used significance levels are 0.1, 0.05 and 0.01. We select the significance level as 0.01 because it is the most conservative one. We found that for ISP-1 and ISP-2, $p$-value $\ll 0.01$ for StartTime. Thus, we can say that StartTime in the March 2012 crawl was highly, statistically significantly different than in the December 2011 crawl. While for ISP-1, StartTime was less in March 2012, it was higher for ISP-2. There was no statistical difference for ISP-3. These results are consistent with Fig.3.

Table 5: $p$-value for two-sample $t$-test

| **$p$-value** | **ISP-1** | **ISP-2** | **ISP-3** |
|---|---|---|---|
| StartTime | $2.2 \times 10^{-16}$ | $1.18 \times 10^{-8}$ | 0.44 |
| DownloadRate | $2.2 \times 10^{-16}$ | 0.52 | 0.22 |
| Average. RTT | $2.2 \times 10^{-16}$ | $1.35 \times 10^{-08}$ | $4.19 \times 10^{-13}$ |

$H_0$: $\mu_{Dec} = \mu_{Mar}$ vs. $H_{alt}$: $\mu_{Dec} \neq \mu_{Mar}$

In regards to the download rate for ISP-1, the March 2012 crawl was highly statistically significantly different (actually higher) than in the December 2011 crawl. However, there was no statistical difference for the download rate for the other two providers. In regard to the Average RTT, the March 2012 crawl was highly statistically significantly different (actually higher) than in the December 2011 crawl for all the providers.

### 4.2.3 Video Server Selection Dynamics

Besides determining QoE at the provider level as discussed above, we also studied how different video server selections impact QoE for different providers. In addition, we studied the change in the server dynamics observed between the two crawls by users connected to the same ISP. We first briefly comment on whom the video servers belong to. We analyzed the mapping from the IP addresses of the servers and the autonomous systems (AS) that they were associated with. We found that for the December 2011 data set, there were three ASes 15169, 36040, 43515 – all are owned by YouTube or its parent company, Google. For the March 2012 data set, the same three ASes showed up, except that we found a non-YouTube autonomous system (AS 2828) to respond to about 2% of the requests. This verifies that CDN is largely owned by YouTube as reported in [94], except that YouTube still seems to have some partnerships with others for some content distribution for serving the Kansas City metropolitan area.

#### 4.2.3.1 Video Server Selection based on ISP

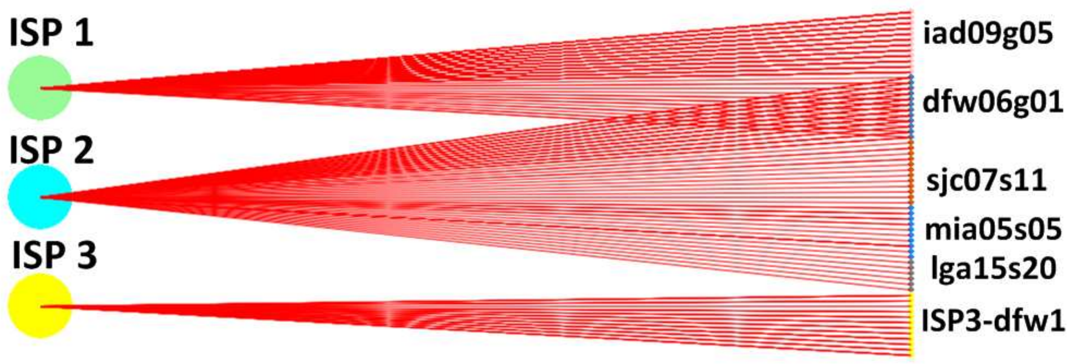


Figure 6: Server access pattern for top ten videos: December 2011

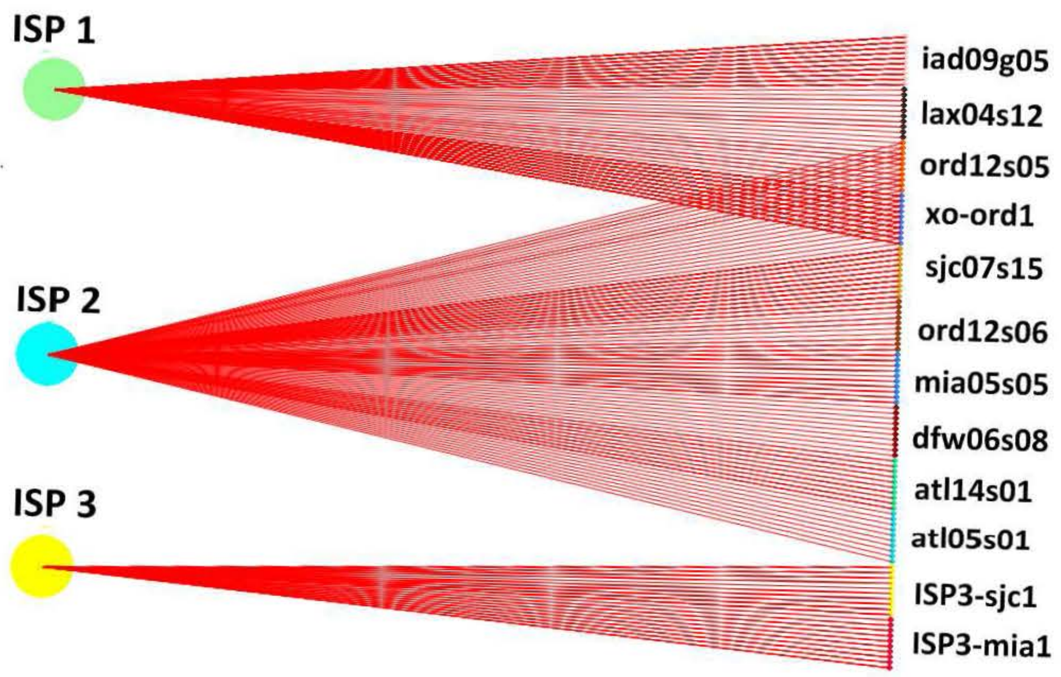


Figure 7: Server access pattern for top ten videos: March 2012

We observed that video servers that were accessed varied significantly between users of different ISPs located in the same geographical location. We next discuss our results by dividing the set of videos into two different groups: the top ten most popular videos for the week and the rest. The top ten videos were accessed by *all* users irrespective of the ISP provider they are associated with.

Fig. 6 shows the access patterns for the ten most popular videos of the week from the December 2011 crawl. The nodes on the left indicate the ISPs from where users are gaining access while each small dot on the right indicates a server cluster that is organized by server IDs. Here we see that except for the server ID `dfw06g01` that was accessed by users in ISP-1 and ISP-2, no other video servers were accessed by users of more than one

ISP. The observations for the March 2012 crawl are shown in Fig. 7. We noted a similar pattern while the common set of server IDs was found to be different and the access spread out to a larger number of video servers.

Based on these observations, we can infer that even when accessed from the same geographic area, the selection of the video servers for the most popular videos was dependent on the ISP and not on the geographic location of the users. If we consider the access patterns for the top ten videos in March 2012, ISP-1 and ISP-2 were served from two common video servers (`xo-ord1, ord12s05`) for some videos while users in ISP-3 did not access any common video servers. Furthermore, by the presence of ISP-3's name in the `SERVER_CODE` of the ISP (not shown here to maintain anonymity) and distinct IP addresses for these servers, which were not observed to be associated with other ISPs, we inferred that these video servers are dedicated to serve requests from users located in this ISP; this could possibly be due to a business partnership between this ISP and YouTube.

When all videos were considered, we found that the server allocation patterns were similar to the ones found for the top ten videos. From Table 4 (March 2012 crawl), we see that out of all video servers that were accessed by users of ISP-1 and ISP-2, only two are common (`xo-ord1, ord12s05`) (similarly, for the December 2011 crawl). Users of ISP-3 are served by a dedicated set of video servers.

From the above observations, it is safe to conclude that video servers at YouTube are divided into cluster groups with each cluster dedicated to different ISPs, or shared in some cases between a group of ISPs. While we can come to this conclusion, the question remains on how providers and YouTube worked together to invoke preferred video server selection. We believe that this was done using DNS. For example, Bind version 9 of DNS introduced a feature called `Views` [66]. This can be used to channelize a request from a source IP address or a group of IP prefixes (that resides in a particular ISP) to resolve to a particular domain name and IP address. In other words, there seems to be an agreement between YouTube and large ISPs so that requests for videos from end users from a particular ISP can be directed to a particular server or a set of specific servers. While [94] discussed that selections are based on DNS, their work does not identify that there are potential business agreements in place between YouTube and ISPs to make this happen. To our knowledge, we are the first to point out that such agreements may be in place between YouTube and large ISPs to ensure that certain servers are preferred based on the source IP addresses of an ISP.

#### 4.2.3.2 RTT and the Location of Video Servers

We observed that the value of the average RTT does not play a significant role in the selection of the video server. For the three different ISPs, servers that were serving the most number of videos are not necessarily the ones that had the lowest RTT. For ISP-1 and ISP-2 in the December crawl, we see that the most popular server clusters `iad09s12` and `dfw06g01` had the highest average RTT values. Also, servers that had the lowest average RTT did not have the best average data rates.

In Fig. 8, we show the scatter plot of RTT and the download rates for all the videos downloaded from the most popular server `dfw06g01`, and other servers by all users in the December 2012 crawl for ISP-2. From this plot, we see that the RTT to the most popular server was not always the lowest. In the March crawl, a similar observation was made for ISP-1, ISP-2, and ISP-3 where the popular clusters were the ones with a relatively higher RTT. Based on this observation, we infer that users were predominantly directed to a specific server cluster depending on the originating provider, which was not necessarily based on the average RTT.

Our observation contradicts the observation in [94], where the authors observed that the server with the lowest RTT values was the preferred one. The previous work considered RTT from the edge of the network, *not* from the end users' computers. Our measurement was conducted from the end users' perspective and confirms the findings in [51]. Thus, it is possible that the network segment from the end users to the edge of the network contributed to the overall RTT. Secondly, the edge of the network to the video server is not a good indicator for how end users would perceive RTTs.

#### 4.2.3.3 Change in Video Server Dynamics

Since the two crawls were performed with a gap of three months, we were able to study the change in the YouTube distribution architecture over this time span for all three ISPs considered (see Table 4).

For ISP-1, there were two video server clusters in the December 2011 crawl; of these two, only one of them was found in the March 2012 crawl along with three new ones. Similarly, for ISP-2, there were six video server clusters in the December 2011 crawl; three of them were found in the March 2012 crawl along with six new ones. This suggests that video servers allocated to the same ISP also change over a short period of

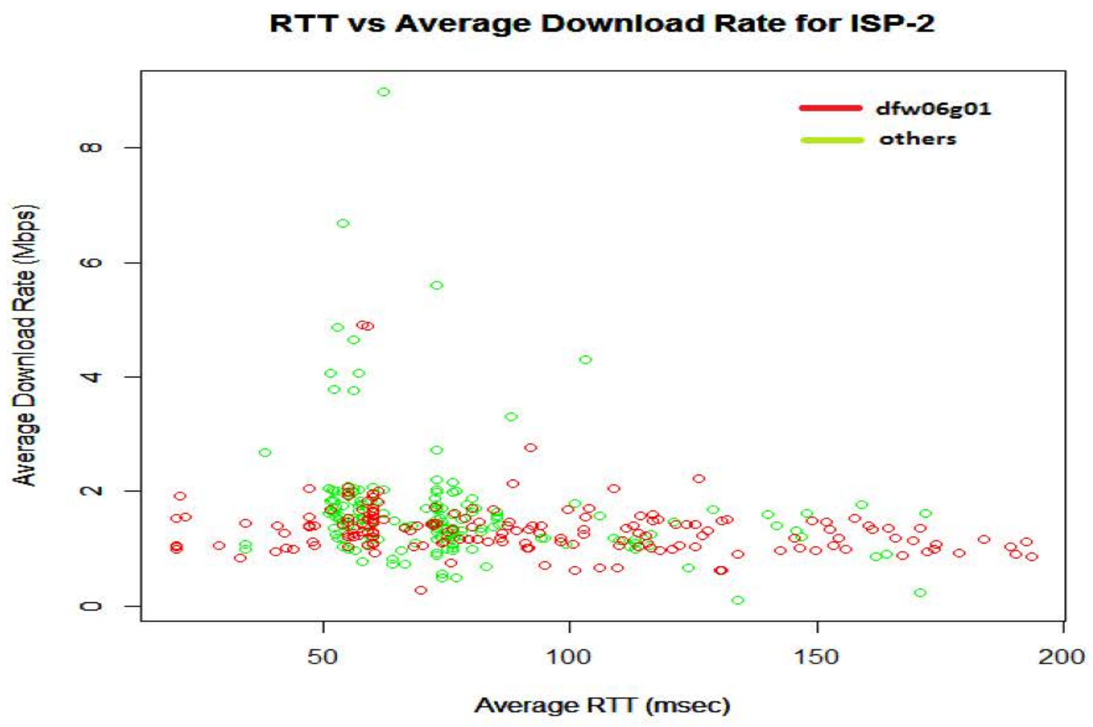


Figure 8: RTT (msec) and Download Rates (Mbps) Comparison for the December 2012 Crawls for ISP-2
time (three months).

Note further that users of ISP-3 were always served by dedicated servers. Servers observed in March 2012 were found to be different compared to the ones in December 2011. Based on the location `SERVER_CODE`, we can also infer that the locations of these servers were different. From this, we infer that YouTube has a dedicated set of servers in multiple locations for ISP-3 customers. These servers, at any given time, were exclusive to this provider irrespective of the popularity of the videos. This substantiates the observation we made in the previous section on policies being in place between YouTube and its providers.

### 4.2.4 Limitation of the Study

Recall that our study is limited to a particular geographic area with users accessing from residential providers in an evening time frame, and that this is an observational study and the data collected is a voluntary and convenient sample. Clearly, observations may differ for different geographic areas. Secondly, we limited our data collection to first 30 sec of each video; observations may differ when the entire video is considered. Our crawl mechanism limits the number of videos crawled, allowing us to observe only a representative set of server locations – it is not meant to capture the entire CDN architecture of YouTube.

The purpose of this case study is to demonstrate that our proposed approach can effectively measure end-users' QoE and can analyze, for example, the impact of provider distribution policies on end-users' QoE. In order to objectively evaluate the impacts of ISPs distribution policies, we recognize that enforcing randomization, such as how Nielsen conducts TV show ratings, is necessitated.

### 4.2.5 Summary

Using *Pytomo* we are able to objectively evaluate the QoE of users accessing YouTube videos in an automated web-crawl. We can also measure the QoS metrics and retrieve the CDN related information. We were able to study the variation in the QoE of YouTube videos when accessed from the same ISP in the same geographic location across different time periods. Based on our analysis, we were able to isolate the video servers that were causing degraded QoE and identify the location of these video servers. We also would like to point out that this was the first study that studied the QoE of YouTube videos and their distribution policies with data collected using active analysis from actual user computers in residential areas.

CHAPTER 5

# ADAPTIVE BITRATE ALGORITHMS FOR DASH

With HTTP based *progressive download* streaming, the video file is treated as a large web object with the video player playing the file while simultaneously downloading it [52]. The video player provides functions such as play, pause, and seek. As long as the rate at which the video is being downloaded is marginally higher than the playback rate, the user experiences seamless playback. However, any reduction in the throughput results in buffer underflow leading to an interruption in the playback. These interruptions, termed as *buffering events*, could result in significant degradation of the users' Quality of Experience (QoE) [39]. Although the *progressive download* technique provides all the benefits of HTTP, it does not support dynamic adaptation of the playback quality according to the network conditions, thus, resulting in playback interruptions due to perturbations in the network.

Dynamic Adaptive Streaming over HTTP (DASH) allows the video player to adapt the bitrate based on the current network conditions and user preferences. Adobe's HTTP Dynamic Streaming (HDS), Microsoft's HTTP Smooth Streaming (HSS), and Apple's HTTP Live streaming are the most popular commercial implementations of DASH. The open standard for HTTP based adaptive streaming, MPEG-DASH [78] was released by MPEG in 2011. In the last few years, more and more video service providers are shifting to DASH. Netflix, YouTube and Hulu are some of the leading video service providers that currently use DASH.

The components of a typical DASH-server are depicted in Fig. 9. As discussed briefly in Section 1.2.2, a DASH server encodes each video into multiple representations that vary in bitrates (encoding rates), display resolution, video dimensions etc. Each representation is split into multiple smaller video chunks called *segments*. Each segment is of a fixed playback duration (1, 2, 4, or 10 seconds). The server also creates a Media Presentation Description (MPD) file for each DASH video. The MPD file is an XML representation of the metadata of the video that includes information about the playback duration, available representations, minimum bandwidth required for each representation, segment duration, codec, etc. The MPD file also contains the URL for the individual segments.

On the client-side, the DASH player determines the set of representations that can be used for its specific platform and depending on the network characteristics, the player downloads the appropriate bitrate for the next DASH segment to be downloaded. The choice of the segment to be downloaded plays a critical role in the QoE management by the DASH player. DASH clients employ adaptive bit rate (ABR) algorithms to determine the appropriate representation for each segment to be downloaded. Consider a simple case where two clients (*client1 and client2*) are accessing the same video from different access networks as shown in Fig. 9. In this case the web-server is hosting a DASH video with three different representations (1Mbps, 2Mbps, and 5Mbps). The server also hosts an MPD file with the metadata of the DASH video. Both clients start by playing the video with the lowest representation to ensure the lowest playback start time. While downloading each segment, the clients measure the network throughput based on the segment fetch times. If the network path for *client1* is capable of supporting the highest video quality, the player will soon shift to the highest bitrate. Whereas for *client2*, if the ABR algorithm determines that the network can only support 2Mbps it switches to the appropriate representation (lower than the highest quality). The ABR algorithm used by each client continuously measure the network throughput to determine the bitrate suitable for each segment. Thus, the client can shift the bitrate at the boundary of each segment.

The aim of an ABR algorithm is to manage the users' QoE by dynamically adapting the bitrate of the video. The users' QoE is typically determined by metrics such as initial delay (time taken to begin playback), frequency of rebuffering events, average playback quality, number of bitrate switching events, and convergence time (time taken to reach the optimal quality) [54]. The bitrate switching events are either positive (switching to higher bitrate) or negative polarity (switching to lower bitrate). Since the three different QoE metrics could be competing with each other, a typical ABR algorithm tries to maximize the average quality of playback while minimizing the other metrics: playback start time, duration and number of buffering events, and the number of bitrate switching events.

As long as the time taken to download the next segment is less than the amount of data buffered, the player does not face interruptions in playback. An ABR algorithm could use the observed network throughput and buffer occupancy in estimating the bitrate for the next segment so that the users' QoE (determined) is maximized. In the following sections, we present some of the related works in ABR algorithms. We then study QoE

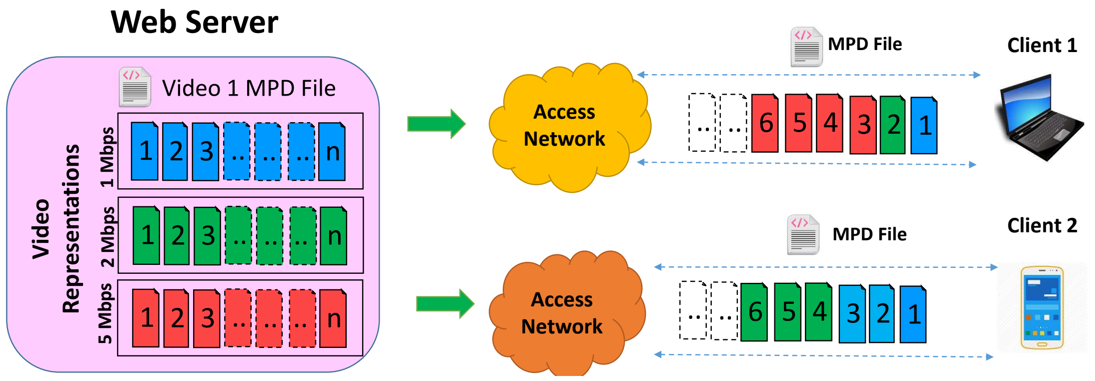


Figure 9: Overview of a DASH System

management with popular ABR algorithms in the literature and in Chapter 6 present our Segment-Aware Rate adaptation (SARA) algorithm that improves the QoE of the user especially in low bandwidth networks.

## 5.1 Related Work

In this section, we discuss existing literature related to ABR algorithms in DASH. The DASH standard does not specify any mechanism for rate adaptation, hence, the client is allowed to use any adaptation scheme. Due to this flexibility, any ABR algorithm can be easily implemented in the player with minimum support or modification on the server end. All the popular video service providers use proprietary ABR algorithms to enhance the users' QoE. Two of these proprietary players and an open-source player (Adobe OSMF) were compared in [13] in terms of their QoE management capabilities. It was observed that the players were either too slow to converge, leading to frequent bitrate switching events or were too slow to start the playback. These observations demanded a need to develop ABR algorithms than can maintain better QoE under varying network conditions.

Most ABR algorithms rely on the network throughput measured at the application layer. Since HTTP is an application layer protocol, these algorithms use the segment fetch times to estimate the network bandwidth. Optimistic estimation of the network bandwidth could result in the ABR algorithm requesting a higher bitrate than what the network could support resulting in playback interruptions. On the other hand, pessimistic estimations could result in lower video quality. In [68], the segment fetch times are used to estimate the network throughput that is in turn used as an input to an additive increase and aggressive decrease algorithm. Using the average value of the segment throughput estimations could result in the favoring the outliers. Hence, a throughput based approach with a harmonic mean download rate to overcome the effect of outliers is used in [48]. In [76], the authors proposed a QoE-aware version of the adaptation algorithm used by OSMF [8] called QDASH. The ABR used by OSMF is a throughput based additive increase and one-step decrease algorithm. QDASH uses in-network throughput measurements to estimate the best bitrate.

Due to the high variability in network conditions and difficulty in accurate bandwidth measurements, a pure buffer-based ABR algorithm is presented in [41]. However, due to the lack of enough buffer information during the initial stages, this approach relies on the network metrics during the starting phase. This approach uses a rate map structure that maps the current buffer occupancy to the optimum bitrate to estimate the next segment's bitrate.

Since the sizes of the individual segments (even for same bitrates) can vary significantly [41, 56], the download times over HTTP for various segments can also vary significantly [38]. The variation in the segment download times (used to indirectly determine the available bandwidth) is more critical for rate adaptation in low bandwidth environments.

In the following sections, we present two of the most popular ABR algorithms in the literature. Later in Chapter 6, we discuss our novel ABR algorithm that improves the QoE of DASH video streaming services.

## 5.2 Throughput Based Rate Adaptation (TBA)

The Throughput-Based rate Adaption (TBA) (Algorithm. 1) is an ABR algorithm that determines the next bitrate based on the measured throughput for the most recent segment downloaded. TBA is an additive up-shift and aggressive down-shift algorithm and is based on the algorithm presented in [68].

Before starting the video playback, the video player retrieves and parses the standard MPD file from the server. From this MPD file the player retrieves the metadata of the video: media playback duration, segment duration, video dimensions, list of representations and the URLs for each of the segments. The representations and the suggested bitrates for each representation are stored in the set $\mathbb{R}$ = $\{r^{min}$, . . . $r^{i}$ . . ., $r^{max}\}$.

Each segment that is downloaded is placed into a video buffer ($B$). The size of this buffer is limited. The current buffer occupancy, i.e., the the number of unplayed segments is given by ($B_{curr}$). With TBA algorithm, an initialization threshold ($B_{init}$) is defined. Before downloading each segment, the video player invokes the TBA algorithm to determine the appropriate representation to be downloaded for the next segment. As long as the

**Algorithm 1:** Throughput Based Rate Adaptation Algorithm [68]

```
Input:
    $Rate_{prev}$: The bitrate of most recent segment
    $B_{curr}$: Current buffer occupancy
    $T_n$: Average Throughput of the most recent $n$ segments
Initialization:
if $B_{curr} \leq B_{init}$ then
    $l_{n+1} = r^{min}$
else
    if $T_n > \epsilon Rate_{prev}$ then
        if $R_{prev} = r^{max}$ then
            $l_{n+1} = r^{max}$ // Bitrate already at maximum
        else
            $l_{n+1} = R_{prev} \uparrow$ // increase by one level
    else if $T_n \geq R_{prev}$ then
        $l_{n+1}$ = $R_{prev}$;
    else
        if $R_{prev} = r^{min}$ then
            $l_{n+1} = r^{min}$ // Bitrate already at minimum
        else
            $l_{n+1}$ = max$\{r^i | r^i < T_n\}$ // decrease to maximum possible
                bitrate
Result:
    $l_{n+1}$: the bitrate of the next segment to be downloaded
```

current buffer occupancy ($B_{curr}$) is below the initialization threshold ($B_{init}$), the lowest bitrate is downloaded. The segment with the lowest bitrate is typically the smallest and by selecting it we ensure that the playback start time is minimized. While downloading each segment, the player measures the throughput observed and the average throughput calculated from the most recent $n$ segments is saved as $T_n$. Before downloading any subsequent segments, the player invokes the TBA algorithm module with the following input parameters: current buffer occupancy ($B_{curr}$), the bitrate used for the previous segment ($Rate_{prev}$), and the average throughput observed for the most recent $n$ segments ($T_n$).

Once $B_{curr}$ increases beyond threshold $B_{init}$, the additive increase phase is initiated. In this phase, if the observed throughput ($T_n$) is greater than the current bitrate by a predefined threshold ($\epsilon$), the bitrate for the next segment is increased. If the current bitrate is already at the highest ($r^{max}$), the maximum bitrate is maintained, else, the next higher bitrate is selected. Thus, the bitrate selected is increased by one-level each time the throughput is found to be higher than the current value, making the TBA algorithm an additive increase algorithm.

If the average throughput ($T_n$) is found to be close to $Rate_{prev}$, the current bitrate is maintained, else, if $T_n$ falls below the current rate then the bitrate is reduced. TBA employs an aggressive down-shift approach. This is to ensure uninterrupted playback, at the expense of video quality. If the current rate is already at the lowest bitrate ($R_{min}$), the lowest bitrate is retained. If not, it selects the highest bitrate that can be supported by $T_n$. Typically, the buffer is limited ($B_{max}$ = 60 segments) to limit the unused data in case the user decides to quit before watching the entire video.

Thus, by using the TBA algorithm, the player is able to adapt the video quality to match the throughput observed in the network.

## 5.3 Buffer-Based Adaptation

A Buffer-based rate adaptation algorithm was presented in [41]. Since the network throughput could vary significantly, the Buffer Based Adaptation (BBA) algorithm avoids using the throughput measurements and selects the next bitrates based exclusively on the current buffer occupancy. The BBA algorithm is presented in Algorithm. 2.

During the initial stages of buffering, when the information from the buffer occupancy is minimum, it does not reflect the network conditions. Hence, BBA uses a network capacity estimate to ramp up the bitrate. During the initialization phase, the bitrate is increased if the rate at which the buffer is filled is 0.875 times greater than the segment playback duration ($V_s$) i.e. $\Delta B \geq 0.875 * V_s$).

According to the BBA approach, the buffer is divided into two regions based on two threshold values: *reservoir (r)* and *cushion (cu)* where $r < cu$. As long as the current buffer occupancy ($B_{curr}$) is below the initial *reservoir (r)*, the BBA algorithm is conservative and selects the lowest bitrate ($r_{min}$). Once the buffer occupancy is more than $r$, the rate adaptation is initiated. If the current buffer occupancy ($B_{curr}$) is higher than the cushion ($r + cu$) then BBA selects the maximum bitrate.

Except for the initial stages where BBA considers the rate at which the buffer is being filled, BBA solely depends on the current level of the buffer occupancy. After the initial phase, the next bitrate is calculated using a *ratemap*. The *ratemap* is a continuous function ($f$) of buffer occupancy $B_{curr}$ and is pinned at both ends ($f(0) = R_{min}$ and $f(B_{max}) = R_{max}$) of the current buffer occupancy $f(B_{curr})$.

If the $B_{curr}$ is between $r$ and $r + cu$, the rate adaption scheme checks the value returned by the rate map function, $f(B_{curr})$. If the value is greater than bitrate one level higher than the $R_{prev}$, denoted by $R_{prev} \uparrow$, then the bitrate for the next segment will be increased by one level. Whereas, if the value is less than the bitrate one level lower than the $R_{prev}$, denoted by $R_{prev} \downarrow$, then the bitrate for the next segment will be decreased to the value that such that $r_i > f(B_{curr})$

The BBA algorithm tends to take an additive increase and aggressive decrease approach based on the buffer ratemap, while $r < B_{cur} < cu$.

**Algorithm 2:** Buffer Based Rate Adaptation Algorithm [41]

```
Data:
  $V_s$ : Segment playback Duration
  $r$ : The size of reservoir
  $cu$ : The size of the cushion
  $\Delta B$: Rate of buffer increase
Input:
  $Rate_{prev}$: The bitrate of most recent segment
  $B_{curr}$: Current buffer occupancy
  $T_n$: Average Throughput of the most recent $n$ segments
Initialization:
if $B_{curr} \leq r$ then
    if $\Delta B \geq 0.875 * V_s$ then
        $l_{n+1} = R_{prev} \uparrow$ // increase by one level
    $l_{n+1}$ = $r^{min}$
else if $B_{curr} \geq (r + cu)$ then
    $l_{n+1}$ = $r^{max}$
else if $f(B_{curr}) \geq R_{prev} \uparrow$ then
    $l_{n+1}$ = max$\{r^i | r_i < f(B_{curr})\}$
else if $f(B_{curr}) \leq R_{prev} \downarrow$ then
    $l_{n+1}$ = min$\{r^i | r_i > f(B_{curr})\}$
else
    $l_{n+1}$ = $R_{prev}$
Result:
  $l_{n+1}$: the bitrate of the next segment to be downloaded
```

# CHAPTER 6

# SEGMENT AWARE RATE ADAPTATION FOR DASH

The ABR algorithms proposed in previous literature typically use parameters like average segment download rate [68], available bandwidth [76] or buffer occupancy [48]. The client typically starts with the lowest representation, and then based on one of the parameters measured during the download, estimates the best representation for each subsequent segment. In most of these case the throughput measurements are based on the segment fetch times. Although the segments of each DASH video are of equal playback duration, due to the encoding schemes and the compression techniques used, the sizes of the segment (even with same bitrates) are found to have significant variations. Our analysis of the Big Buck Bunny video [1, 6] for three different representations (3.9 Mbps, 3.6 Mbps, and 2.9 Mbps) found that the segments vary in sizes; see Fig. 10. For the representation with a 3.9 Mbps rate, we note that the segment size varies from 538 KB to 3.9 MB with the average segment size being 1.8 MB. A similar trend was found in the segment sizes for the other bitrates.

In this chapter, we present Segment Aware Rate Adaptation (SARA), a novel DASH rate adaptation algorithm that we developed to improve the QoE for DASH videos by considering the variation in the segment sizes during the throughput measurement and rate adaptation phase. SARA was published in [56].

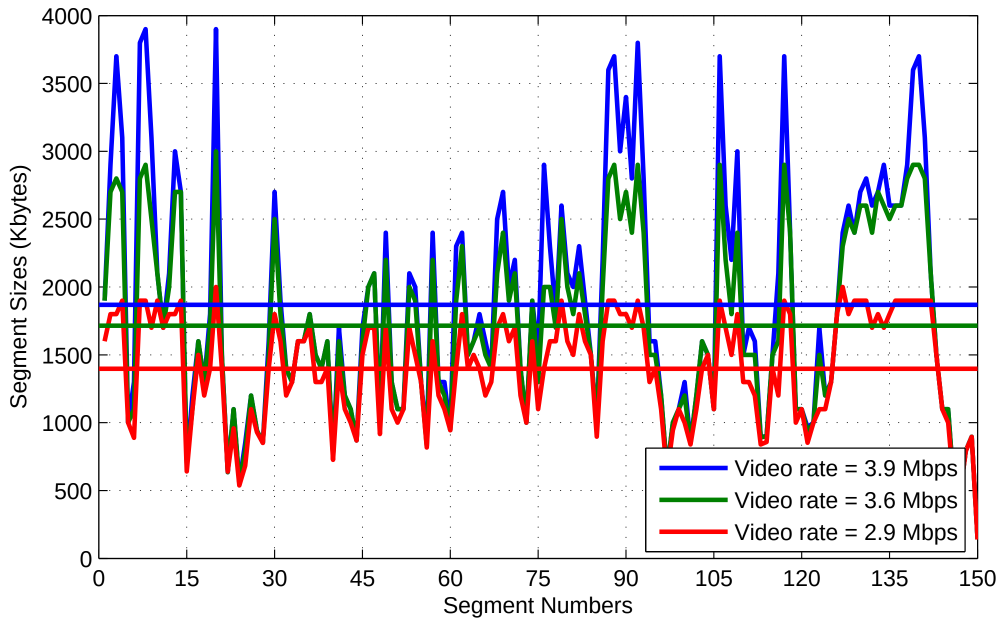


Figure 10: Variation in Segment Sizes for the Big Buck Bunny Video: 4 sec. segments

SARA uses buffer occupancy, throughput measurements, and the individual segment sizes to estimate the optimum bitrate for the next segments (Algorithm. 3). Since the segment download rates are affected by their sizes [38], SARA calculates the throughput as the Weighted Harmonic Mean (WHM), where the weights are proportional to the sizes of the segments. The WHM throughput also reduces the effect of outliers in the throughput measurements.

In this chapter, we provide a detailed description of the proposed SARA algorithm and its associated components: an enhanced MPD file, a WHM throughput estimation module and finally the SARA algorithm.

## 6.1 Enhanced MPD

Each DASH video is associated with a Media Presentation Description (MPD) is an XML file that contains information related to the video such as: the metadata related to the video (playback duration, segment dotation, playback timescale), the adaptation sets (video set, audio set, subtitles set), representations supported for each adaptation set. The video set representations could wary from each other in various aspects such as codec, video display parameters (height, width, frame-rate), recommended bandwidth. The individual URLs for each segment of every representation set is also listed in the MPD file.

We propose to enhance the standard MPD file by listing the individual segment sizes along with the URLs. Using these segment sizes the client could make an informed decision during the rate adaptation phase. In our evaluation setup, we added the segment sizes to the standard MPD file during the pre-processing phase at the DASH server. These sizes are later used in the throughput estimation (Section 6.2) and SARA algorithm (Section 6.3).

## 6.2 Throughput Estimation with Weighted Harmonic Mean

In order for the rate adaptation algorithm to pick the appropriate bitrate, most ABR algorithms estimate the throughput based on the download rates of the individual segments. When the average segment throughput is used to estimate the overall network throughput, the measurements could be affected by the instantaneous variations since average value tends to favor the outliers. The effect of instantaneous variations in the measured throughput can be avoided by using harmonic mean and was used in [48]. However, the segments being downloaded for a DASH streaming session are of different sizes, and since the average download rate over HTTP depends on the file size distribution [38]. In order to accurately predict the download rate for the next segment, we assigned weights ($w_1$, $w_2..w_n$) that were proportional to the segment sizes. These weights and the respective segment download rates ($d_1$, $d_2..d_n$) were used to calculate the Weighted Harmonic Mean (WHM) download rate. The weighted harmonic mean download rate for $n$ downloaded segments is given by

$$H_n = \frac{\sum_{i=1}^{n} w_i}{\sum_{i=1}^{n} \frac{w_i}{d_i}} \tag{6.1}$$

Based on the $H_n$ computed for last $n$ segments, the time to download the next segment is predicted by $w_{n+1}/H_n$. By using WHM download rate, we are able to estimate the download rate of the segments irrespective of the variation in the segment sizes.

## 6.3 Segment Aware Rate Adaptation Algorithm (SARA)

Our algorithm, SARA selects the most suitable representation for the next segment to be downloaded from the set of available representations ($\mathbb{R}$). Prior to starting the video playback the player downloads the enhanced MPD file from the video server. The MPD file is parsed at the client-end to retrieve the video representations and the suggested bandwidths for each representation. The list of bandwidths are stored as a set of bitrates,

$r = \{r^{min}, \ldots, r^{m}, \ldots, r^{max}\}$. Also the list of individual segment URLs and their respective sizes are also parsed and stored.

Each segment is downloaded and placed in a buffer of maximum size $B_{max}$. The buffer is associated with three thresholds: $I$, $B_\alpha$, and $B_\beta$ as illustrated in Fig. 11. These thresholds are defined in terms of the number of segments. Based on the buffer occupancy at any given time, $B_{curr}$, the rate adaptation goes through the following four stages:

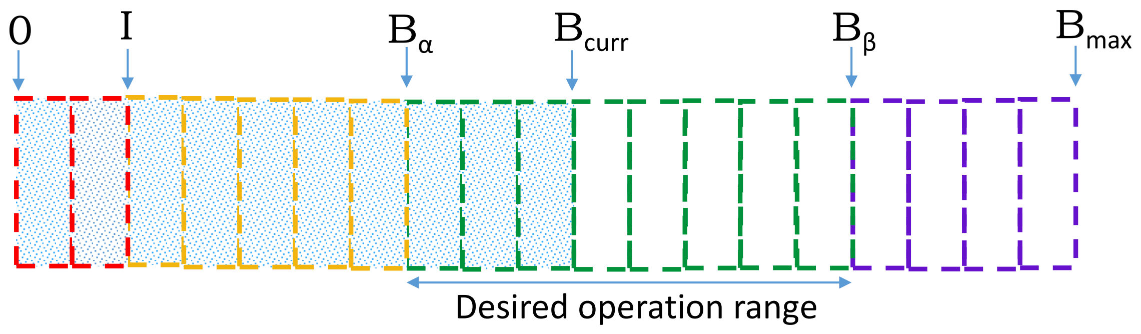


Figure 11: SARA Streaming Buffer Thresholds

1. **Fast Start** ($B_{curr} \leq I$): When the buffer occupancy is below $I$, the lowest bitrate is selected. This ensures that the playback start time is kept as low as possible. It was observed that minimizing playback start time is important to prevent the user from abandoning the video session [59].

2. **Additive increase** ($I < B_{curr} \leq B_\alpha$): Once the buffer occupancy goes beyond $I$, the algorithm enters the additive increase stage. In this stage, the bitrate is incremented in small (single) steps to avoid the buffer going back below $I$ and thus, taking a conservative approach. The video quality is increased only if the amount

of time required to download the next segment of the bitrate one more than the current level ($r^{curr} \uparrow$) is less than the video left in the safe region of the buffer ($w_{n+1}^{curr+1}/H_n < B_{curr} - I$). By using an additive increase approach, we ensure that the video quality not increased to a very high value resulting from over-estimating the network capacity.

3. **Aggressive switching** ($B_\alpha < B_{curr} \leq B_\beta$): The region between $B_\alpha$ and $B_\beta$ is the most preferred buffer occupancy. In this stage, based on the current network bandwidth and the buffer occupancy, the most suitable bitrate that is greater than or equal to the current bitrate is selected. During this stage, SARA determines the maximum bitrate that can be downloaded before the video buffer goes lower than the safe region ($B_{curr} - I$). In order to determine this, it considers the next-segment size for every representation. However, if it determines that the download time required for the next-segment of the current bitrate is higher than $B_{curr} - I$, then it switches to a lower bitrate value that is most suitable.

4. **Delayed Download** ($B_\beta < B_{curr} \leq B_{max}$): When the buffer occupancy increases beyond $B_\beta$, the most suitable bitrate for the current network bandwidth is selected; however, the request for the segment is sent only when the buffer occupancy falls to $B_\beta$. The delay is denoted by $\delta$ and is given by $B_{curr}$ - $B_\beta$. The delayed download limits the total number of segments in the video buffer, thus avoiding unnecessary downloads in case the user prematurely quits watching the video.

The rate adaptation algorithm, SARA, is presented in Algorithm 3. SARA is

**Algorithm 3:** Segment Aware Rate Adaptation Algorithm

```
Data:
    $\mathbb{R}$ : Set of available bitrates $\{r^{min}, ..., r^{i}, ..., r^{max}\}$
    $I$, $B_{\alpha}$, $B_{\beta}$, $B_{max}$: Buffer constants (number of segments)
Input:
    $n$: Segment number of the most recent download
    $r^{curr}$: Bitrate of the most recently downloaded segment
    $B_{curr}$: Current buffer occupancy in seconds
    $r$ = $\{r^{min}, .., r^{m}, .., r^{max}\}$ Available bitrates
    $W_{n+1}$ = $\{w^{min}_{n+1}, ..w^{i}_{n+1}, ..w^{max}_{n+1},\}$ sizes of the $(n+1)^{th}$ segment
    $H_n$: Weighted Harmonic mean download rate for the first n segments
Initialization:
if $B_{curr} \leq I$ ;                                    // Fast Start
then
    $l_{n+1} = r^{min}$;
else
    if $(\frac{w^{curr}_{n+1}}{H_n}) > B_{curr} - I$ then
        $l_{n+1}$ = max$\{r^{i} | r^{i} \in \mathbb{R}, \frac{w^{i}_{n+1}}{H_n} \leq B_{curr} - I, i \leq curr\}$;
        $\delta$ = 0;
    else if $B_{curr} \leq B_{\alpha}$ ;                  // Additive Increase
    then
        if $\frac{w^{curr+1}_{n+1}}{H_n} < B_{curr} - I$ then
            $l_{n+1} = r^{curr} \uparrow$ ;               // increase by one level
        else
            $l_{n+1} = r^{curr}$;
        $\delta$ = 0;
    else if $B_{curr} \leq B_{\beta}$ // Aggressive Switching
    then
        $l_{n+1}$ = max$\{r^{i} | r^{i} \in \mathbb{R}, \frac{w^{i}_{n+1}}{H_n} \leq B_{curr} - I, i \geq curr\}$;
        $\delta$ = 0;
    else if $B_{curr} > B_{\beta}$ // Delayed Download
    then
        $l_{n+1}$ = max$\{r^{i} | r^{i} \in \mathbb{R}, \frac{w^{i}_{n+1}}{H_n} \leq B_{curr} - B_{\alpha}, i \geq curr\}$;
        $\delta$ = $B_{curr}$ - $B_{\beta}$
    else
        $l_{n+1}$ = $r^{curr}$;
        $\delta$ = 0
Result:
$l_{n+1}$: the bitrate of the next segment to be downloaded
    $\delta$: The wait time before downloading the next segment
```

invoked after downloading each segment to determine the bitrate to be selected for the next segment. The algorithm is initialized with the list of the available video bitrates (representations), $\mathbb{R}$ and the thresholds for the buffer $I$ (default=2), $B_\alpha$, $B_\beta$, $B_{\max}$. These parameters remain constant during the entire playback.

SARA is invoked with the following inputs: the segment number of the most recent segment that was downloaded, the current bitrate ($r^{curr}$; default is $r^{min}$), current buffer occupancy ($B_{curr}$), the sizes of $(n+1)^{th}$ segment across all the available representations $(w_{n+1}^{min}, \dots w_{n+1}^{m}, \dots w_{n+1}^{max})$, and the current weighted harmonic mean download rate.

At the start of the video session (or after an interruption due to buffer starvation), the number of segments in the buffer is below $I$ and the playback begins in the *Fast Start* phase, during which the lowest bitrate is selected. Also, the delay $\delta$ is set to zero so that the next segment is downloaded immediately. Once the buffer occupancy increase beyond $I$, the adaptation enters the *Additive Increase* stage.

The time required to download the next segment that corresponds to the current bitrate is given by $(w_{n+1}^{curr}/H_n)$. At any time during the playback, it is not feasible to download the next segment of the current bitrate before the buffer goes below $I$, i.e., if the time taken to download the next segment is greater than $B_{curr} - I$; the highest possible bitrate that can be downloaded in the duration $B_{curr} - I$ is selected based on the next segment size and the current weighted harmonic mean download rate ($H_n$). SARA tries to maintain the buffer occupancy above $I$; else, it switches to the lowest bitrate ($r^{min}$).

In the *Additive Increase* stage, the algorithm starts the adaptation process during which the video bitrate is gradually increased. The increase in the quality is done in single steps; $r^{curr}\uparrow$ indicates the next level of the bitrate. By using single step increments we ensure gradual quality changes, thus, avoiding the unpleasant effect of rapid bitrate switches. However, the step size is a configurable parameter that can be based on the desired aggressiveness. When the video session is in the additive increase stage, i.e., the current buffer occupancy ($B_{curr}$) is below $B_\alpha$. If the amount of time needed to download the next segment of the bitrate, which is one more than the current bitrate i.e., $r^{curr}\uparrow$ is less than $B_{curr} - I$, then the next level is chosen. Otherwise, the current bitrate is maintained. Use of additive increase increases the number of bitrate switching events compared to aggressive switching however, since these switches are of positive polarity, they are not perceived negatively by the users [26].

When $B_{curr} < B_\alpha$, SARA takes a conservative approach of giving a higher priority to minimize the start time and interruptions over the video quality. Once $B_{curr}$ goes beyond $B_\alpha$, it starts to be more optimistic. While $B_\alpha < B_{curr} < B_\beta$ the highest bitrate whose segment could be downloaded before the current buffer goes below $I$ ($w^i_{n+1}/H_n \leq B_{curr} - I$) is selected. By utilizing the segment sizes in the decision making process, we ensure that even when the segment sizes fluctuate, the download time of the segment can be predicted with better accuracy. In the *aggressive switching* stage, the next segment request is sent immediately. The algorithm selects the highest possible video rate, while making sure that the buffer is maintained at a steady level.

When the network conditions are favorable and the buffer occupancy increases beyond $B_\beta$, the algorithm selects the best possible bitrate similar to the aggressive switching stage, but instead of sending the request for the next segment immediately, it waits until the buffer occupancy falls to $B_\beta$. This limits an unnecessary data fetch in case the user decides to prematurely quit watching the video. The algorithm returns the bitrate for the next segment and the waiting time to issue the request for the next segment.

The main objective of SARA is to be able to provide consistent QoE to the users irrespective of the segment size variations, and also to provide lower negative bitrate switching events by better estimation of the segment download times.

## 6.4 Preliminary Evaluation of SARA

As a preliminary evaluation, we studied the performance of SARA under limited bandwidth in comparison with a basic rate adaptation algorithm. The basic rate adaptation algorithm starts by downloading the segment with the lowest bitrate. While downloading the first two segments (same as $I$ in SARA) with the lowest bitrate, the average download rate of the segments is measured. When the buffer occupancy is greater than two segments, the bitrate is increased from the lowest bitrate. Based on the measured average throughput and the current buffer occupancy, it selects the bitrate for the next segment, which is one less than the the bitrate closest to the available bandwidth. The buffer size for the basic adaptation was set to a maximum of 10 segments (40 seconds) and uses an optimistic switch-up and switch-down bitrate switching.

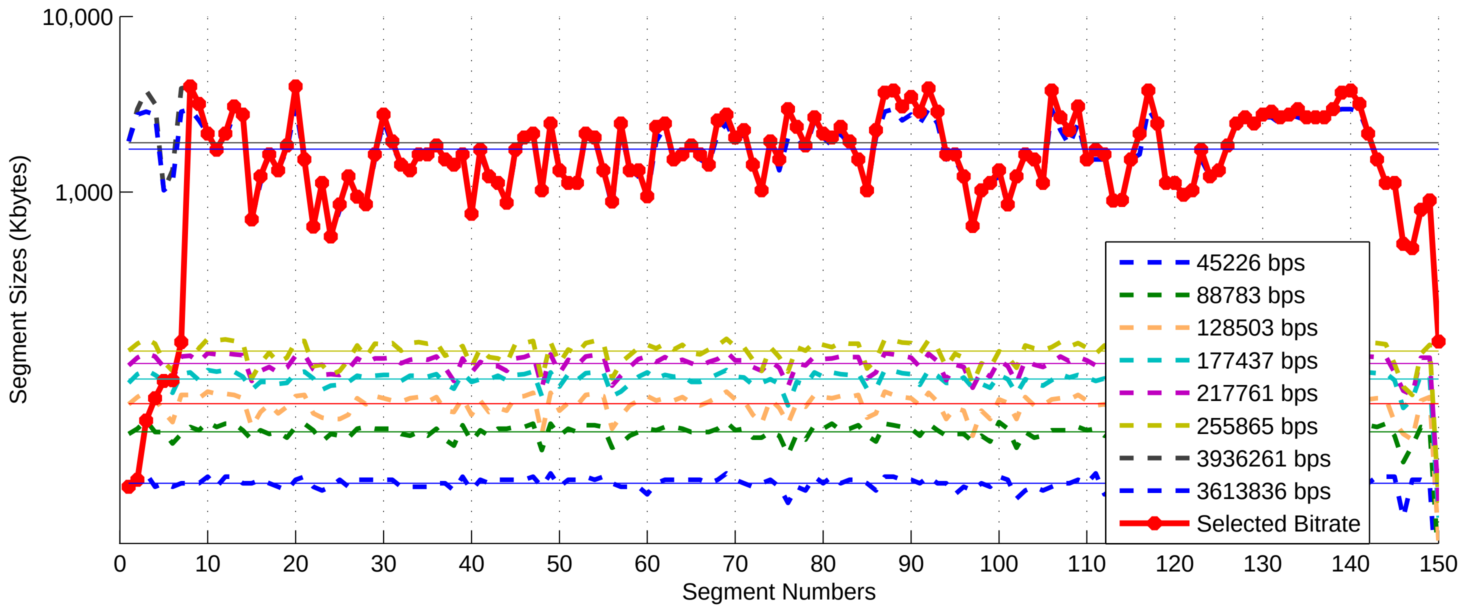


Figure 12: Bitrates Selected by SARA with Fixed Bandwidth = 1 Mbps

For the preliminary evaluation we used a single client and server setup over the *Global Environment for Networking Innovation* (GENI) [21] testbed. We used the popular open-source video *Big Buck Bunny*. The open source version of this video is available in 20 different representations. The total length of the video is around 596 seconds, and each representation is split into 150 segments, each with a 4 second playback duration.

We first demonstrate the rate-adaptation mechanism of SARA when the bandwidth between the server and the client was fixed at 1 Mbps as shown in Fig. 12. The variation in segment sizes for different bitrates available for the Big Buck Bunny video are plotted. Since SARA uses aggressive up switching some of the bitrate levels were skipped. In the current figure we only plot the variation in segment sizes for bitrates that were played during playback. The figure also shows the actual bitrates selected by SARA algorithm for each of the segments. As we can see, SARA initially starts by playing the lowest bitrates as gradually increases to the highest bitrates.

We then studied the effect of the variation in the segment sizes on the two adaptation schemes. We did this by limiting the network bandwidth between the client and the server to 1 Mbps. The buffer size of the basic adaptation algorithm was configured to be 10 segments and SARA was configured with $B_\alpha = 5$, $B_\beta = 10$, and $B_{\max} = 12$. The video was played using both ABR algorithms, the experiment being repeated 10 times. The results reported are based on the statistical mode of the bitrate values measured, since video qualities are at discrete values. In Fig. 13, we show the variation in the video bitrate selected by each ABR along with the segment sizes (in KB). Even with a steady bandwidth we noticed that the variation in the segment sizes affected the segment download

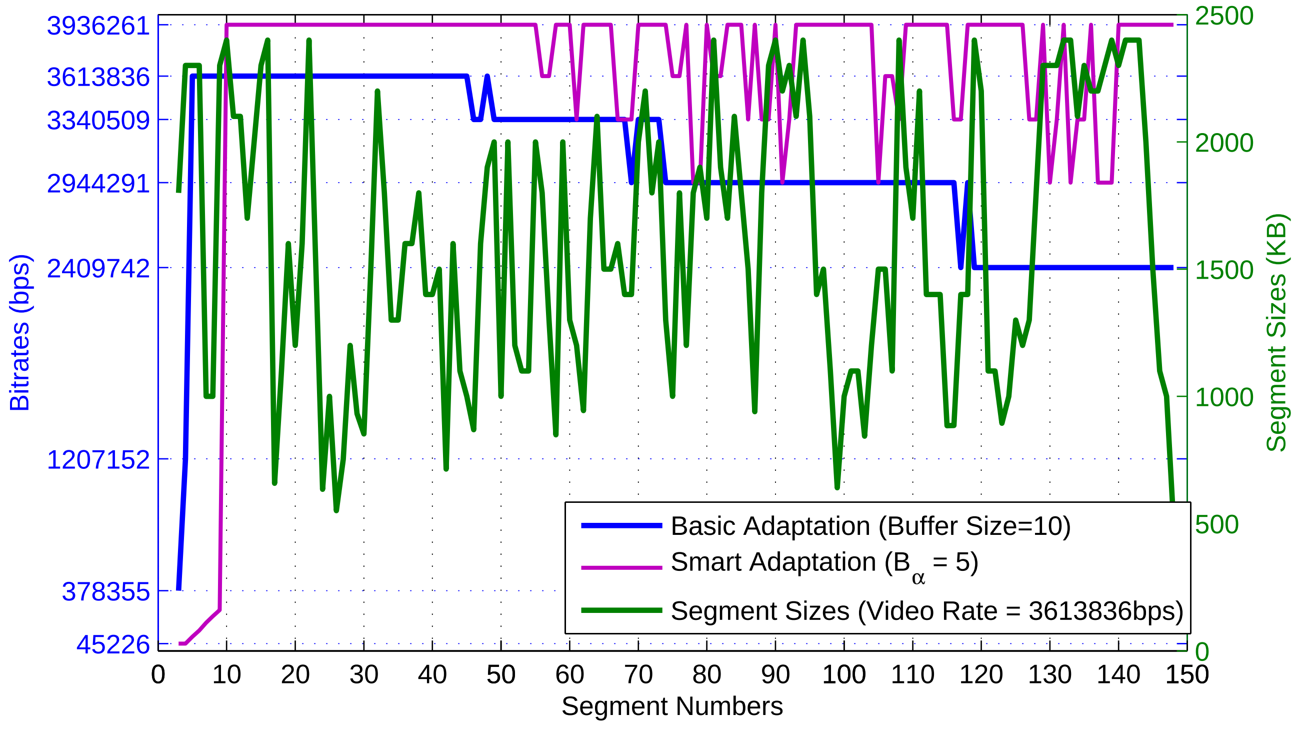


Figure 13: Effect of Segment Sizes on Basic and SARA (Bandwidth = 1 Mbps)

times. Although the sudden changes in the segment sizes affected the bitrate selected by both the schemes, the basic algorithm failed to anticipate the decrease in the segment sizes, forcing it to pick a lower bitrate. However, since SARA had the segment information it was able to predict the segment download times and thus, maintained higher bitrates for longer durations.

Next, we studied the bitrate adaptation for both the algorithms with the network bandwidth fixed at 4 Mbps and 8 Mbps. The results are shown in Fig. 14 and Fig. 15, respectively. In both the cases, we observed that due to the *Additive Increase* stage of SARA, it takes a longer time to reach the highest bitrate compared to the basic ABR algorithm. Particularly, with $B_\alpha = 10$, it takes about 20 seconds to reach the highest bitrate. However, once SARA reaches the highest bitrate, it sustains the higher bitrate for longer durations. Even with $B_\alpha = 5$, we see that the bitrate picked by SARA was always better than (or at least as good as) the basic adaptation. With $B_\alpha = 10$, SARA converges to the highest bitrate steadily, but at a slower rate than the basic scheme. Subsequently, there are no bitrate switches when the segment sizes vary. Having a larger buffer gives a cushion to SARA and avoids responding to sudden changes in the segment download rates. The use of a buffer size of 10 segments translates to a video playback of 40 seconds, which seems reasonable considering that a user may watch a movie that may last 30 minutes or longer as is the case with most videos on Netflix.

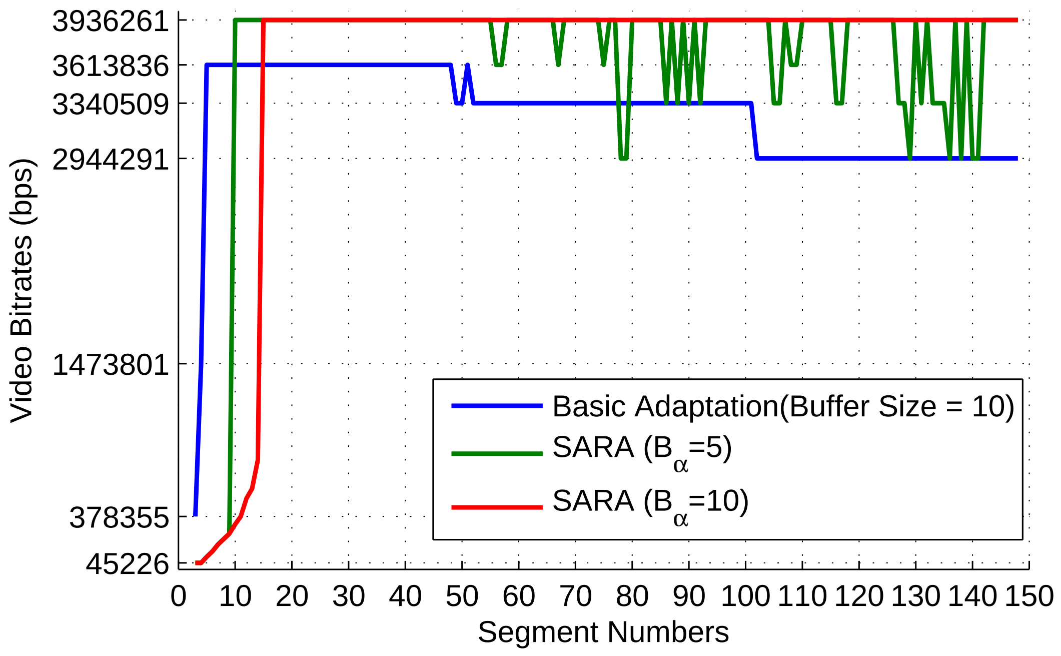


Figure 14: Bitrate Variation when Bandwidth = 4 Mbps

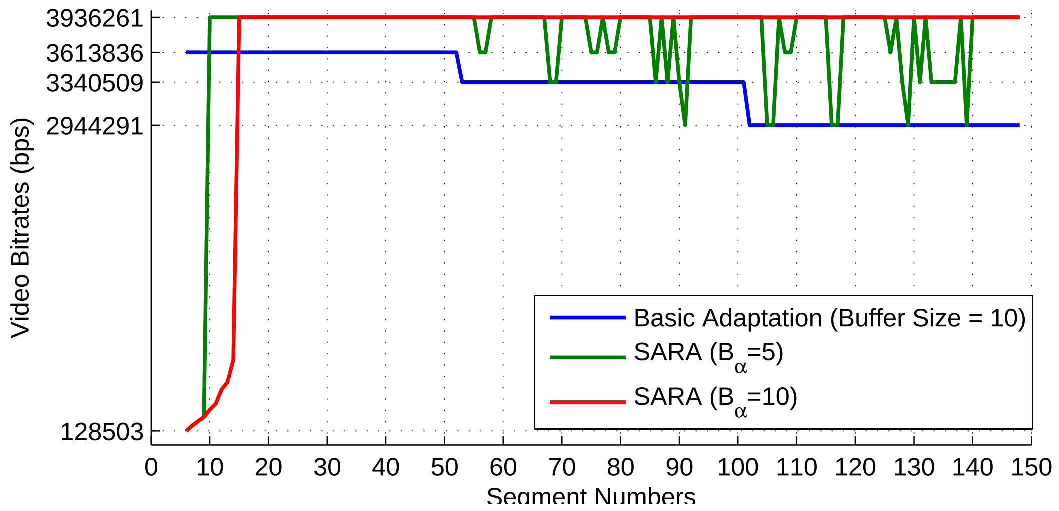


Figure 15: Bitrate Variation when Bandwidth = 8 Mbps

Based on our initial evaluations we find that $B_\alpha = 10$ provides better QoE in terms of bitrate switching events. Hence, for the extensive evaluation and comparative analysis for SARA presented in Chapter 7, we use $B_\alpha = 10$.

# CHAPTER 7

# PERFORMANCE EVALUATION OF SARA

In this section we present the results from an evaluation of QoE with SARA on the GENI testbed under varying network conditions and compare its performance with the TBA and BBA algorithms.

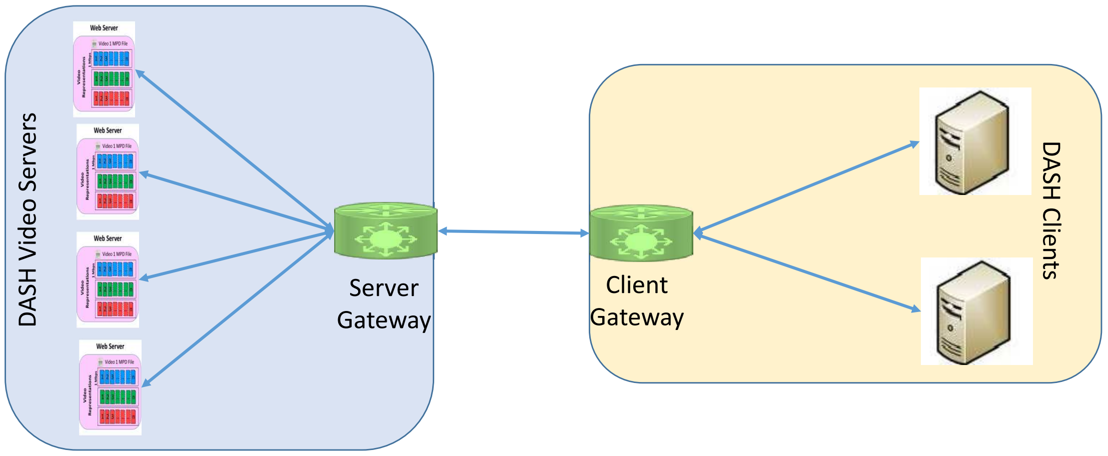


Figure 16: Experiment Layout

## 7.1 Evaluation Platform

The ABR algorithms were evaluated in a controlled globally distributed networked virtual testbed, *Global Environment for Networking Innovation* (GENI) [21]. With GENI we were able to create a virtual network topology that is geographically distributed. We created a topology consisting of a server cluster and multiple clients connected through by gateways on either end as depicted in Fig. 16. At the server end, a cluster of four

Table 6: DASH Video Datasets

| Title | Number of Bitrates | Genre |
|---|---|---|
| Big Buck Bunny | 20 | Animation |
| Of Forest and Men | 19 | Documentary |
| The Swiss Account | 17 | Sport |
| Valkaama | 20 | Movie |

HTTP web servers, each running Apache2 was used. The clients ran on Ubuntu 12.04. The server cluster and clients were connected by gateway nodes on each side. We used *Traffic Control (tc)* [61] on the gateway nodes to modify the network conditions between the clients and the server clusters to emulate a real network . *Traffic Control* is an in-built Linux module to configure the available bandwidth between the servers and the clients.

The video players used by the commercial DASH video services are closed-source and proprietary, hence difficult to be modified. In order to evaluate SARA, we developed our own open-source emulated DASH player, *AStream*, the code for which can be accessed from GitHub [49]. *AStream* is an open-source Python based player that has a buffer module that emulates a video playback buffer. Data from the buffer is played one-second at a time, and the data can be to the buffer written until it is full. The DASH client module in *AStream* starts by downloading and parsing the MPD file from the server. It can be configured to select any one of the three ABR algorithms: SARA, BBA or TBA. The different buffer parameters for each algorithm can be configured to study the performance of the algorithm under different conditions and settings. Each of the clients in our experiments were configured to run the emulated DASH player *AStream*.

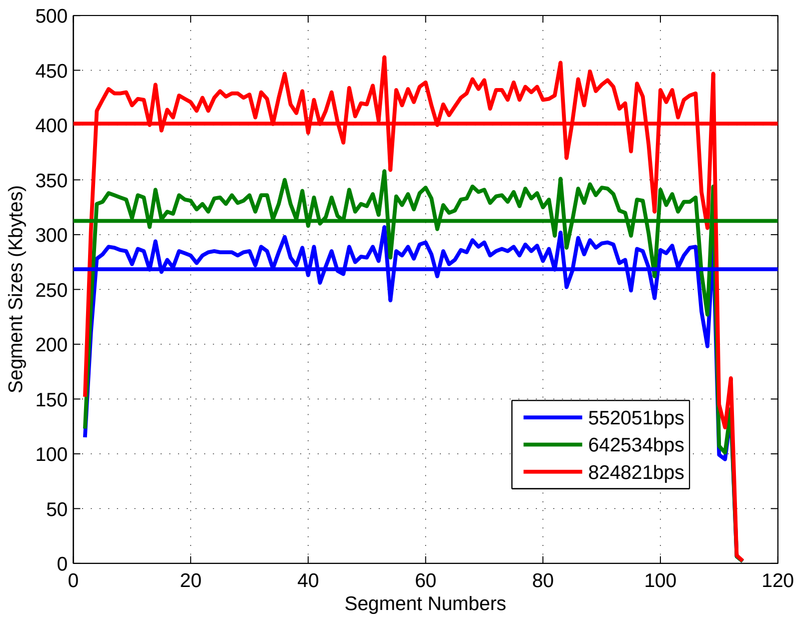


Figure 17: Segment Size Variations for the Video: Of Forest Men

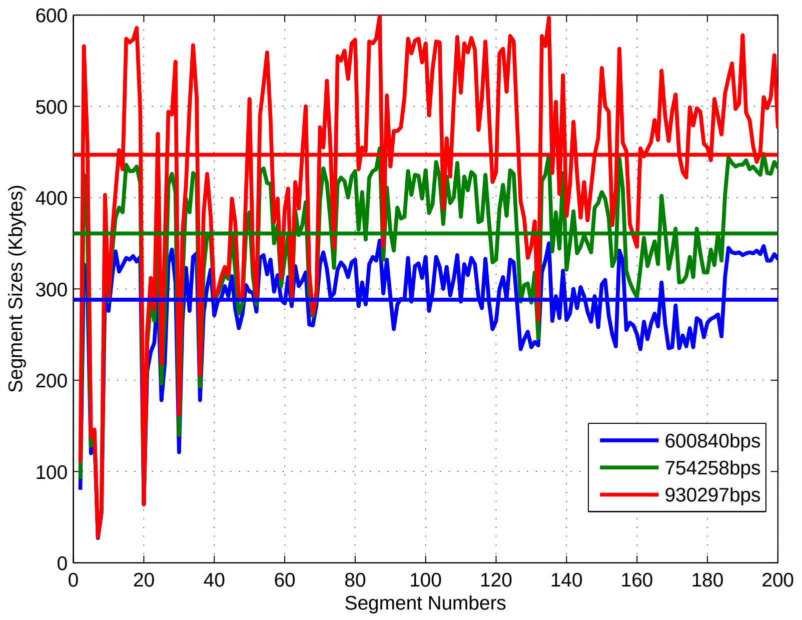


Figure 18: Segment Size Variations for the Video: The Swiss Account

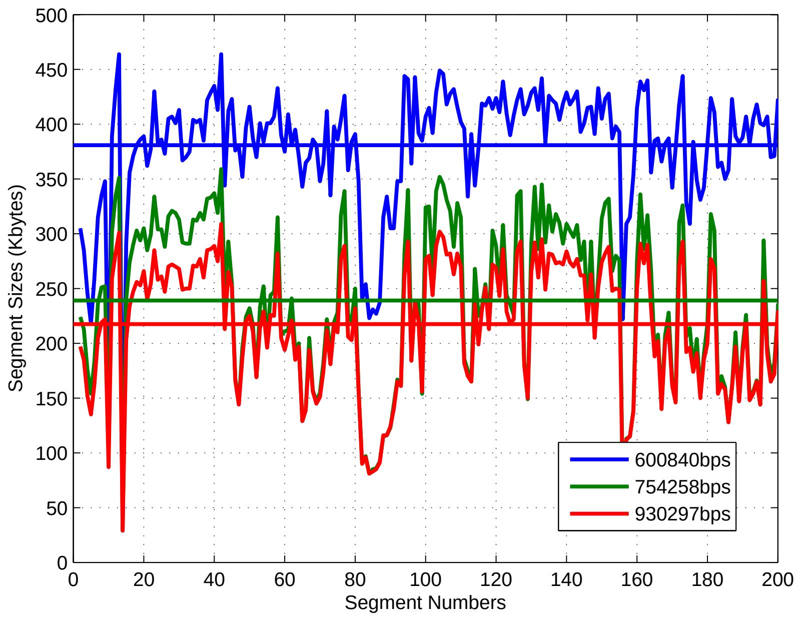


Figure 19: Segment Size Variations for the Video: Valkaama

In the current evaluation, we considered four different videos, each from a different genre: animation, documentary, sport, and movie (See Table 6) obtained from [6]. The available video bitrates for each video were between 17 and 20. Each segment was of 4 seconds fixed playback duration. We limited the video session durations to a maximum playback duration of 10 minutes. The variation in the segment sizes for the Big Bunny Bunny was presented in Fig. 10 and the remaining three videos are presented in Fig. 17, Fig. 18 and Fig. 19. Although each of these video have 17 to 20 representations, only three representations are plotted for clarity and a similar pattern was found for the other representations.

While DASH ensures that there are no interruptions in playback by switching the bitrate, frequent bitrate switching is shown to degrade the QoE of users [64]. On the other hand, a higher quality of the playback improves the QoE of the users [102]. An ABR algorithm starts by playing the lowest bitrate and ramps up towards higher bitrates. The time taken to reach the highest possible bitrate for any network bandwidth is called the *convergence time*. To study the QoE management capability of all the three ABR algorithms we used three different QoE metrics to evalaute their performance. In addition to the two objective metrics number of bitrate switching events and convergence time discussed in Section 2.1.1 we also introduce a new quality factor, $Q_{ratio}$. The relative video quality measure $Q_{ratio}$ is the ratio of the total quality of the video played when compared to basic TBA algorithm. Let $f_{TBA}(x)$, $f_{BBA}(x)$ and $f_{SARA}(x)$ denote the bitrate selected for a segment $x$ by the TBA, BBA, and SARA algorithms respectively. The $Q_{ratio}$ for BBA and SARA in comparison to the TBA algorithm is given by

$$Q_{ratio(BBA)} = \frac{\int f_{BBA}(x)\,dx}{\int f_{TBA}(x)\,dx} \tag{7.1}$$

$$Q_{ratio(SARA)} = \frac{\int f_{SARA}(x)\,dx}{\int f_{TBA}(x)\,dx} \tag{7.2}$$

## 7.2 Bandwidth Variation Scenarios

We evaluate the QoE metrics under varying bandwidth environments and different type of network interruptions. Four different bandwidth scenarios were enforced by configuring the *Traffic Control* module on the server gateway node and the scenarios were as follows

1. **Fixed Bandwidth:** In this case, we limit the maximum bandwidth between the

client and the servers to a fixed bandwidth of either 0.5, 1, 4, or 6 Mbps. These values were selected to cover the low bandwidth network environments; particularly targeted in the design of SARA algorithm. With fixed bandwidth, we aim to evaluate the convergence time of the ABR algorithm under steady but limited network environments.

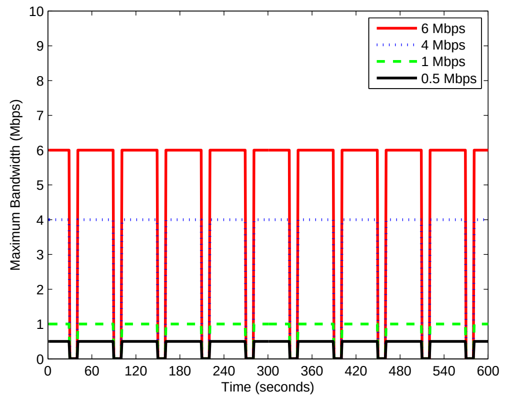
2. **Short Interruptions:** For the various bandwidths listed above, we considered multiple short interruptions in the network data transfer. For each of the bandwidth scenarios (1, 4, or 6 Mbps), we introduced 10 second interruptions (bandwidth falling to 0.005Mbps) for every minute of segment download as shown in Fig. 20(a).

3. **Long Interruptions:** For long interruptions, we considered 3 counts of 1 minute interruptions (bandwidth falling to 0.005Mbps), for all four bandwidths as shown in Fig. 20(b).

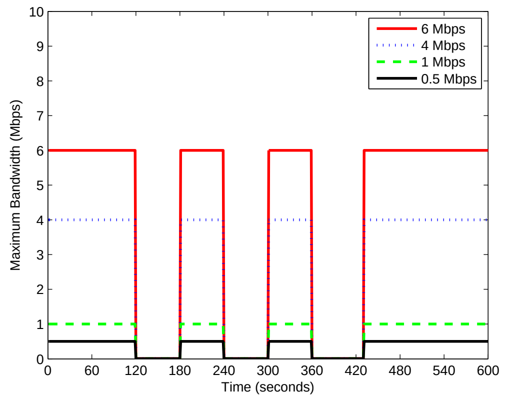
4. **Truncated Gaussian:** When the bandwidth was around a few Mbps, it was observed to follow a Gaussian distribution [47]; hence, to emulate a public network with low bandwidth, we used a truncated Gaussian with a mean bandwidth of 3Mbps as depicted in Fig. 20(c).

## 7.3 Evaluation of QoE

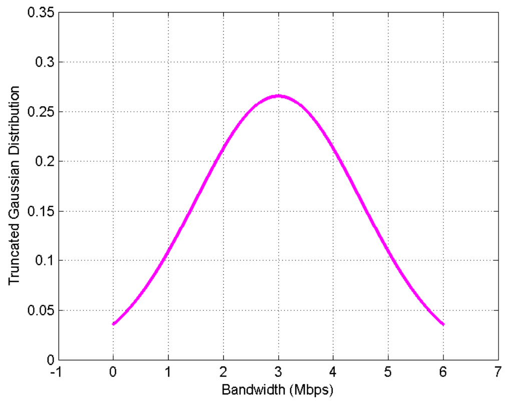
For every scenario listed in Section 7.2, we evaluated the performance of the three ABR algorithms: TBA, BBA, and SARA. For each of the first three scenarios, we also changed the maximum bandwidth to 0.5, 1, 4 and 6 Mbps, replicated 5 times, for each of



(a) Short Interruptions



(b) Long Interruptions



(c) Truncated Gaussian

Figure 20: Bandwidth Scenarios

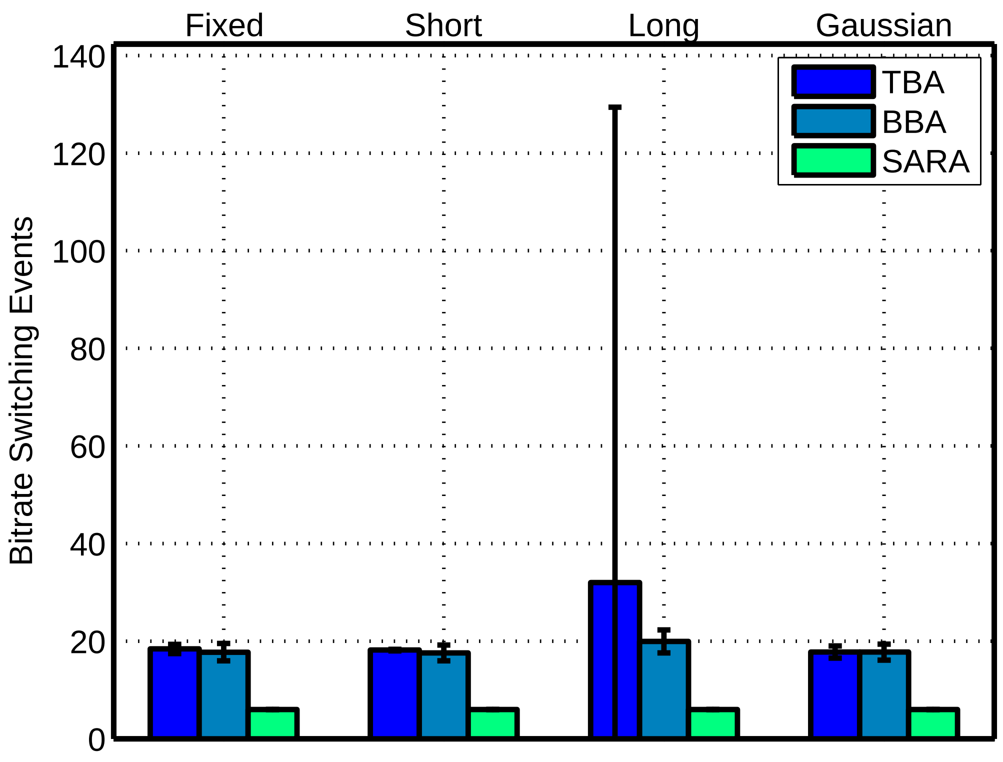


Figure 21: Bitrate Switching Events: Mean Bandwidth = 4Mbps

the four videos. For the Gaussian case, we used 10 trials for each video and algorithm combination. We discuss the QoE at the client-end based on different QoE metrics in the following sections. The errors bars shown in the bar plots discussed in the following sections indicate variance.

### 7.3.1 Bitrate Switching Events

In Section 2.1.1, we discussed the bitrate switching events with DASH. For improved QoE with DASH streaming it is necessary to minimize the number of bitrate

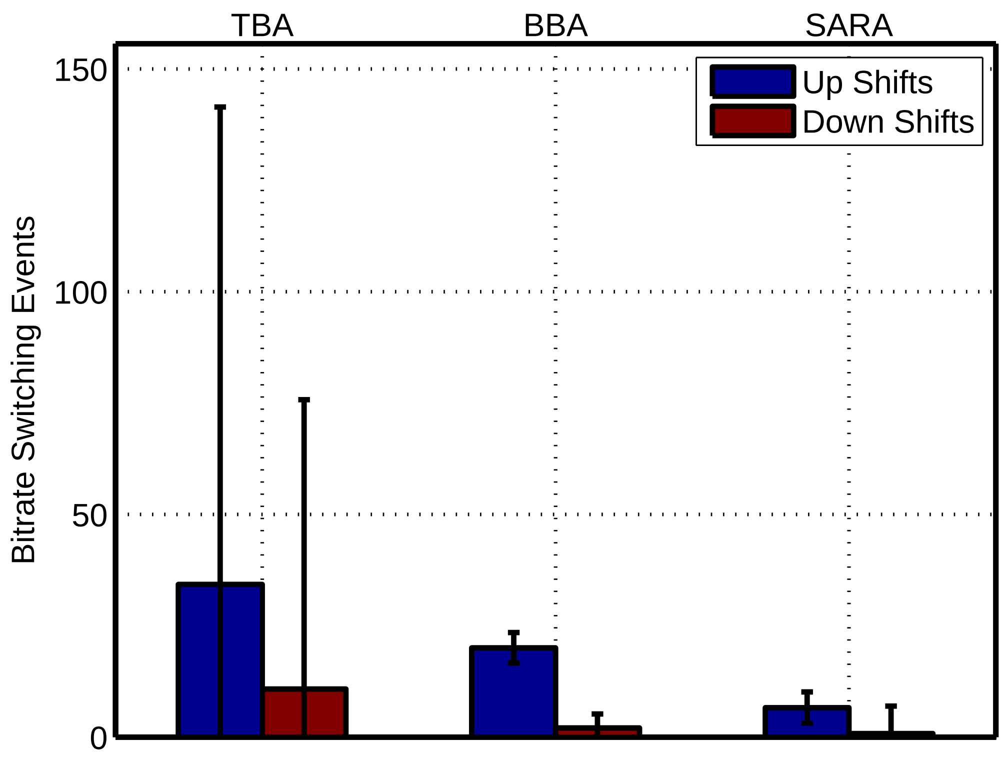


Figure 22: Polarity of Bitrate Switches with Long Interruptions: Mean Bandwidth = 1Mbps

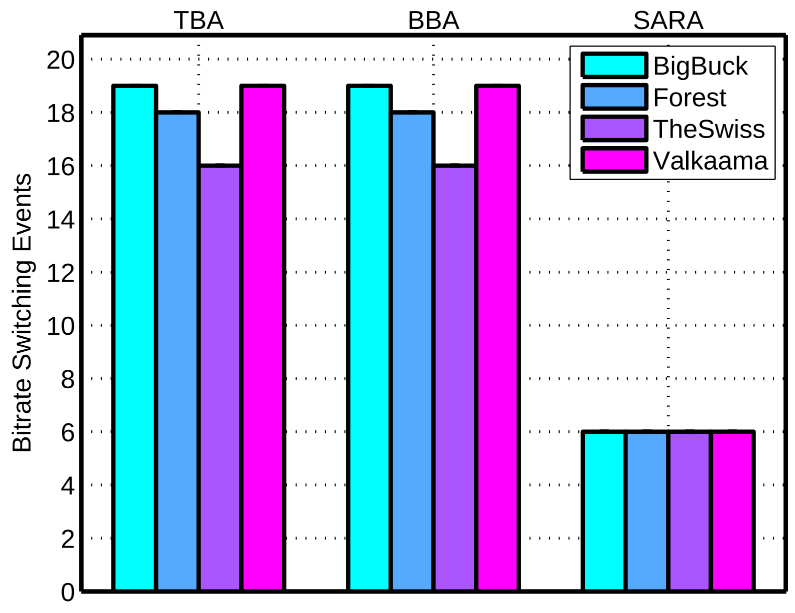
switching events. We evaluated the number of bitrate switching events and their polarities observed for each of the adaptation algorithms under varying network conditions.

In Fig. 21, we plotted the number of bitrate switching events observed for all the videos under the different bandwidth scenarios when the maximum bitrate was set to 4Mbps. The error bars indicate the variance of the number of bitrate switching events for the respective scenarios. For the fixed and short interruptions case, we found that TBA and BBA faced a similar number of bitrate switching events whereas SARA experienced 30%

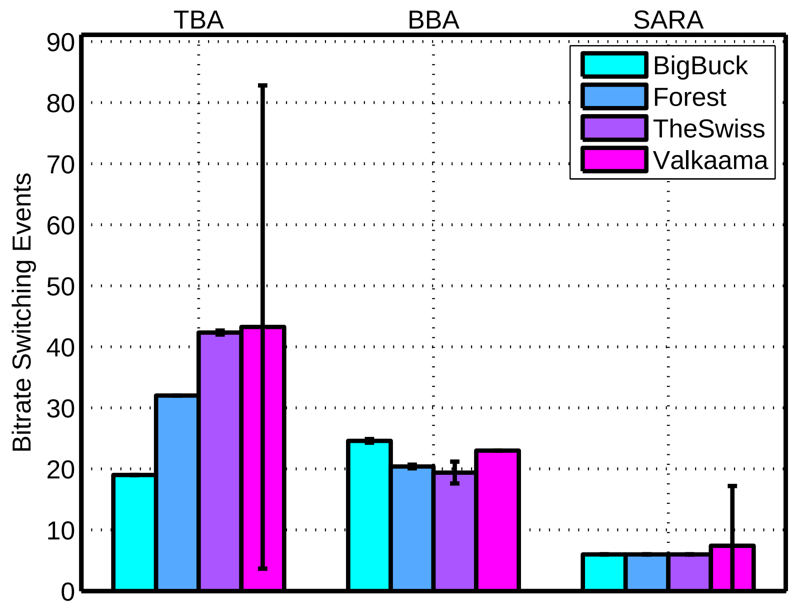
less bitrate switching events. In case of long interruptions, TBA performed significantly worse with a very high variance. This high variance was observed fo the Valkaama video as seen from Fig. 23(b) and this is caused by the high fluctuations in segment sizes similar to the changes around the segment numbers 50 and 82 as shown in Fig. 19. From this result, we can also see that SARA not only reduces the number of bitrate switching events but also provides consistent bitrate switching irrespective of the bandwidth conditions.

Different polarity of the bitrate switching events is found to affect the users differently. Users tend to be more tolerant towards positive switching events as compared to negative switching events [26]. Users also tend to perceive positive switching events as a favorable event. In Fig. 22, we present the polarity of the bitrate switches in the case of long interruptions, where the maximum bandwidth was limited to 1 Mbps. Since the ABR algorithms are additive increase and aggressive switch-down, we observed that most of the switching events are upward switches. With SARA, we not only observed a reduction in positive switches, we find the negative switches to be significantly lower. The variance in the switching events with TBA is very high due to the high switching events observed with one particular video (Valkaama). Since SARA explicitly considers the segment size variations, it is able to better estimate the variation in the segment fetch times and hence, sustain a consistent QoE, irrespective of the video specific segment size distributions.

When we considered the QoE with individual videos for short and long interruptions (Fig. 23), we found that the video selection affects the QoE with TBA and BBA algorithms. This can be attributed to the lack of explicit consideration of the segment size



(a) Short Interruptions



(b) Long Interruptions

Figure 23: Bitrate Switching Events: Mean bandwidth = 0.5Mbps

variations. In Fig. 10, Fig. 17, Fig. 18 and Fig. 19 it was shown that even though the segments are of same playback duration, the sizes for the videos can vary significantly. These variations, if not considered, can affect the ABR algorithm's perception of the network bandwidth, thus affecting the QoE management. Since, SARA considers these variations, the QoE management with SARA is more consistent across all the video.

### 7.3.2 Video Quality

In Table 7 we present the video quality measure for BBA and SARA in comparison with TBA for all the scenarios. We see that overall, SARA performs 4%-5% better than TBA for fixed and short interruption scenarios, whereas BBA is relatively worse. Video playback with SARA experienced significantly higher quality (18%) when the network experiences long interruptions, which is significantly better than TBA or BBA. Furthermore, SARA was able to sustain this improved quality even in the low bandwidth cases. We find that better estimation of segment fetch times is critical in improving the video quality in low bandwidth environments.

Table 7: Video Quality Measurement of BBA and SARA in Comparison with TBA

| **Bandwidth Scenarios** | **Video Quality Measure ($Q_{ratio}$)** | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | **0.5Mbps** | | **1Mbps** | | **4Mbps** | | **6Mbps** | |
| | BBA | SARA | BBA | SARA | BBA | SARA | BBA | SARA |
| Fixed | 0.84 | 1.04 | 0.83 | 1.04 | 0.83 | 1.05 | 0.83 | 1.04 |
| Short Interruptions | 0.82 | 1.03 | 0.84 | 1.04 | 0.84 | 1.05 | 0.84 | 1.04 |
| Long Interruptions | 0.93 | 1.19 | 0.97 | 1.17 | 0.91 | 1.13 | 0.98 | 1.18 |

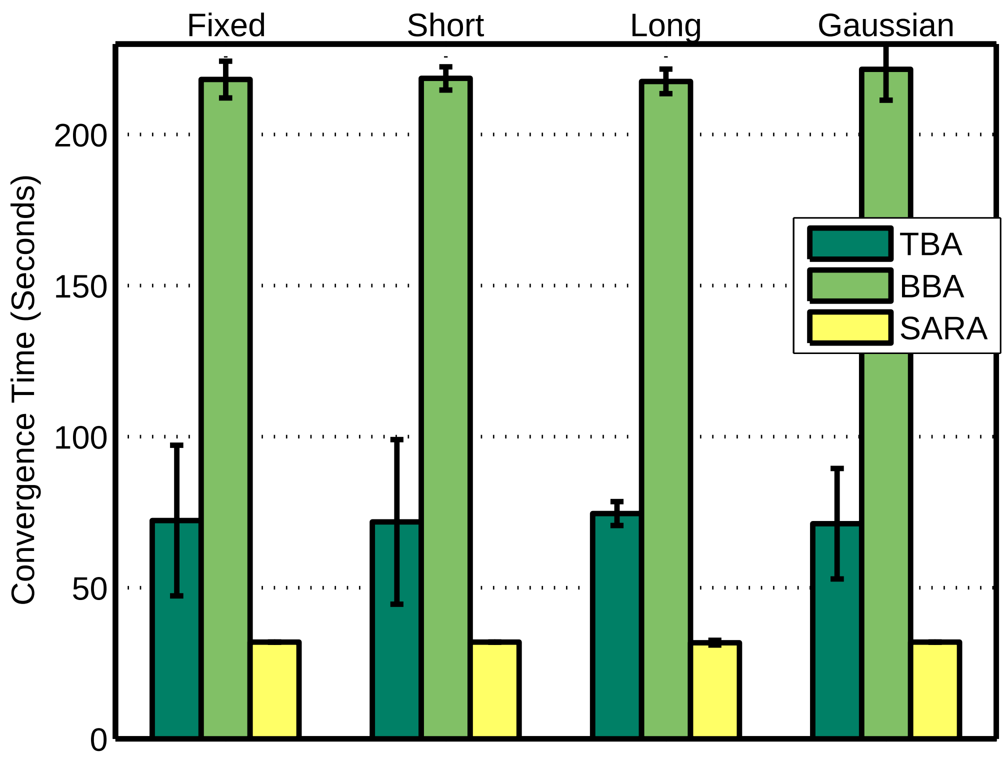


Figure 24: Convergence Time: Mean Bandwidth = 6Mbps

### 7.3.3 Convergence Time

One of the challenges of an ABR is to reduce the convergence time (Section 2.1.1). For the BBA that depends on the buffer occupancy to determine the appropriate bitrate, the convergence time was found to be higher. In the current evaluation, where the buffer size was 240 seconds (as recommended by [41]), the BBA algorithm waited until 90% (216 seconds) of the buffer was full before switching to the highest bitrate. However, in the case of SARA, the throughput measurements, along with the awareness of the segment sizes, enabled SARA to make better estimates of the download rate of the next segments.

This drove the player towards the highest bitrate much faster. In Fig. 24, we see that the convergence times of SARA were significantly better than both TBA and BBA. We found similar trends when the maximum bandwidth was reduced to 0.5, 1, or 4 Mbps.

## 7.4 Summary

The ABR algorithms employed by DASH clients can improve the QoE of the users by selecting the best bitrate. However, the existing ABR algorithms are found to be either too slow in convergence, cause unnecessary bitrate switching, or deliver video quality that is not optimal. The throughput and buffer-based adaptation techniques are found to be lacking in QoE management in low bandwidth networks. One of the reasons we found this to be as it was, was the assumption that the video segments are of the same size. The variation in the segment sizes can significantly affect the throughput and download times observed for each segment. With the Segment-Aware Rate Adaptation (SARA) algorithm we limit the error in the download rate estimation. In this chapter, we demonstrated the SARA algorithm can improve the QoE of the users in DASH systems. This improvement was demonstrated by evaluating SARA under various network conditions and by comparing its performance with the throughput-based (TBA) and buffer-based adaptation (BBA) algorithms. These evaluations showed that SARA performs significantly better than TBA and BBA with lower bitrate switching events, higher video quality, and faster convergence times. We believe that this algorithm can improve the QoE management for DASH video players in mobile networks and other bandwidth constrained network environments.

# CHAPTER 8

# CONCLUSION AND FUTURE RESEARCH

Quality of Experience for online video streaming services is an important factor in understanding the user's perception of the video services. In this theses we present solutions for the measurement and improvement of QoE for online video streaming services.

As a solution to measure the QoE of online video streaming services we designed *Pytomo*, an active QoE measurement tool that can used to emulate an user's *YouTube* browsing session and collect the QoE metrics such as video quality, playback start time, interruptions, and duration of interruptions in an automated manner. Using *Pytomo* we were able to correlate the QoE of *YouTube* videos with the video selection strategies of different ISPs. The platform independent nature and automated QoE collection of *Pytomo* enables the users, network engineers and the service providers to collect the QoE metrics from various end devices without having to make the users watch the videos.

In order to improve the QoE of DASH video streaming services, we studied the limitations of the existing ABR algorithms. We found that the existing algorithms assume that the video segments are of same sizes, which affected the estimation of segment fetch times. We then presented an enhanced MPD file that lists the individual segment sizes along with the URLs and designed a novel rate Segment Aware Rate Adaptation (SARA) algorithm. With SARA we are able to leverage the segment sizes in conjunction with the throughput measurements and buffer occupancy to provide better QoE even in low bandwidth environments. Using the GENI testbed we simulated variable network environments with short and long interruptions in playback and also a truncated Gaussian scenario that replicates real networks to study the performance of SARA in comparison with the Throughput Based Adaptation (TBA) and Buffer Based Adaptation (BBA) schemes. Based on these evaluations, we demonstrated that SARA supports a better video quality in addition to lower bitrate switching events and better convergence times. Due to the segment awareness employed by SARA, we observed that it provides consistent QoE irrespective of the video segment size distributions, which was not the case with TBA or BBA algorithms.

The next step for *Pytomo* it to extend it to perform QoE measurement for DASH videos from different video service providers (YouTube, Dailymotion etc). Secondly a mobile friendly version would enable us to perform QoE measurements for hand-held devices.

Ultimately, we plan to integrate SARA into an existing video player, to evaluate the QoE based on metrics collected during video playback and also collect the MOS score based on user feedback.

# APPENDIX A

## SAMPLE MPD FILE

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- MPD file Generated with GPAC version 0.5.1-DEV-rev5379 on
   2014-09-10T13:30:18Z-->
<MPD xmlns="urn:mpeg:dash:schema:mpd:2011" minBufferTime="PT1
   .500000S" type="static" mediaPresentationDuration="PT0H9M56
   .46S" profiles="urn:mpeg:dash:profile:isoff-live:2011">
  <Period duration="PT0H9M56.46S">
    <AdaptationSet mimeType="video/mp4" segmentAlignment="true"
         group="1" maxWidth="480" maxHeight="360" maxFrameRate="24"
          par="4:3">
      <Representation id="320x240 45.0kbps" mimeType="video/mp4"
          codecs="avc1.42c00d" width="320" height="240" frameRate
          ="24" sar="1:1" startWithSAP="1" bandwidth="45226" >
            <SegmentTemplate timescale="96" media="media/
               BigBuckBunny/4sec/bunny_$Bandwidth$bps/
               BigBuckBunny_4s$Number$%d.m4s" startNumber="1"
               duration="384" initialization="media/BigBuckBunny
               /4sec/bunny_$Bandwidth$bps/BigBuckBunny_4s_init.
               mp4" />
            <SegmentSize id="BigBuckBunny_4s1.m4s" size="168.0"
               scale="Kbits"/>
         <SegmentSize id="BigBuckBunny_4s2.m4s" size="184.0"
            scale="Kbits"/>
         <SegmentSize id="BigBuckBunny_4s3.m4s" size="200.0"
            scale="Kbits"/>
```

```
<SegmentSize id="BigBuckBunny_4s4.m4s" size="168.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s5.m4s" size="176.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s6.m4s" size="168.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s7.m4s" size="176.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s8.m4s" size="176.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s9.m4s" size="176.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s10.m4s" size="192.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s11.m4s" size="168.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s12.m4s" size="192.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s13.m4s" size="192.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s14.m4s" size="176.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s15.m4s" size="176.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s16.m4s" size="184.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s17.m4s" size="176.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s18.m4s" size="168.0"
   scale="Kbits"/>
```

```
<SegmentSize id="BigBuckBunny_4s19.m4s" size="160.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s20.m4s" size="184.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s21.m4s" size="192.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s22.m4s" size="168.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s23.m4s" size="160.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s24.m4s" size="168.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s25.m4s" size="184.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s26.m4s" size="168.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s27.m4s" size="184.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s28.m4s" size="184.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s29.m4s" size="184.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s30.m4s" size="184.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s31.m4s" size="184.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s32.m4s" size="168.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s33.m4s" size="168.0"
   scale="Kbits"/>
```

```
<SegmentSize id="BigBuckBunny_4s34.m4s" size="168.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s35.m4s" size="168.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s36.m4s" size="176.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s37.m4s" size="176.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s38.m4s" size="160.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s39.m4s" size="192.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s40.m4s" size="160.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s41.m4s" size="184.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s42.m4s" size="176.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s43.m4s" size="184.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s44.m4s" size="184.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s45.m4s" size="184.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s46.m4s" size="184.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s47.m4s" size="192.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s48.m4s" size="168.0"
   scale="Kbits"/>
```

```
<SegmentSize id="BigBuckBunny_4s49.m4s" size="200.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s50.m4s" size="168.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s51.m4s" size="184.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s52.m4s" size="184.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s53.m4s" size="192.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s54.m4s" size="184.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s55.m4s" size="192.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s56.m4s" size="176.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s57.m4s" size="168.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s58.m4s" size="168.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s59.m4s" size="168.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s60.m4s" size="152.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s61.m4s" size="176.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s62.m4s" size="184.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s63.m4s" size="184.0"
   scale="Kbits"/>
```

```
<SegmentSize id="BigBuckBunny_4s64.m4s" size="184.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s65.m4s" size="184.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s66.m4s" size="184.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s67.m4s" size="176.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s68.m4s" size="184.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s69.m4s" size="200.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s70.m4s" size="184.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s71.m4s" size="176.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s72.m4s" size="168.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s73.m4s" size="176.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s74.m4s" size="184.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s75.m4s" size="168.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s76.m4s" size="136.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s77.m4s" size="168.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s78.m4s" size="160.0"
   scale="Kbits"/>
```

```
<SegmentSize id="BigBuckBunny_4s79.m4s" size="192.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s80.m4s" size="176.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s81.m4s" size="192.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s82.m4s" size="176.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s83.m4s" size="184.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s84.m4s" size="184.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s85.m4s" size="176.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s86.m4s" size="160.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s87.m4s" size="192.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s88.m4s" size="192.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s89.m4s" size="184.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s90.m4s" size="184.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s91.m4s" size="176.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s92.m4s" size="192.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s93.m4s" size="176.0"
   scale="Kbits"/>
```

```
<SegmentSize id="BigBuckBunny_4s94.m4s" size="176.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s95.m4s" size="152.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s96.m4s" size="168.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s97.m4s" size="160.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s98.m4s" size="176.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s99.m4s" size="168.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s100.m4s" size="192.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s101.m4s" size="184.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s102.m4s" size="144.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s103.m4s" size="160.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s104.m4s" size="168.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s105.m4s" size="160.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s106.m4s" size="168.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s107.m4s" size="176.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s108.m4s" size="176.0"
   scale="Kbits"/>
```

```
<SegmentSize id="BigBuckBunny_4s109.m4s" size="184.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s110.m4s" size="176.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s111.m4s" size="200.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s112.m4s" size="152.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s113.m4s" size="160.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s114.m4s" size="176.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s115.m4s" size="184.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s116.m4s" size="136.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s117.m4s" size="176.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s118.m4s" size="192.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s119.m4s" size="160.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s120.m4s" size="160.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s121.m4s" size="168.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s122.m4s" size="168.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s123.m4s" size="160.0"
   scale="Kbits"/>
```

```
<SegmentSize id="BigBuckBunny_4s124.m4s" size="144.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s125.m4s" size="184.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s126.m4s" size="192.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s127.m4s" size="200.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s128.m4s" size="192.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s129.m4s" size="192.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s130.m4s" size="184.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s131.m4s" size="200.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s132.m4s" size="168.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s133.m4s" size="184.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s134.m4s" size="192.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s135.m4s" size="184.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s136.m4s" size="176.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s137.m4s" size="184.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s138.m4s" size="184.0"
   scale="Kbits"/>
```

```
      <SegmentSize id="BigBuckBunny_4s139.m4s" size="192.0"
         scale="Kbits"/>
      <SegmentSize id="BigBuckBunny_4s140.m4s" size="192.0"
         scale="Kbits"/>
      <SegmentSize id="BigBuckBunny_4s141.m4s" size="184.0"
         scale="Kbits"/>
      <SegmentSize id="BigBuckBunny_4s142.m4s" size="176.0"
         scale="Kbits"/>
      <SegmentSize id="BigBuckBunny_4s143.m4s" size="192.0"
         scale="Kbits"/>
      <SegmentSize id="BigBuckBunny_4s144.m4s" size="192.0"
         scale="Kbits"/>
      <SegmentSize id="BigBuckBunny_4s145.m4s" size="192.0"
         scale="Kbits"/>
      <SegmentSize id="BigBuckBunny_4s146.m4s" size="112.0"
         scale="Kbits"/>
      <SegmentSize id="BigBuckBunny_4s147.m4s" size="184.0"
         scale="Kbits"/>
      <SegmentSize id="BigBuckBunny_4s148.m4s" size="184.0"
         scale="Kbits"/>
      <SegmentSize id="BigBuckBunny_4s149.m4s" size="176.0"
         scale="Kbits"/>
      <SegmentSize id="BigBuckBunny_4s150.m4s" size="32.0"
         scale="Kbits"/>
    </Representation>
 <Representation id="320x240 89.0kbps" mimeType="video/mp4"
    codecs="avc1.42c00d" width="320" height="240" frameRate
    ="24" sar="1:1" startWithSAP="1" bandwidth="88783" >
```

```
   <SegmentTemplate timescale="96" media="media/
      BigBuckBunny/4sec/bunny_$Bandwidth$bps/
      BigBuckBunny_4s$Number$%d.m4s" startNumber="1"
      duration="384" initialization="media/BigBuckBunny
      /4sec/bunny_$Bandwidth$bps/BigBuckBunny_4s_init.
      mp4" />
   <SegmentSize id="BigBuckBunny_4s1.m4s" size="336.0"
      scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s2.m4s" size="360.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s3.m4s" size="400.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s4.m4s" size="344.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s5.m4s" size="344.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s6.m4s" size="296.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s7.m4s" size="336.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s8.m4s" size="368.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s9.m4s" size="352.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s10.m4s" size="392.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s11.m4s" size="368.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s12.m4s" size="384.0"
   scale="Kbits"/>
```

```
<SegmentSize id="BigBuckBunny_4s13.m4s" size="384.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s14.m4s" size="352.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s15.m4s" size="312.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s16.m4s" size="352.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s17.m4s" size="336.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s18.m4s" size="344.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s19.m4s" size="320.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s20.m4s" size="368.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s21.m4s" size="384.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s22.m4s" size="352.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s23.m4s" size="304.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s24.m4s" size="336.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s25.m4s" size="328.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s26.m4s" size="320.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s27.m4s" size="360.0"
   scale="Kbits"/>
```

```
<SegmentSize id="BigBuckBunny_4s28.m4s" size="368.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s29.m4s" size="360.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s30.m4s" size="360.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s31.m4s" size="360.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s32.m4s" size="336.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s33.m4s" size="328.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s34.m4s" size="344.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s35.m4s" size="328.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s36.m4s" size="360.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s37.m4s" size="312.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s38.m4s" size="288.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s39.m4s" size="376.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s40.m4s" size="304.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s41.m4s" size="352.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s42.m4s" size="328.0"
   scale="Kbits"/>
```

```
<SegmentSize id="BigBuckBunny_4s43.m4s" size="360.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s44.m4s" size="360.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s45.m4s" size="368.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s46.m4s" size="368.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s47.m4s" size="384.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s48.m4s" size="272.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s49.m4s" size="384.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s50.m4s" size="328.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s51.m4s" size="360.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s52.m4s" size="344.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s53.m4s" size="376.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s54.m4s" size="376.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s55.m4s" size="368.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s56.m4s" size="280.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s57.m4s" size="296.0"
   scale="Kbits"/>
```

```
<SegmentSize id="BigBuckBunny_4s58.m4s" size="328.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s59.m4s" size="344.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s60.m4s" size="360.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s61.m4s" size="352.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s62.m4s" size="368.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s63.m4s" size="368.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s64.m4s" size="360.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s65.m4s" size="344.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s66.m4s" size="344.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s67.m4s" size="344.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s68.m4s" size="360.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s69.m4s" size="392.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s70.m4s" size="352.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s71.m4s" size="360.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s72.m4s" size="320.0"
   scale="Kbits"/>
```

```
<SegmentSize id="BigBuckBunny_4s73.m4s" size="320.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s74.m4s" size="344.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s75.m4s" size="328.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s76.m4s" size="256.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s77.m4s" size="336.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s78.m4s" size="312.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s79.m4s" size="376.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s80.m4s" size="360.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s81.m4s" size="384.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s82.m4s" size="352.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s83.m4s" size="360.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s84.m4s" size="376.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s85.m4s" size="328.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s86.m4s" size="304.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s87.m4s" size="384.0"
   scale="Kbits"/>
```

```
<SegmentSize id="BigBuckBunny_4s88.m4s" size="376.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s89.m4s" size="368.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s90.m4s" size="360.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s91.m4s" size="328.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s92.m4s" size="384.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s93.m4s" size="352.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s94.m4s" size="328.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s95.m4s" size="336.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s96.m4s" size="336.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s97.m4s" size="304.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s98.m4s" size="328.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s99.m4s" size="304.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s100.m4s" size="392.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s101.m4s" size="360.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s102.m4s" size="280.0"
   scale="Kbits"/>
```

```
<SegmentSize id="BigBuckBunny_4s103.m4s" size="344.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s104.m4s" size="320.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s105.m4s" size="328.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s106.m4s" size="352.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s107.m4s" size="352.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s108.m4s" size="352.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s109.m4s" size="368.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s110.m4s" size="352.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s111.m4s" size="360.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s112.m4s" size="328.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s113.m4s" size="328.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s114.m4s" size="320.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s115.m4s" size="368.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s116.m4s" size="280.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s117.m4s" size="360.0"
   scale="Kbits"/>
```

```
<SegmentSize id="BigBuckBunny_4s118.m4s" size="376.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s119.m4s" size="304.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s120.m4s" size="312.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s121.m4s" size="296.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s122.m4s" size="296.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s123.m4s" size="328.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s124.m4s" size="296.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s125.m4s" size="360.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s126.m4s" size="384.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s127.m4s" size="392.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s128.m4s" size="376.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s129.m4s" size="384.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s130.m4s" size="376.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s131.m4s" size="392.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s132.m4s" size="352.0"
   scale="Kbits"/>
```

```
<SegmentSize id="BigBuckBunny_4s133.m4s" size="376.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s134.m4s" size="368.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s135.m4s" size="352.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s136.m4s" size="376.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s137.m4s" size="376.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s138.m4s" size="368.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s139.m4s" size="384.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s140.m4s" size="384.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s141.m4s" size="376.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s142.m4s" size="376.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s143.m4s" size="368.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s144.m4s" size="384.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s145.m4s" size="328.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s146.m4s" size="232.0"
   scale="Kbits"/>
<SegmentSize id="BigBuckBunny_4s147.m4s" size="288.0"
   scale="Kbits"/>
```

```
          <SegmentSize id="BigBuckBunny_4s148.m4s" size="368.0"
             scale="Kbits"/>
          <SegmentSize id="BigBuckBunny_4s149.m4s" size="360.0"
             scale="Kbits"/>
          <SegmentSize id="BigBuckBunny_4s150.m4s" size="60.8"
             scale="Kbits"/>
      </Representation>

    </AdaptationSet>
  </Period>
</MPD>
%\end{verbatim}
```

# REFERENCE LIST

[1] Big Buck Bunny Movie. http://www.bigbuckbunny.org.

[2] CAPTCHA: Telling Humans and Computers Apart Automatically. http://captcha.net/.

[3] Charles Web Debugging Proxy Application. http://www.charlesproxy.com/.

[4] Google Public DNS Resolver. https://developers.google.com/speed/public-dns/.

[5] HTTP Archive. http://httparchive.org/trends.php.

[6] ITEC DASH Dataset, note = Retrieved 12.13.2014, url = http://www-itec.uni-klu.ac.at/ftp/datasets/.

[7] Open DNS Resolver. https://www.opendns.com/.

[8] Open Source Media Framework Blog. Retrieved 5.27.2015.

[9] SQLite. http://sqlite.org.

[10] Adhikari, V., Guo, Y., Hao, F., Hilt, V., and Zhang, Z.-L. A tale of three CDNs: An active measurement study of Hulu and its CDNs. In *Computer Communications Workshops (INFOCOM WKSHPS), 2012 IEEE Conference on* (March 2012), pp. 7–12.

[11] Adhikari, V. K., Jain, S., Chen, Y., and Zhang, Z.-L. How Do You ’Tube’. In *Proceedings of the ACM SIGMETRICS Joint International Conference on Measurement and Modeling of Computer Systems* (New York, NY, USA, 2011), SIGMETRICS ’11, ACM, pp. 137–138.

[12] Agboma, F., and Liotta, A. QoE-aware QoS management. In *Proc. 6th International Conference on Advances in Mobile Computing and Multimedia (MoMM)* (2008), pp. 111–116.

[13] Akhshabi, S., Begen, A. C., and Dovrolis, C. An experimental evaluation of rate-adaptation algorithms in adaptive streaming over HTTP. In *Proc. of the 2nd ACM conf. on Multimedia systems* (2011), pp. 157–168.

[14] Alberti, C., Renzi, D., Timmerer, C., Mueller, C., Lederer, S., Battista, S., and Mattavelli, M. Automated QoE evaluation of Dynamic Adaptive Streaming over HTTP. In *Proc. Quality of Multimedia Experience (QoMEX)* (2013), pp. 58–63.

[15] Alcock, S., and Nelson, R. Application Flow Control in YouTube Video Streams. *SIGCOMM Comput. Commun. Rev. 41*, 2 (Apr. 2011), 24–30.

[16] Amazon. Amazon Mechanical Turk. Retrieved: 4.3.2014.

[17] Aroussi, S., Bouabana-Tebibel, T., and Mellouk, A. Empirical QoE/QoS correlation model based on multiple parameters for VoD flows. In *Proc. IEEE Global Communications Conference (GLOBECOM)* (2012), pp. 1963–1968.

[18] Arsan, T. Review of bandwidth estimation tools and application to bandwidth adaptive video streaming. In *High Capacity Optical Networks and Enabling Technologies (HONET), 2012 9th International Conference on* (Dec 2012), pp. 152–156.

[19] Ata, S., Huang, D., Liu, X., Wada, A., Xing, T., Juluri, P., Chung, C.-J., Sato, Y., and Medhi, D. SeRViTR: A framework, implementation, and a testbed for a trustworthy future Internet. *Computer Networks 63* (2014), 128–146.

[20] Balachandran, A., Sekar, V., Akella, A., Seshan, S., Stoica, I., and Zhang, H. Developing a predictive model of quality of experience for internet video. In *Proceedings of the ACM SIGCOMM 2013 conference on SIGCOMM* (New York, NY, USA, 2013), SIGCOMM '13, ACM, pp. 339–350.

[21] Berman, M., Chase, J. S., Landweber, L., Nakao, A., Ott, M., Raychaudhuri, D., Ricci, R., and Seskar, I. GENI: a federated testbed for innovative network experiments. *Computer Networks 61* (2014), 5–23.

[22] Callet, P. L., Möller, S., and Perkis, A. Qualinet White Paper on Definitions of Quality of Experience. *European Network on Quality of Experience in Multimedia Systems and Services (COST Action IC 1003)* (2013).

[23] Cherif, W., Ksentini, A., Negru, D., and Sidibe, M. A-PSQA: Efficient real-time video streaming QoE tool in a future media internet context. In *IEEE International Conference on Multimedia and Expo (ICME)* (2011), pp. 1–6.

[24] Cisco. Cisco Visual Networking Index: Forecast and Methodology, 20142019. Retrieved 4.3.2014.

[25] Claypool, M., and Tanner, J. The Effects of Jitter on the Perceptual Quality of Video. In *Proc. ACM Multimedia Conference* (1999), pp. 115–118.

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

[27] Csizmar Dalal, A., Kawaler, E., and Tucker, S. Towards Real-Time Stream Quality Prediction: Predicting Video Stream Quality from Partial Stream Information. In *Quality of Service in Heterogeneous Networks* (2009), vol. 22, pp. 20–33.

[28] Csizmar Dalal, A., A., Musicant, D., Olson, J., McMenamy, B., Benzaid, S., Kazez, B., and Bolan, E. Predicting User-Perceived Quality Ratings from Streaming Media Data. In *Proc. IEEE International Conference on Communications (ICC)* (2007), pp. 65 –72.

[29] Dalal, A., and Purrington, K. Discerning user-perceived media stream quality through application-layer measurements. In *Proc. International Conference on Multimedia Services Access Networks (MSAN)* (2005), pp. 44 – 48.

[30] Dalal, A. C., and Perry, E. A new architecture for measuring and assessing streaming media quality. In *Proceedings of the Third Workshop on Passive and Active Measurement Workshop (PAM 2003)* (2003), pp. 223–231.

[31] Dobrian, F., Sekar, V., Awan, A., Stoica, I., Joseph, D., Ganjam, A., Zhan, J., and Zhang, H. Understanding the Impact of Video Quality on User Engagement. In *Proc. ACM SIGCOMM* (2011), vol. 41, pp. 362–373.

[32] Engelke, U., and Zepernick, H.-J. Perceptual-based Quality Metrics for Image and Video Services: A Survey. In *Next Generation Internet Networks, 3rd EuroNGI Conference on* (May 2007), pp. 190–197.

[33] Fiedler, M., Hossfeld, T., and Tran-Gia, P. A generic quantitative relationship between quality of experience and quality of service. *Network, IEEE 24*, 2 (2010), 36–41.

[34] Finamore, A., Mellia, M., Munafò, M. M., Torres, R., and Rao, S. G. YouTube Everywhere: Impact of Device and Infrastructure Synergies on User Experience. In *Proceedings of the 2011 ACM SIGCOMM Conference on Internet Measurement Conference* (2011), IMC '11, pp. 345–360.

[35] French, H., Lin, J., Phan, T., and Dalal, A. Real time video QoE analysis of RTMP streams. In *Proc. 30th International Performance Computing and Communications Conference (IPCCC)* (2011), pp. 1 –2.

[36] Ghobadi, M., Cheng, Y., Jain, A., and Mathis, M. Trickle: Rate Limiting YouTube Video Streaming. In *Proceedings of the 2012 USENIX Conference on Annual Technical Conference* (Berkeley, CA, USA, 2012), USENIX ATC'12, USENIX Association, pp. 17–17.

[37] Gill, P., Arlitt, M., Li, Z., and Mahanti, A. YouTube Traffic Characterization: A View From the Edge. In *In: Proc. of IMC* (2007).

[38] Heidemann, J., Obraczka, K., and Touch, J. Modeling the performance of HTTP over several transport protocols. *Networking, IEEE/ACM Trans. on 5*, 5 (1997), 616–630.

[39] Hoßfeld, T., Egger, S., Schatz, R., Fiedler, M., Masuch, K., and Lorentzen, C. Initial delay vs. interruptions: Between the devil and the deep blue sea. In *Quality of Multimedia Experience (QoMEX), 2012 Fourth International Workshop on* (July 2012), pp. 1–6.

[40] Hoßfeld, T., Seufert, M., Hirth, M., Zinner, T., Tran-Gia, P., and Schatz, R. Quantification of YouTube QoE via Crowdsourcing. In *Proc. IEEE International Symposium on Multimedia (ISM)* (2011), pp. 494 –499.

[41] Huang, T.-Y., Johari, R., McKeown, N., Trunnell, M., and Watson, M. A buffer-based approach to rate adaptation: Evidence from a large video streaming service. In *Proc. of the ACM SIGCOMM* (2014), pp. 187–198.

[42] Huysegems, R., De Vleeschauwer, B., De Schepper, K., Hawinkel, C., Wu, T., Laevens, K., and Van Leekwijck, W. Session reconstruction for HTTP adaptive streaming: Laying the foundation for network-based QoE monitoring. In *Proc. IEEE International Workshop on Quality of Service (IWQoS)* (2012), pp. 1–9.

[43] ITU-T P.1201 (2012) Amd. 2. Amendment 2: New Appendix III - Use of P.1201 for non-adaptive, progressive download type media streaming. Recommendations of the ITU, Telecommunications Sector, 12 2013.

[44] ITU-T Recommendation P.10/G.100-Amendment 2. Vocabulary for performance and quality of service. Recommendations of the ITU,Telecommunications Sector, 12 2012.

[45] ITU-T Recommendation P.10/G.100-Amendment 3. Vocabulary for performance and quality of service. Recommendations of the ITU,Telecommunications Sector, 12 2012.

[46] ITU-T Recommendation P.910. Subjective video quality assessment methdods for multimedia applications. Recommendations of the ITU,Telecommunications Sector.

[47] Jain, M., and Dovrolis, C. End-to-end Estimation of the Available Bandwidth Variation Range. In *Proceedings of the 2005 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems* (New York, NY, USA, 2005), SIGMETRICS ’05, ACM, pp. 265–276.

[48] Jiang, J., Sekar, V., and Zhang, H. Improving fairness, efficiency, and stability in HTTP-based adaptive video streaming with FESTIVE. In *Proc. of the 8th Emerging networking experiments and technologies* (2012), pp. 97–108.

[49] Juluri, P. AStream: A rate adaptaion model for DASH. https://github.com/pari685/AStream. Retrieved 5.27.2015.

[50] Juluri, P., and Medhi, D. Cache'n DASH: Efficient Caching for DASH. In *ACM SIGCOMM 2015 (Poster paper)* (2015).

[51] Juluri, P., Plissonneau, L., and Medhi, D. Pytomo: A Tool for Analyzing Playback Quality of YouTube Videos. In *Proc. International Teletraffic Congress (ITC)* (2011), pp. 304–305. (poster paper).

[52] Juluri, P., Plissonneau, L., Zeng, Y., and Medhi, D. Viewing YouTube from a metropolitan area: What do users accessing from residential ISPs experience? In *Integrated Network Management (IM 2013), 2013 IFIP/IEEE International Symposium on* (2013), IEEE, pp. 589–595.

[53] Juluri, P., Tamarapalli, V., and Medhi, D. Look-Ahead Rate Adaptation Algorithm for DASH under Varying Network Environments. In *11th DRCN 2015 (Poster paper)* (2015).

[54] Juluri, P., Tamarapalli, V., and Medhi, D. Measurement of Quality of Experience of Video-on-Demand Services: A Survey. *Communications Surveys Tutorials, IEEE PP*, 99 (2015), 1–1.

[55] Juluri, P., Tamarapalli, V., and Medhi, D. QoE Management in DASH Systems Using Segment Aware Rate Adaptation Algorithm. In *Network and Service Management (CNSM), 2015 11th International Conference on* (2015).

[56] Juluri, P., Tamarapalli, V., and Medhi, D. SARA: Segment Aware Rate Adaptation Algorithm for Dynamic Adaptive Streaming over HTTP. In *ICC QoE-FI Workshop* (2015).

[57] Jumisko-Pyykkö, S., Kumar, V., V, M., Liinasuo, M., and Hannuksela, M. Acceptance of Audiovisual Quality in Erroneous Television Sequences over a DVB-H Channel. *Proc. International Workshop in Video Processing and Quality Metrics for Consumer Electronics* (2006).

[58] Khirman, S., and Henriksen, P. Relationship between Quality-of-service and Quality-of-Experience for Public Internet Service. In *Proc. 3rd Passive Active Measurement Workshop* (2002).

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

[60] Krishnappa, D. K., Bhat, D., and Zink, M. DASHing YouTube: An analysis of using DASH in YouTube video service. *38th Annual IEEE Conference on Local Computer Networks 0* (2013), 407–415.

[61] Kuznetso, A. N. Traffic contol settings. http://manpages.ubuntu.com/manpages/precise/man8/tc.8.html.

[62] Latré, S., Staelens, N., Simoens, P., Vleeschauwer, B. D., de Meerssche, W. V., Turck, F. D., Dhoedt, B., Demeester, P., den Berghe, S. V., Huysegems, R., and

Verzijp, N. On-line Estimation of the QoE of Progressive Download Based Services in Multimedia Access Networks. In *Proc. International Conference on Internet Computing* (2008).

[63] Leider, R. MPEG DASH SuperSession - a session sponsored by DASH. Tech. rep., IBC, 2014.

[64] Lewcio, B., Belmudez, B., Enghardt, T., and Moller, S. On the way to high-quality video calls in future mobile networks. In *Proc. Quality of Multimedia Experience (QoMEX)* (2011), pp. 43–48.

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

[66] Liu, C. Views in BIND 9. May 2000, `http://www.oreillynet.com/pub/a/oreilly/networking/news/views_0501.html`.

[67] Liu, C., Bouazizi, I., and Gabbouj, M. Rate Adaptation for Adaptive HTTP Streaming. In *Proceedings of the Second Annual ACM Conference on Multimedia Systems* (New York, NY, USA, 2011), MMSys '11, ACM, pp. 169–174.

[68] Liu, C., Bouazizi, I., and Gabbouj, M. Rate adaptation for adaptive HTTP streaming. In *Proc. of the 2nd ACM conf. on Multimedia systems* (2011), pp. 169–174.

[69] Liu, X., Dobrian, F., Milner, H., Jiang, J., Sekar, V., Stoica, I., and Zhang, H. A case for a coordinated internet video control plane. In *Proc. ACM SIGCOMM* (2012), pp. 359–370.

[70] Liu, X., Juluri, P., and Medhi, D. An experimental study on dynamic network reconfiguration in a virtualized network environment using autonomic management. In *Integrated Network Management (IM 2013), 2013 IFIP/IEEE International Symposium on* (2013), IEEE, pp. 616–622.

[71] Liu, X., Wada, A., Xing, T., Juluri, P., Sato, Y., Ata, S., Huang, D., and Medhi, D. SeRViTR: A framework for trust and policy management for a secure Internet and its proof-of-concept implementation. In *Network Operations and Management Symposium (NOMS), 2012 IEEE* (2012), IEEE, pp. 1159–1166.

[72] Microworkers. Microworkers. Retrieved 4.3.2014.

[73] Miller, K., Quacchio, E., Gennari, G., and Wolisz, A. Adaptation algorithm for adaptive streaming over HTTP. In *Packet Video Workshop (PV), 2012 19th International* (May 2012), pp. 173–178.

[74] Mok, R., Chan, E., and Chang, R. Measuring the quality of experience of HTTP video streaming. In *Proc. IFIP/IEEE International Symposium on Integrated Network Management (IM)* (2011), pp. 485 –492.

[75] Mok, R. K., Chan, E. W., Luo, X., and Chang, R. K. Inferring the QoE of HTTP video streaming from user-viewing activities. In *Proc. ACM SIGCOMM workshop on Measurements Up the Stack (W-MUST)* (2011), pp. 31–36.

[76] Mok, R. K., Luo, X., Chan, E. W., and Chang, R. K. QDASH: a QoE-aware DASH system. In *Proceedings of the 3rd Multimedia Systems Conference* (2012), ACM, pp. 11–22.

[77] Montenovo, M., Perot, A., Carli, M., Cicchetti, P., and Neri, A. Objective quality evaluation of video services. *Second International Workshop on Video Processing and Quality Metrics for Consumer Electronics* (2006).

[78] MPEG. MPEG Standards. `http://mpeg.chiariglione.org/standards/mpeg-dash`. [Online: accessed 17-September-2013].

[79] Plissonneau, L., and Biersack, E. A Longitudinal View of HTTP Video Streaming Performance. In *MMSys 2012, ACM Multimedia Systems, Chapel Hill, USA* (2012).

[80] Plissonneau, L., Biersack, E., and Juluri, P. Analyzing the impact of YouTube delivery policies on user experience. In *Proc. International Teletraffic Congress (ITC)* (2012), pp. 1–8.

[81] Qi, Y., and Dai, M. The effect of frame freezing and frame skipping on video quality. In *Intelligent Information Hiding and Multimedia Signal Processing, 2006. IIH-MSP'06. International Conference on* (2006), IEEE, pp. 423–426.

[82] Rugel, S., Bauschert, T., Knoll, T. M., and Eckert, M. A Network-based Method for Measurement of Internet Video Streaming Quality. In *ITG-FG 5.2.3 - Next Generation Networks* (2011).

[83] Saxena, M., Sharan, U., and Fahmy, S. Analyzing Video Services in Web 2.0: A Global Perspective. In *Proc. International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV)* (2008), pp. 39–44.

[84] Schatz, R., Hoßfeld, T., and Casas, P. Passive YouTube QoE Monitoring for ISPs. In *Proc. International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS),* (2012), pp. 358–364.

[85] Schatz, R., Hoßfeld, T., Janowski, L., and Egger, S. From Packets to People: Quality of Experience as a New Measurement Challenge. In *Data Traffic Monitoring and Analysis*, E. Biersack, C. Callegari, and M. Matijasevic, Eds., vol. 7754 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2013, pp. 219–263.

[86] Serral-Gracià, R., Cerqueira, E., Curado, M., Yannuzzi, M., Monteiro, E., and Masip-Bruin, X. An Overview of Quality of Experience Measurement Challenges for Video Applications in IP Networks. In *Proc. International Conference on Wired/Wireless Internet Communications (WWIC)* (2010), pp. 252–263.

[87] Shaikh, J., Fiedler, M., and Collange, D. Quality of Experience from user and network perspectives. *annals of telecommunications - annales des tÃ©lÃ©communications 65*, 1-2 (2010), 47–57.

[88] Shao, B., Renzi, D., Amon, P., Xilouris, G., Zotos, N., Battista, S., Kourtis, A., and Mattavelli, M. An adaptive system for real-time scalable video streaming with end-to-end QoS control. In *Proc. 11th International Workshop on Image Analysis for Multimedia Interactive Services (WIAMIS)* (2010), pp. 1–4.

[89] Sidibé, M., Koumaras, H., Kofler, I., Mehaoua, A., Kourtis, A., and Timmerer, C. A novel monitoring architecture for media services adaptation based on network QoS to perceived QoS mapping. *Signal, Image and Video Processing 2*, 4 (2008), 307–320.

[90] Singh, K., Hadjadj-Aoul, Y., and Rubino, G. Quality of experience estimation for adaptive HTTP/TCP video streaming using H.264/AVC. In *Proc. Consumer Communications and Networking Conference (CCNC)* (2012), pp. 127–131.

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

[92] Staehle, B., Hirth, M., Pries, R., Wamser, F., and Staehle, D. YoMo: a Youtube application comfort monitoring tool. *New Dimensions in the Assessment and Support of Quality of Experience for Multimedia Applications, Tampere, Finland* (2010), 1–3.

[93] Teyeb, O., Sørensen, T. B., Mogensen, P., and Wigard, J. Subjective evaluation of packet service performance in UMTS and heterogeneous networks. In *Proc. ACM international workshop on Quality of service & security for wireless and mobile networks (Q2SWinet)* (2006), pp. 95–102.

[94] Torres, R., Finamore, A., Kim, J. R., Mellia, M., Munafo, M. M., and Rao, S. Dissecting Video Server Selection Strategies in the YouTube CDN. In *Proceedings of the 2011 31st International Conference on Distributed Computing Systems (ICDCS'11)* (2011), pp. 248–257.

[95] Tstat. Tstat: TCP Statistic and Analysis Tool. Retrieved: 4.3.2014.

[96] Vera, D., RodrÃguez-Bocca, P., and Rubino, G. QoE Monitoring Platform for Video Delivery Networks. In *IP Operations and Management*, D. Medhi, J. Nogueira, T. Pfeifer, and S. Wu, Eds., vol. 4786 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2007, pp. 131–142.

[97] Vlado Menkovski, Georgios Exarchakos, A. L. Online Learning for Quality of Experience Management. In *Proc. 19th Machine Learning conference of Belgium and The Netherlands* (2010).

[98] Vleeschauwer, B. D., Meerssche, W. V. D., Simoens, P., Turck, F. D., Dhoedt, B., Demeester, P., Struyve, K., Caenegem, T. V., Gilon, E., Dequeker, H., and Six, E. Enabling autonomic access network QoE management through TCP connection monitoring. In *Proc. IEEE Workshop on Autonomic Communications and Network Management of (ACNM)* (2007), pp. 56–63.

[99] Wang, Z., Lu, L., and Bovik, A. Video quality assessment using structural distortion measurement. In *Image Processing. 2002. Proceedings. 2002 International Conference on* (2002), vol. 3, pp. III–65–III–68 vol.3.

[100] Winkler, S., Sharma, A., and McNally, D. Perceptual Video Quality and Blockiness Metrics for Multimedia Streaming Applications. In *in Proceedings of the International Symposium on Wireless Personal Multimedia Communications* (2001), pp. 547–552.

[101] Yang, F., Song, J., Wan, S., and Wu, H. R. Content-Adaptive Packet-Layer Model for Quality Assessment of Networked Video Services. *Selected Topics in Signal Processing, IEEE Journal of 6*, 6 (Oct 2012), 672–683.

[102] Yitong, L., Yun, S., Yinian, M., Jing, L., Qi, L., and Dacheng, Y. A study on Quality of Experience for adaptive streaming service. In *Communications Workshops (ICC), 2013 IEEE International Conference on* (June 2013), pp. 682–686.

[103] You, J., Reiter, U., Hannuksela, M. M., Gabbouj, M., and Perkis, A. Perceptual-based Quality Assessment for Audio-visual Services: A Survey. *Image Commun. 25*, 7 (Aug. 2010), 482–501.

[104] YouTube. YouTube Statistics. `http://www.youtube.com/yt/press/statistics.html`, 2015. [Online; accessed 6-July-2015].

# VITA

Parikshit Juluri was born on December 6, 1985 in Hyderabad, Telangana, India. He attended Mahatma Gandhi Institute of Technology, affiliated to Jawaharlal Nehru Technological University in September 2003 and graduated in May 2007 with a Bachelor‘s in Technology in Electrical and Electronics Engineering.

Mr Juluri attended University of Missouri - Kansas City (UMKC) in September 2007 and received his Master of Science in Electrical Engineering in December 2008. Mr. Juluri joined the Inter-disciplinary Ph.D program in UMKC with Telecommunications and Computer Networking (TCN) and Electrical Engineering as his coordinating discipline and co-discipline, respectively.

Mr. Juluri received Best Student Poster Awards in the $23^{rd}$ International Teletraffic Congress (ITC 2011) in San Francisco, CA and $11^{th}$ International Conference on Design of Reliable Communications Networks, 2015 at Kansas City, USA.

Mr. Juluri was a Network Analyst Intern at Orange Labs (France Telecom), Nice, France from January to June, 2011. Mr. Juluri was a Core Network Engineering Intern at Samsung Telecommunications America (STA), Richardson, Texas in the Summers of 2012, 2013 and 2014.

Upon completion of his Ph.D requirements, Mr. Juluri plans to continue his research in the measurement and improvement of Quality of Experience (QoE) for online media services.