Measuring and Improving the Quality of Experience of Adaptive Rate Video

Hyunwoo Nam

Submitted in partial fulfillment of the
requirements for the degree
of Doctor of Philosophy
in the Graduate School of Arts and Sciences

COLUMBIA UNIVERSITY

2016

ABSTRACT

Measuring and Improving the Quality of Experience of Adaptive Rate Video

Hyunwoo Nam

Today's popular over-the-top (OTT) video streaming services such as YouTube, Netflix and Hulu deliver video contents to viewers using adaptive bitrate (ABR) technologies. In ABR streaming, a video player running on a viewer's device adaptively changes bitrates to match given network conditions. However, providing reliable streaming is challenging. First, an ABR player may select an inappropriate bitrate during playback due to the lack of direct knowledge of access networks, frequent user mobility and rapidly changing channel conditions. Second, OTT content is delivered to viewers without any cooperation with Internet service providers (ISPs). Last, there are no appropriate tools that evaluate the performance of ABR streaming along with video quality of experience (QoE).

This thesis describes how to improve the video QoE of OTT video streaming services using ABR technologies. Our analysis starts from understanding ABR heuristics. How does ABR streaming work? What factors does an ABR player consider when switching bitrates during a download? Then, we propose our solutions to improve existing ABR streaming from the perspective of network operators who deliver video content through their networks and video service providers who build ABR players running on viewers' devices.

From the network operators' point of view, we propose to find a better video content server based on round trip times (RTTs) between an edge node of a wireless network and available video content servers when a viewer requests a video. The edge node can be an Internet Service Provider (ISP) router in a Wi-Fi network and a packet data network gateway (P-GW) in a 4G network. During the experiments, our solution showed better TCP performance (e.g., higher TCP throughput during playback) 146 times out of 200 experiments (73%) over Wi-Fi networks and 162 times out of 200 experiments (81%) over

3G networks. In addition, we claim that the wireless edge nodes can assist an ABR video player in selecting the best available bitrate by controlling the available bandwidth in the radio access network between a base station and a viewer's device. In our Wi-Fi testbed, the proposed solution saved up to 21% of radio bandwidth on mobile devices and enhanced the viewing experience by reducing rebufferings during playback. Last, we assert that software-defined networking (SDN) can improve video QoE by dynamically controlling routing paths of video streaming flows based on the provisioned networking information collected from SDN-enabled networking devices. Using an off-the-shelf SDN platform, we showed that our proposed solution can reduce rebufferings by 50% and provide higher bitrates during a download.

From the perspective of video service providers, higher video QoE can be achieved by improving ABR heuristics implemented in an ABR player. To support this idea, we investigated the role of playout buffer size in ABR streaming and its impact on video QoE. Through our video QoE survey, we proved that a large buffer does not always outperform a small buffer, especially under rapidly varying network conditions. Based on this finding, we suggest to dynamically change the maximum buffer size in an ABR player depending on the current capacity of its playout buffer for improving the QoE of viewers. During the experiments, our proposed solution improved the viewing experience by offering 15% higher average played bitrate, 70% fewer bitrate changes and 50% shorter rebuffering duration.

Our experimental results show that even small changes of ABR heuristics and new features of network systems can greatly affect video QoE. However, it is still difficult for video service providers or network operators to evaluate new ABR heuristics or network system changes due to lack of accurate QoE monitoring systems. In order to solve this issue, we have developed YouSlow ("YouTube Too Slow!? - YouSlow") as a new approach to monitoring video QoE for the analysis of ABR performance. The lightweight web browser plug-in and mobile application are designed to monitor various playback events (e.g., rebuffering duration and frequency of bitrate changes) directly from within ABR video players and calculate statistics along with video QoE. Using YouSlow, we investigate the impact of the above playback events on video abandonment: about 10% of viewers abandoned the YouTube videos when the pre-roll ads lasted for 15 seconds. Even increasing the bitrate can

annoy viewers; they prefer a high starting bitrate with no bitrate changes during playback. Our regression analysis shows that bitrate changes do not affect video abandonment significantly and the abandonment rate can be estimated accurately using the rebuffering ratio and the number of rebufferings ($R^2 = 0.94$).

The thesis includes four main contributions. First, we investigate today's popular OTT video streaming services (e.g., YouTube and Netflix) that use ABR streaming technologies. Second, we propose to build QoS and QoE aware video streaming that can be implemented in existing wireless networks (e.g., Wi-Fi, 3G and 4G) and in SDN-enabled networks. Third, we propose to improve current ABR heuristics by dynamically changing the playout buffer size under varying network conditions. Last, we designed and implemented a new monitoring system for measuring video QoE.

Table of Contents

List of Figures

List of Tables

Acknowledgments

First and foremost, I would like to express my deepest gratitude to my advisor, Prof. Henning Schulzrinne for supporting me for the past five years. Under his insightful guidance, I learned how to find problems systematically, pay attention to detail and be practical in all the time of research. I could not ask for a better advisor in my Ph.D. journey.

I would like to thank my dissertation defense committee members: Prof. Shih-Fu Chang, Prof. Vishal Misra, Prof. Gil Zussman and Dr. Doru Calin. Their criticisms and suggestions for my Ph.D. thesis improved my research. I look forward to more of their insights as I continue my research in the future.

I'm very glad to have worked with wonderful researchers from Bell Labs: Dr. Doru Calin and Bongho Kim. I worked with them closely and we published many papers together during my past internships at Bell Labs. Without their valuable research experiences, I would not have been able to thrive in my doctoral program and balance my research.

I would also like to thank Hwa-jung Han and Jung Woo Lee from Verizon. They sponsored a part of my research and jump started my video streaming research with insightful advices when I first began.

I want to thank all the members of the IRT group and my colleagues: Jae Woo Lee (his enthusiasm and love for teaching is contagious), Se Gi Hong (I appreciate his career advices and tips in life as well), Wonsang Song (helped me a lot during the early years of my research), Jong Yul Kim (the most nice and friendly person I've worked with), Kyung-Wook Hwang (helped me a lot during my Ph.D. years at Columbia), Kyung Hwa Kim (this guy is the one I spent most of my time with for the past five years and I was very happy to be a part of his wedding) and Jan Janak (the smartest and nicest guy, who I would go to when I had questions). It would have been impossible for me to finish my dissertation without them. Thank you.

I would also like to thank my lovely cat, Benjamin, previously known as Captain Cat during his shelter days. He was my first foster cat three years ago. Now weighting 18 pounds, he is my forever roommate and a professional couch potato.

I sincerely thank Jongmee Kim for sticking by my side. She is the only person who enjoys and appreciates my sense of humor. With Benjamin, she was always there when I needed. None of these mean much without her.

Last but not least, I especially want to thank my family, my mother Mija Shon, my father Myung Bong Nam, my sister Hyuna Nam. They always told me, “take your time as you need”. Sorry, it took a while. This thesis is possible because of their support and patience.

To my mother Mija Shon and my father Myung Bong Nam.

Chapter 1

Introduction

Over-the-top (OTT) video refers to delivery of audio and video over the Internet without any involvement of network operators [1; 2]. Example applications include Netflix, YouTube, HBO Go and Hulu, which can be distinguished from other OTT services such as Skype or Facetime for interactive voice and video calls, on the one hand, and WhatsApp, Facebook messenger, KakaoTalk or WeChat for asynchronous instant messages on mobile devices. Such OTT video streaming platform contrasts with the traditional billing models - purchasing media services from an Internet service provider (ISP) such as Internet Protocol television (IPTV) or Pay-TV video on demand (VOD). The ISPs deliver the video in best-effort mode, thus they are not responsible for reliable delivery and viewing quality while a video is being played [3].

The popularity of OTT video streaming is growing steadily. According to recent reports [4; 5; 6; 7], video streaming traffic accounts for more than 50% of total online data consumption in 2014, and it is predicted to reach 66% by the end of 2015. The growth of video streaming is boosted by the increasing number of viewers on mobile devices such as smartphones and tablets. For example, 89 million U.S. viewers were watching online videos on their mobile devices in 2014. In 2016, the number is predicted to reach 110 million, and the total amount of traffic is expected to exceed 50% of total mobile data consumption.

There are a number of video streaming protocols [8]. Real-time Transport Protocol (RTP) [9] is used for transfer of streaming media over IP networks. The Real Time Streaming Protocol (RTSP) [10] is designed to control media sessions between clients and servers.

Adobe's Real Time Messaging Protocol (RTMP) [11] is a network protocol for delivering live and on-demand streaming to Adobe Flash applications. Microsoft Media Server (MMS) [12] is a Microsoft's proprietary network streaming protocol to transfer unicast data in Windows media services. Today's OTT video service providers stream their videos to viewers over HTTP or HTTPS. Such HTTP-based video streaming generally uses two different technologies: *progressive download* and adaptive bitrate (ABR[1]). In *progressive download*, a video server streams only at a single bitrate [14] and viewers must watch the same video bitrate regardless of their local network conditions. Therefore, it is possible for a viewer to frequently experience rebufferings (i.e., a video is paused and then resumes playing repeatedly) when the bitrate requires higher network bandwidth than what a network can handle. In order to resolve the problem, several ABR technologies have been introduced, such as Apple HTTP Live Streaming (HLS) [15], Microsoft IIS Smooth Streaming (SS) [16], Adobe HTTP Dynamic Streaming (HDS) [17] and Dynamic Adaptive Streaming over HTTP (DASH) [18]. In ABR streaming, a video server contains several video files encoded in segments at multiple bitrates. The ABR player running on the viewer's device adapts the best available bitrate based on various factors such as estimated bandwidth in the network or hardware specifications (e.g., smartphones or desktops) of the viewer's device.

This thesis explores today's popular OTT streaming services. How do they deliver video content to the viewers? What factors are taken into account when switching bitrates and when does an ABR player change bitrates? To achieve higher Quality of Service (QoS) and Quality of Experience (QoE) of existing OTT video streaming, we propose to dynamically select the best video content servers and control streaming flows based on changing network conditions in mobile or software-defined networking (SDN) enabled networks. We suggest that ABR players should adjust their playout buffer size depending on the remaining buffer occupancy during a download. In addition, we designed and implemented a new video monitoring system called YouSlow ("YouTube Too Slow!? - YouSlow") that can evaluate various QoE metrics while a video is being delivered to the viewer.

[1]The term of ABR used in this thesis denotes an adaptive bitrate streaming technology. It is different from Available Bit Rate (ABR) services in Asynchronous Transfer Mode (ATM) networks [13], where ATM switches use local network information to dynamically control the allowable rates among users in the network.

1.1 Challenges

In spite of the increasing popularity of VOD viewing and advanced ABR streaming technologies, we are still faced with several challenges when providing reliable streaming services:

The first challenge is that OTT video service providers stream their videos without any cooperation with network operators. In order to provide quality of service (QoS) for video streaming, however, network operators need to understand how video content is delivered through their networks. For instance, they can apply different networking strategies (e.g., dynamic resource allocation in radio access networks and different queuing management on routers) depending on various video bitrates and ABR heuristics that are used by different OTT streaming services.

The second challenge from the perspective of video service providers is the difficulty of building a better ABR media player without an in-depth analysis of ABR heuristics. The performance of ABR streaming entirely relies on the ABR heuristics implemented in the ABR player. Thus, to improve ABR streaming, we need to analyze what kind of factors (e.g., bitrate, playout buffer status and screen size of a viewer's device) are used in the ABR heuristics, and understand how the factors affect bitrate switching during a download. We also note that an ABR player is designed to estimate available downloading throughput to select the best bitrate while a video is being delivered to the viewer. Because the player has no access to transit or last mile networks of viewers, it is necessary to estimate network conditions accurately to find the right switching point among bitrates during playback.

The last challenge is the lack of appropriate monitoring and evaluation tools that can analyze the performance of ABR streaming. Without knowing the impact of ABR heuristic or network system changes on video quality of experience (QoE), it is difficult for network operators and video service providers to build better video streaming platform. Some researchers [19; 20; 21] have used QoS metrics (e.g., monitoring throughput, goodput, packet delay and jitter from network middle-boxes between viewers' devices and video servers) to estimate QoE of viewers. However, these metrics are typically used to pinpoint network impairments, and do not accurately reflect the viewer's watching experience.

1.2 Overview and main contributions

This thesis consists of four parts. Part I focuses on investigating OTT video streaming platform using ABR technologies. We analyze how they deliver video contents to viewers and how ABR streaming works. Hence, the analysis can be regarded as a prelude to the following parts. Parts II and III, which consist of Chapters 3 to 6, explain our solutions on improving current ABR streaming from the perspective of network operators and video service providers. Part IV introduces our new video monitoring system that analyzes various QoE metrics while a video is being delivered to the viewer. We briefly describe the main contribution of each chapter:

Chapter 2 of Part I focuses on the analysis of ABR streaming [22; 23]. For an empirical study, we analyze ABR streaming of YouTube and Netflix while playing the videos on iOS and Android mobile devices via different access networks, namely Wi-Fi, 3G, and 4G. Next, we investigate ABR heuristics that are used in bitrate switching during playback. We first start from comparing two HTTP-based video streaming technologies: *progressive download* and ABR. Based on the study of popular ABR technologies (Apple's HLS, Microsoft's SS, and 3GPP/MPEG DASH), we found that the bitrate switching decision can vary depending on current playout buffer level, network conditions and hardware specification of a viewer's device.

Chapter 3 of Part II presents a dynamic network condition-aware video content server selection algorithm [24]. Our analysis shows that the YouTube delivery cloud typically assigns a video content server in content delivery networks (CDNs) that is geographically close to a viewer. However, the network conditions between a viewer and a video content server can be unstable even if the video content server is located near the viewer. In order to solve this problem, we propose to discover a video content server that offers better network conditions between a corresponding edge node of a wireless network and video content servers. The edge node can be an ISP router in a Wi-Fi network, a Radio Network Controller (RNC) node in a 3G network or a Packet Data Network Gateway (P-GW) in a 4G network. In our proposed architecture, an edge node selects a preferred video content server based on measured round trip times (RTTs) when a viewer requests a video. Our evaluation proves that a video content server chosen by our RTT-based video content server selection algorithm typically provides more reliable viewing experiences during playback, with higher TCP performance than location-based algorithms.

Chapter 4 introduces QoS-aware video streaming in wireless networks [23]. While understanding ABR streaming in Part I, we found that an ABR player may discard a large part of video content even though it is successfully delivered to a viewer. This unwanted behavior often occurs when the player changes the bitrates under fluctuating network conditions and the playout buffer is full while downloading a video. Some of the measurements show that the discarded data may exceed 35% of the total video content. In order to reduce this waste of network resources and improve the QoE for viewers, our solution suggests a selective packet discarding mechanism that can be placed in a wireless edge node (e.g., local ISP routers or P-GWs). In addition, our QoS-aware rules assist the ABR player in selecting an appropriate bitrate by dynamically controlling the available bandwidth of the streaming flow under changing network conditions. In our experimental setup, the proposed platform shows up to 20% of improvement in saving down-link bandwidth as well as reducing rebuffering events for improved QoE.

Chapter 5 introduces QoE-aware video streaming using software-defined networking (SDN) [25]. To provide smooth streaming under congested network conditions, an ABR player reduces the load by downgrading to lower bitrates. However, the low quality of video can also interrupt viewing experience during playback. To resolve the root causes of congestion problems and improve QoE, we propose to leverage an SDN platform in ABR streaming. Our proposed SDN application monitors the network conditions of streaming flows in real time and dynamically changes routing paths in wide area networks (WANs) using multi-protocol label switching (MPLS) traffic engineering (TE). We use an off-the-shelf SDN platform (Juniper network's Junos Space [26]) to show the feasibility of our approach. In our testbed setup, the SDN controller automatically selects better routing paths under congested networks to provide smoother streaming which leads to more than 50% reduction in rebuffering events.

Chapter 6 of Part III analyzes the impact of playout buffer size in ABR streaming on video QoE. Through Part I, we showed that playout buffer size plays a key role in bitrate switching during playback. In Chapter 6, we demonstrate that a large buffer typically causes fewer bitrate changes and rebufferings compared to a small buffer. However, we also observe that a small buffer may achieve higher QoE by providing high bitrate shortly than a large buffer, especially under fast varying network conditions. To verify this, we conduct subjective experiments collecting data from more than 200 participants using an online crowdsourcing platform. Based on these findings, we propose that ABR players adaptively change their maximum playout buffer size depending on the remaining buffer occupancy during a download. In our testbed, we observe that an ABR player that dynamically switches between small and large buffers can reduce the number of bitrate changes up to 70%, rebuffering duration up to 50% and increase average played bitrate up to 15% compared to players with a fixed buffer size.

Chapter 7 of Part IV introduces YouSlow [27; 28; 29], an application that can detect various playback events (e.g., starting bitrate, start-up delay, rebufferings and bitrate changes) directly from within video players embedded in web browsers or mobile applications. Using YouSlow, we analyzed more than 1,400,000 YouTube views from more than 110 countries. We investigate the impact of the above playback events on video abandonment: about 10% of viewers abandoned the videos when the pre-roll ads lasted for 15 seconds in YouTube. More viewer abandoned the videos when they suffered from multiple rebufferings than a single rebuffering event, even if the total rebuffering duration is the same. Our analysis shows that viewers prefer constant bitrate to increasing bitrate during playback. We show that tracking rebuffering ratio during playback is useful to quantify abandonment rates for short videos. Our regression analysis using the rebuffering ratio and the number of rebufferings achieves an R-squared value of 0.94 in predicting the video abandonment rate in YouTube.

Part I

Prelude: Understanding OTT Video Streaming and ABR Streaming Technologies

Chapter 2

An Empirical Study of OTT Video Streaming

2.1 Introduction

Today's popular video streaming services (e.g., Netflix, Hulu and YouTube) stream video contents to viewers over HTTP. To provide smooth streaming, they use ABR technologies such as Apple HTTP Live Streaming (HLS) [15], Microsoft IIS Smooth Streaming (SS) [16], Adobe HTTP Dynamic Streaming [17] and Dynamic Adaptive Streaming over HTTP (DASH) [18]. In ABR streaming, a video player is designed to dynamically adjust the video bitrate based on estimated network conditions, buffer occupancy and hardware specifications (e.g., smartphones vs. desktops) of viewers' devices. Therefore, depending on how appropriately the player selects the best available bitrate during playback, user-perceived video quality can vary. As an example, a viewer may experience frequent rebufferings when the player requests a higher bitrate than what is actually available in the network, or there is also a possibility of being stuck with a low bitrate throughout the entire playback if the network capacity is underestimated by the player. Thus, analyzing ABR technologies is the first step to understanding OTT video streaming platforms.

This chapter focuses on the analysis of OTT video streaming platforms and ABR technologies. We make two contributions: First, we analyze ABR technologies and its fundamental heuristics. Our analysis starts from understanding ABR technologies compared to *progressive download*. Then, we examine the existing ABR heuristics to answer the following questions: How does ABR streaming work? When does an ABR player change bitrates during a download? What factors are taken into account for a bitrate switching? Second, for an empirical analysis, we investigate today's two popular video streaming services, namely YouTube and Netflix, while playing the videos on mobile devices (iOS and Android) using their own applications (not by a web browser) over wireless networks (Wi-Fi, 3G and 4G) under varying network conditions. Our experimental results show that an ABR player downloads video content by sending a series of HTTP GET messages during playback, and the downloading traffic behavior varies depending on the operating system (OS), the hardware performance of a viewer's device and the network condition. Compared to Android, for example, YouTube video player for iOS sends more HTTP GET messages via new TCP connections to download duplicate video content for a possible later re-play by the viewer. We will explain the details in Section 2.4.

The remainder of the chapter is organized as follows. We briefly describe online video delivery background in Section 2.2. In Section 2.3, we introduce ABR streaming technologies and its heuristics. In Section 2.4, we analyze the YouTube and Netflix video streaming platforms. We summarize our insights in Section 2.5.

2.2 Online video delivery background

During the early days of video streaming, proprietary protocols such as Microsoft Media Server (MMS) [12] and Real Player's Progressive Networks (PNM/PNA) [30] were developed to stream video and audio data. These non-open protocols have largely been replaced by the open standard protocols such as Real-time Transport Protocol (RTP) [9] and Real Time Streaming Protocol (RTSP) [10]. RTP is a real-time end-to-end transport protocol that transfers the actual video and audio data over multicast or unicast network services. It typically runs on top of IP and UDP. RTSP is an application level network protocol to manage multiple end-to-end media sessions, and allows a viewer to control the delivery of streaming from a media server such as *play*, *pause* and *fast forwarding* during playback. The main problem of using RTP is that its payload format is not codec agnostic. This means that a new media codec cannot be easily supported in RTP unless a new playload format standard is agreed upon [31]. In addition, firewalls and network address translation (NAT) routers would block such UDP-based video delivery. Due to these problems, most of today's OTT streaming services have adopted HTTP-based streaming technologies such as *progressive download* and ABR streaming. Below, we briefly describe three popular streaming technologies; *progressive download*, RTMP/RTSP streaming and ABR streaming.

2.2.1 Progressive download

In *progressive download*, a video is delivered by a regular HTTP web server over HTTP rather than a streaming server. This mechanism is easy to setup and cost-effective since it does not require any special streaming servers. When there is a video request, an HTTP web server pushes the video content as quickly as it can. The playback can start as soon as enough content has been downloaded and *fast forwarding* (skipping ahead) is only possible for the downloaded content. There is a security concern since the player caches the video content on the viewer's device and it is easy to copy. To prevent this, Digital Rights Management (DRM) can be used to protect the audio and video content [32]. Moreover, *progressive download* provides no quality adjustment; no matter what download speed is experienced and what devices are used, the player downloads the same quality of the video file.

2.2.2 RTMP/RTSP chunk based delivery

This content delivery mechanism uses RTMP (Real Time Messaging Protocol) [33] and RTSP [10] for streaming video and audio data between Flash servers and Flash players. A special media server such as Flash Media Server [34] and Wowza [35] streams a series of video chunks and a Flash player consumes the content instantly without any local caching. The streaming server using dynamic RTMP [36] contains multiple bitrates for a single video content and allows the player to automatically change the bitrates during playback based on the network conditions. However, RTMP/RTSP streaming requires a special Flash-based media server and the licensing cost is expensive.

2.2.3 ABR streaming

Today's popular video streaming services such as YouTube, Netflix, HBO GO and BBC prefer ABR streaming technologies, having advantages of automatic quality switching and ease of delivery over HTTP. There are four popular ABR technologies: Apple's HLS [37], Microsoft's SS [16], Adobe's HDS [17] and 3GPP/MPEG DASH [38]. In ABR streaming, a video server contains multiple bitrates encoded for a single video content and each bitrate file is chopped into small segments. A segment size is measured in seconds (not bytes) and its length is typically between two and ten seconds. A manifest file contains the bitrate information such as the index of segments and their location. Before playback, an ABR player downloads the manifest file and it dynamically adapts the bitrate based on CPU availability and network conditions while a video is being played. In ABR streaming, viewers need to install various plug-ins depending on different types of devices (mobile and desktop), OSs and ABR streaming technologies. For example, Adobe's HDS requires Flash plug-ins. For Microsoft's SS, viewers need to install the Silverlight extensions in their web browsers. Apple's HLS supports all Apple's devices but may not properly work on old Android devices. Because of this, today's OTT streaming services such as YouTube and Netflix prefer HTML5 video that most web browsers (Safari 3+, Internet Explorer 9+, Firefox 3.5+, Chrome 3+ and Opera 10.5+) support (Table 2.4). HTML5 enables MPEG DASH native playback using Media Source Extensions (MSE) that allows JavaScript to deliver media streams for playback within web browsers [39]. Using MSE, viewers can dynamically change for a media stream without using plug-ins.

2.3 Understanding ABR streaming technologies

To better comprehend ABR streaming, we need to first understand how the video encoding for ABR streaming differs from the encoding for *progressive download*. To decode a video properly, a player needs to download an I-frame (intra frame), also known as a key frame, while a video is being played. In *progressive download*, an MPEG I-frame is inserted periodically (e.g., every ten seconds) into a single video file. In ABR streaming, a source video is encoded into multiple different files, each at different bitrates, and each such file is divided into a series of small segments. Each segment contains at least one key frame, preferably at the beginning of the segment. Depending on encoding tools, a single segment may have multiple key frames. For example, according to the technical note for Apple's HLS [40], a segment size is ten seconds and the key frame interval is three seconds.

A segment size is generally two and ten seconds long. Smaller segment sizes lead to decreased encoding efficiency in terms of GOP (group-of-pictures) frame size. Because of the higher number of segments, more I-frames are needed in the final encoding. On the other hand, longer segment sizes may cause frequent rebufferings under unreliable network conditions. For example, let's suppose that an ABR player is downloading a segment and the network is congested. The segment size is ten seconds and uses a single key frame. We note that an ABR player can switch bitrate only at an I frame. In this case, the player is unable to select lower bitrates until the requested segment has been downloaded in full. If the playout buffer is nearly empty, this may cause frequent rebufferings in the middle of the download. To prevent this, the player may use a timer; when the timer expires, it abandons current downloading segment and requests a low-quality segment. But, this may cause frequent bitrate changes if the timer length is too short. If the segment size is shorter (e.g., two or five seconds instead of ten seconds), the player can handle this situation better by switching to lower bitrates more quickly.

In ABR streaming, a player uses a set of heuristics to find the best available bitrate during playback. Based on our own analysis and technical overview of ABR streaming [41; 15], the following inputs are generally considered in the bitrate switching:

- real-time available network bandwidth and amount of video remaining in the playout buffer during playback;
- screen resolution and video rendering capabilities of viewers' devices;
- frame rate and viewers' interactive actions (e.g., resizing the browser window) during playback.

A player may experience frequent frame drops when a system is running multitasking that requires significant RAM and CPU usage. When a large number of frames is dropped,

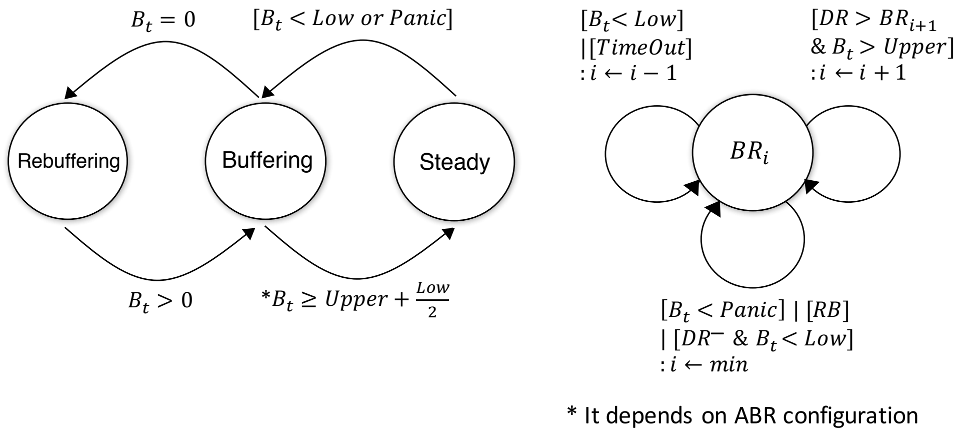

$B_t = 0$
$[B_t < Low\ or\ Panic]$
Rebuffering
Buffering
Steady
$B_t > 0$
$*B_t \geq Upper + \frac{Low}{2}$
$[B_t < Low]$ $|[TimeOut]$ $: i \leftarrow i - 1$
$[DR > BR_{i+1}$ $\&\ B_t > Upper]$ $: i \leftarrow i + 1$
BR_i
$[B_t < Panic]\ |\ [RB]$ $|\ [DR^- \ \&\ B_t < Low]$ $: i \leftarrow min$
* It depends on ABR configuration

Figure 2.1: Finite state machine (FSM) of state change and bitrate switching behavior of Microsoft's SS players

the player flushes its buffer and re-downloads the discarded segments at lower encoding rates to provide a good video quality.

While a video is being played, the state of video player can be `Buffering`, `Steady` or `Rebuffering`. We define `Buffering` state when an ABR player aggressively downloads video content into its playout buffer. The player requests the next segment right after it completely downloads the current segment (back-to-back HTTP requests) so that the buffer can be filled as quickly as possible. When the playout buffer is above a configured threshold, the player goes into `Steady` state. Instead of increasing the playout buffer level by downloading the segments back-to-back, the player in `Steady` state tries to keep the buffer full. In order to avoid buffer overflow, it requests a segment every segment duration. When the playout buffer is running low, the state will switch to `Buffering` again. `Rebuffering` is referred as buffer stalling or video buffering. It occurs when there is no video content available in the playout buffer during playback.

We examine the source code of the Silverlight extension, an ABR player for Microsoft's SS [42], and summarize the ABR player's state change and bitrate switching behaviors in Figure 2.1. The parameters are described as follows:

- $\mathtt{B}_t$ represents how much video content is currently left in the playout buffer (in seconds).

- $\texttt{BR}_\text{i}$ represents the video bitrate (in kb/s) selected by a player during playback, where $\texttt{BR}_\text{min} \leq \texttt{BR}_\text{i} \leq \texttt{BR}_\text{max}$. $\texttt{BR}_\text{min}$ indicates the minimum bitrate and $\texttt{BR}_\text{max}$ presents the maximum bitrate among the available bitrates of the video.

- `Panic, Low` and `Upper`: An ABR player takes into account three pre-defined thresholds when it changes state (`Buffering` or `Steady`) and picks the best available bitrate during playback. For example, when a buffer level becomes lower than the `Low` threshold, the bitrate will be downgraded by one step ($\texttt{BR}_\text{i} \rightarrow \texttt{BR}_\text{i-1}$). When the buffer level is lower than the `Panic` threshold, it directly drops down to the lowest bitrate ($\texttt{BR}_\text{i} \rightarrow \texttt{BR}_\text{min}$). When the buffer level is higher than the `Upper` threshold and the measured network throughput is larger than the next bitrate, then the current bitrate will be increased by one step ($\texttt{BR}_\text{i} \rightarrow \texttt{BR}_{\text{i}+1}$). The bitrate can be increased or decreased by multiple steps at a time (e.g., two or three steps) when the available bandwidth is changing rapidly. Before the player attempts to increase the bitrate, it waits for a certain amount of time such as three and five seconds to prevent frequent bitrate changes. All these settings depend on the ABR configuration.

- `TimeOut`: The timer is set to estimate network conditions. It activates when the elapsed time for downloading a requested segment is longer than the expected time. In such case, the bitrate is decreased for the next request.

- `DR` denotes current downloading data rate measured by a bandwidth estimator in an ABR player. $\texttt{DR}^-$ indicates that the available bandwidth in the network is decreasing. The time period required for the estimator to analyze the network conditions depends on the ABR configuration. For example, the estimator measures the average download throughput over the most recent three or five segments.

2.3.1 Network traffic behavior in ABR streaming

Based on extensive measurements, we first describe how videos are delivered to players using ABR streaming. During our experiments, we used the YouTube mobile application on various iOS and Android mobile devices over Wi-Fi networks. Figure 2.2 shows a simplified video traffic flow diagram between a video player and a YouTube's video content server.

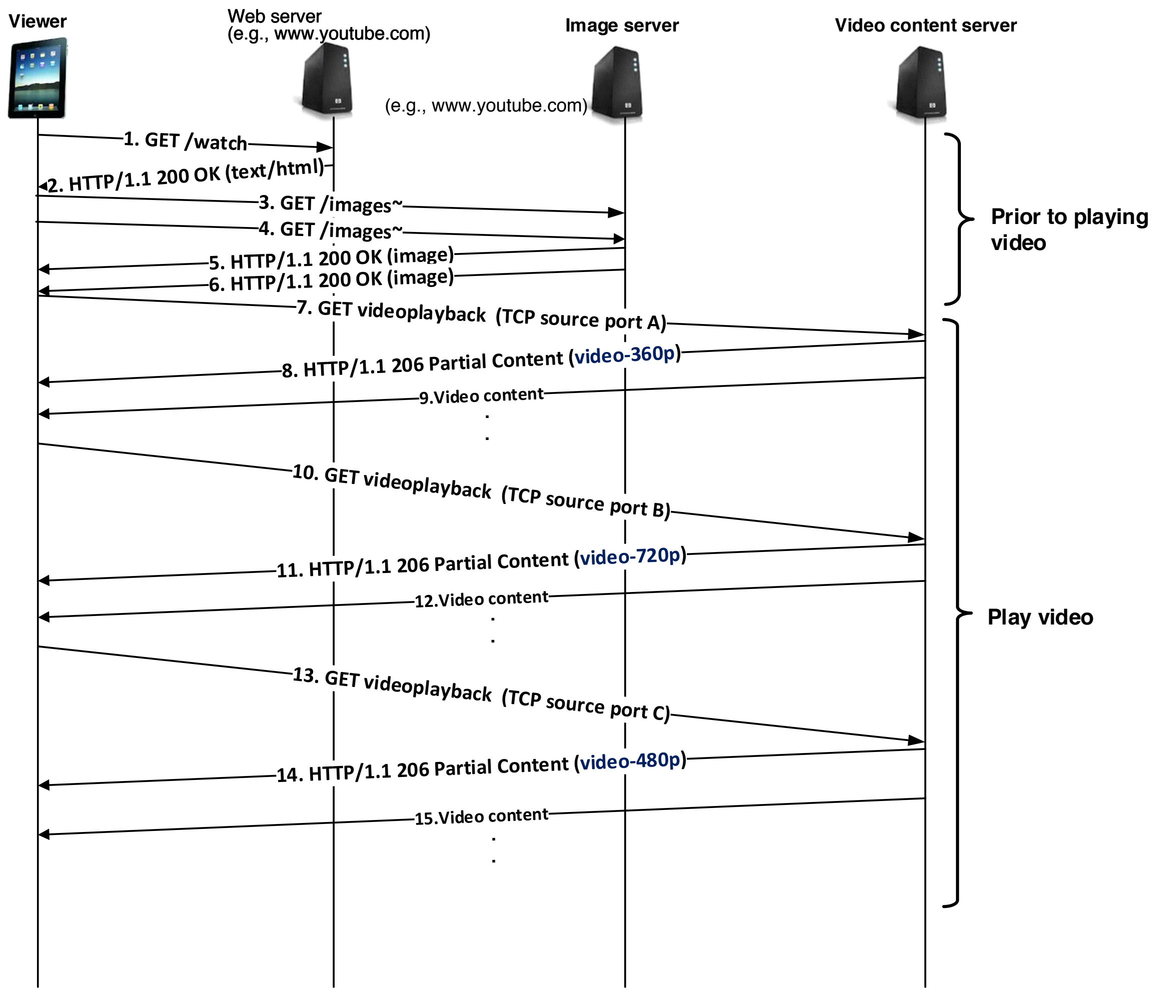

Viewer
Web server (e.g., www.youtube.com)
Image server
(e.g., www.youtube.com)
Video content server
1. GET /watch
2. HTTP/1.1 200 OK (text/html)
3. GET /images~
4. GET /images~
5. HTTP/1.1 200 OK (image)
6. HTTP/1.1 200 OK (image)
7. GET videoplayback (TCP source port A)
8. HTTP/1.1 206 Partial Content (video-360p)
9.Video content
10. GET videoplayback (TCP source port B)
11. HTTP/1.1 206 Partial Content (video-720p)
12.Video content
13. GET videoplayback (TCP source port C)
14. HTTP/1.1 206 Partial Content (video-480p)
15.Video content
Prior to playing video
Play video

Figure 2.2: YouTube video streaming platform

Before a player connects to a video content server, the player sends an HTTP GET message (step 1) to the web server that hosts the video streaming domain (e.g., `www.youtube.com`). The GET URL includes the unique identification of the requested video, the selected bitrate and user-agent information such as the OS and the video player version running on the device. Then the server responds to the player with the IP addresses of a selected video content server that contains the video content and a web content server where the player will download background images from. The response message also includes a video manifest file that contains the video information such as available bitrates and file names (step 2). The player transmits a set of HTTP GET messages (step 3 and 4) in parallel with downloading background images (step 5 and 6). The majority of these images consist of web page images and key frames of other videos related to the requested video. When the viewer clicks the player's play button, the player starts downloading the video content from the assigned video content server by sending an HTTP GET message (step 7, TCP source port A). Then, the content server streams the video with the requested bitrate. If the connected video content server does not contain the requested video, it responds with the HTTP 302 message to redirect the request to other available servers. When the player changes the bitrate, it sends a new HTTP GET message with a different TCP source port (step 10 and 13, TCP source ports B and C). The player does not change the TCP destination port (80). We note that the player does not establish multiple TCP connections at the same time to download the video content in parallel. Rather, it first terminates the opened TCP connection before establishing a new connection. We will analyze the downloading traffic behavior on different devices over different networks in the following section.

2.4 Understanding OTT video streaming platforms

We investigate the video streaming platforms of YouTube and Netflix. We conducted our experiments while playing the videos on mobile devices (iOS and Android) over wireless networks (Wi-Fi, 3G and 4G) under changing network conditions. As shown in Figure 2.3, we have designed the Java-based Video Streaming Packet Analyzer (VSPA) tool to capture and analyze TCP/IP and HTTP packets for video streaming between a viewer and a video content server. The tool uses `jpcap` [43], a network packet capture library, to analyze video packets in real time and tcpdump files (e.g., *.pcap) captured by `Wireshark` [44]. As shown in Figure 2.4, it supports the analysis of YouTube, Netflix, and Verizon's Redbox Instant. Table 2.1 shows the hardware specifications of the selected iOS and Android mobile devices that we used in our experiments.

2.4.1 An analysis of YouTube video streaming

YouTube uses Adobe's Flash, Apple's HLS and MPEG DASH to deliver H.264 videos. On January 27, 2015, YouTube announced that HTML5 will be the default playback method

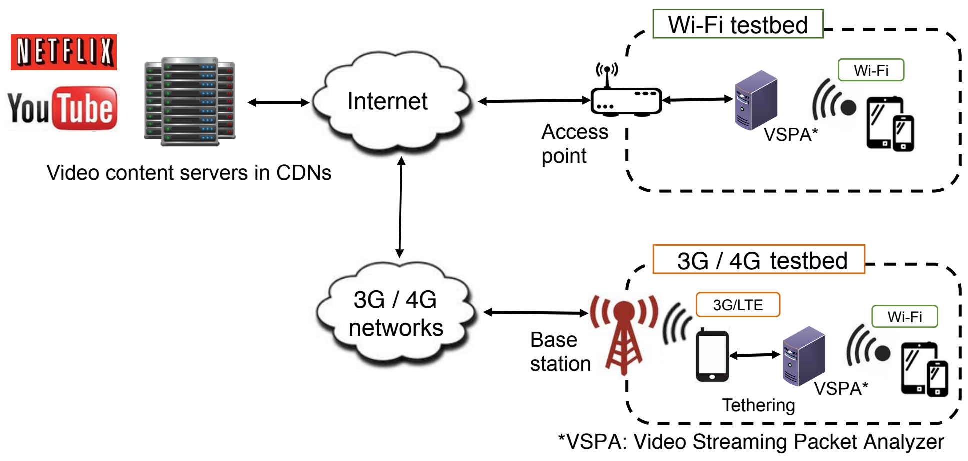


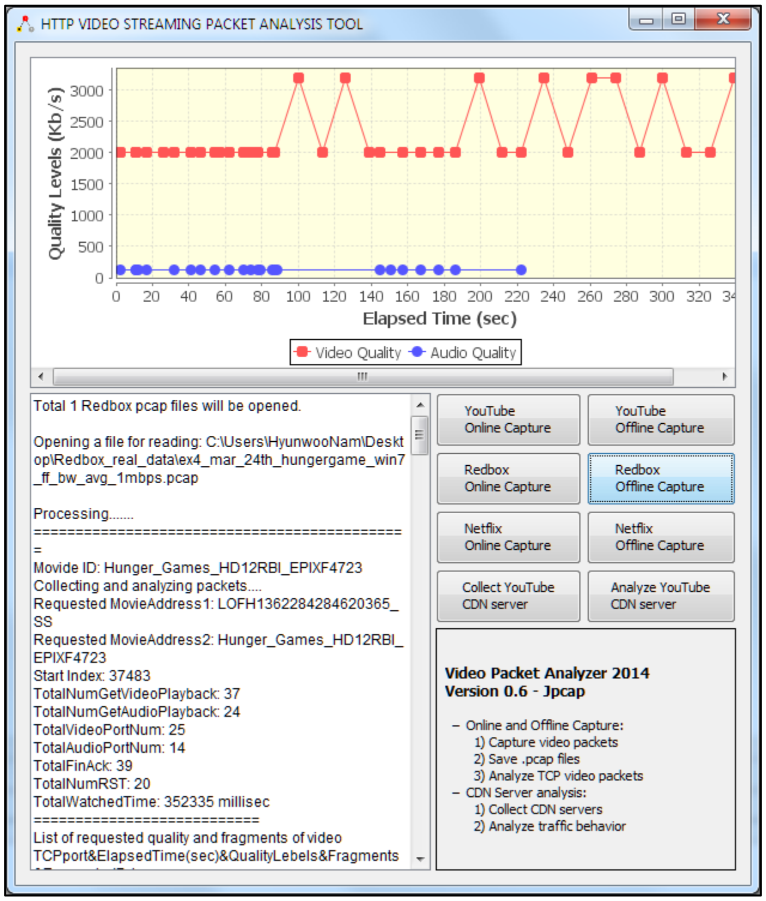
Figure 2.3: A testbed to analyze OTT video streaming services on mobile devices in wireless networks

Table 2.1: iOS and Android mobile devices used for measurements

Device	Operating system	Screen resolution	Memory
iPad 3	iOS 6.1.2	1920 × 1080	1024 MB
iPhone 4S	iOS 6.1.2	640 × 960	512 MB
iPhone 3G	iOS 4.1.2	320 × 480	128 MB
Nexus 7	Android 4.2.1	1280 × 800	1 GB
Nexus S 4G	Android 4.1.1	480 × 800	512 MB

and deprecate its Flash embeds and APIs [45]. For live TV streaming, YouTube also uses Apple's HLS for iOS and Mac OS, and the segment duration is 5 seconds [46].

We analyzed how YouTube streams videos to mobile devices over wireless networks. For our experiments, we played 450 videos on the mobile devices using its mobile application (not by web browsers) under varying network conditions. The videos were randomly selected in terms of the diversity in genre (animation, action movie, music video, live concert and sports), length (from five minutes to two hours) and video quality (high quality - HQ



```
Total 1 Redbox pcap files will be opened.

Opening a file for reading: C:\Users\HyunwooNam\Deskt
op\Redbox_real_data\ex4_mar_24th_hungergame_win7
_ff_bw_avg_1mbps.pcap

Processing.......
==============================================
=
Movide ID: Hunger_Games_HD12RBI_EPIXF4723
Collecting and analyzing packets....
Requested MovieAddress1: LOFH1362284284620365_
SS
Requested MovieAddress2: Hunger_Games_HD12RBI_
EPIXF4723
Start Index: 37483
TotalNumGetVideoPlayback: 37
TotalNumGetAudioPlayback: 24
TotalVideoPortNum: 25
TotalAudioPortNum: 14
TotalFinAck: 39
TotalNumRST: 20
TotalWatchedTime: 352335 millisec
============================
List of requested quality and fragments of video
TCPport&ElapsedTime(sec)&QualityLebels&Fragments
```

Video Packet Analyzer 2014
Version 0.6 - Jpcap

- Online and Offline Capture:
  1) Capture video packets
  2) Save .pcap files
  3) Analyze TCP video packets
- CDN Server analysis:
  1) Collect CDN servers
  2) Analyze traffic behavior

Figure 2.4: Java-based VSPA tool

360p and high definition - HD 720p). Our analysis of YouTube video streaming can be summarized as follows:

Sending plain HTTP GET messages to request a video: YouTube video player uses Apple's HLS for mobile devices. It uses a plain HTTP GET request including a header field that specifies the byte range of the video file (e.g., `Range: bytes=10000 - 50000`). The video content server then responds with an `HTTP 206 Partial Content` message (status code 206) and sends the requested range of the video. Unlike iOS, the video player for Android only defines the starting point in the HTTP header (e.g., `Range: bytes=10000 -`). Then, the video content server pushes the video from the requested starting point to the end of the video file.

Selecting different bitrates based on hardware capabilities: The YouTube video player selects different bitrates regardless of the OS and the radio interface, but it is closely correlated to hardware performance. For instance, the players on iPad 3, Nexus 7 and iPhone 4S chose HD (720p) bitrates when they requested the videos. On the other hand, the players on iPhone 3G and Nexus S 4G selected HQ (360p) bitrates due to the smaller size of the display screen (Table 2.1).

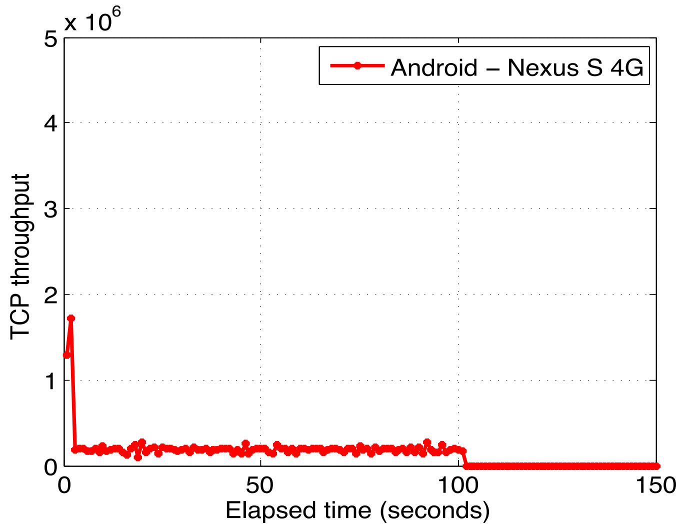
Performing fast start downloading: As shown in Figure 2.5, we observe a high TCP throughput at the beginning of playback. This is because a video player aggressively downloads video segments until its playout buffer becomes full (`Buffering` state). After then, it periodically downloads a segment every time there is a space in the buffer (`Steady` state).

Sending a series of HTTP GET messages while downloading a video: The YouTube video player repeats sending an HTTP GET message and receiving a video content segment while playing a video. We found that the player establishes a new TCP connection every time it sends a new HTTP GET message.

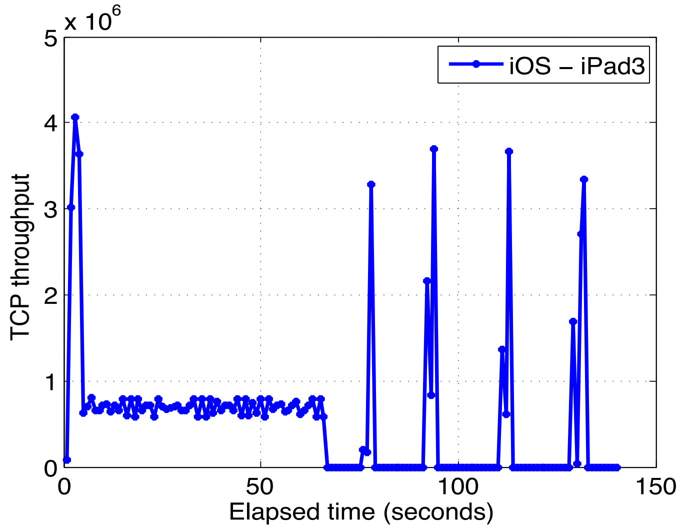
We address our findings below:

- **Impact of OS:** Our analysis shows that the YouTube traffic behavior varies depending on OSs. For example, our experimental results show that the YouTube video player for iOS typically sends more HTTP GET messages than the one for Android during a download. As investigated by Yao Liu et al. [47], one of the reasons is that the YouTube video player for iOS sends additional HTTP GET messages to request duplicate video content after the video has been played. The redundant video content is requested with the current playing bitrate and stored in the buffer for a possible later re-play by the viewers. We see this additional traffic on iOS devices (Figure 2.5b) but do not observe it on Android devices (Figure 2.5a).

- **Influence of network condition:** Using `netem` [48], we intentionally shaped the bandwidth in the network and added packet latency while playing the videos. We found that when the network experienced congestion, the video player sent more HTTP GET messages to change the bitrate, compared to stable network conditions.



(a) Android (Nexus S 4G) - HQ 360p



(b) iOS (iPad 3) - HD 720p

Figure 2.5: TCP throughput while playing the same YouTube video on an iPad 3 and a Nexus S 4G over Wi-Fi networks

2.4.2 An analysis of Netflix video streaming

We analyzed Netflix video packets while playing the videos on the mobile devices using its mobile application. According to Netflix tech blog [49], Netflix develops its own adaptive streaming technology. Today's Netflix supports MPEG-DASH for HTML5 enabled web browsers and Microsoft's SS for the Internet Explorer web browser [50]. It also supports Apple's HLS for their mobile applications. When the Netflix video player requests a video, it receives a manifest file containing the information of video bitrates over an SSL connection. Therefore, our VSPA tool cannot retrieve the information via packet capturing. According to Netflix [51], iOS and Android mobile devices support video streaming in 480p resolution, and the HD (720p and 1080p) videos can be viewed on devices that are capable of higher performance such as Sony PlayStation 3 and Apple TV. This indicates that the video bitrate of Netflix is also selected based on the hardware specification of the viewer's device.

We summarize our experimental results as follows:

Two separate TCP connections: Unlike YouTube, the Netflix video player simultaneously establishes two TCP connections with a video content server to download video and audio files in parallel. Netflix supports H.264 (AVC), VC-1, H.263 and H.265 (HEVC) for video and WMA, Dolby Digital, Dolby Digital Plus, AAC and Ogg Vorbis for audio [50].

Periodic HTTP GET messages: Our experiments show that the Netflix video player generates periodic HTTP GET messages to download a video. The HTTP header in each HTTP GET message specifies the short range of the video or audio files to be downloaded. For example, Figure 2.6 shows the TCP sequence numbers during playback measured from the iOS device. During the experiments, the player requested the video file every 10 seconds. Unlike iOS, the traffic behavior on Android is quite straightforward. The Netflix video player for Android requests the whole video and audio files at once. It sends a new HTTP GET message when it needs to change the bitrate.

2.4.3 Summary of key observations

Table 2.2 summarizes our analysis of YouTube and Netflix video streaming. Throughout our experimental results, we show how a video player downloads video content in ABR streaming by examining YouTube and Netflix. We conducted the same experiments via different wireless access networks such as 3G and 4G. For instance, we conducted the same experiments in 3G and 4G networks and did not find any differences from the Wi-Fi experimental results. The same bitrate of video was played on the mobile devices, and the video player downloaded the video in the same way regardless of the wireless interface. We

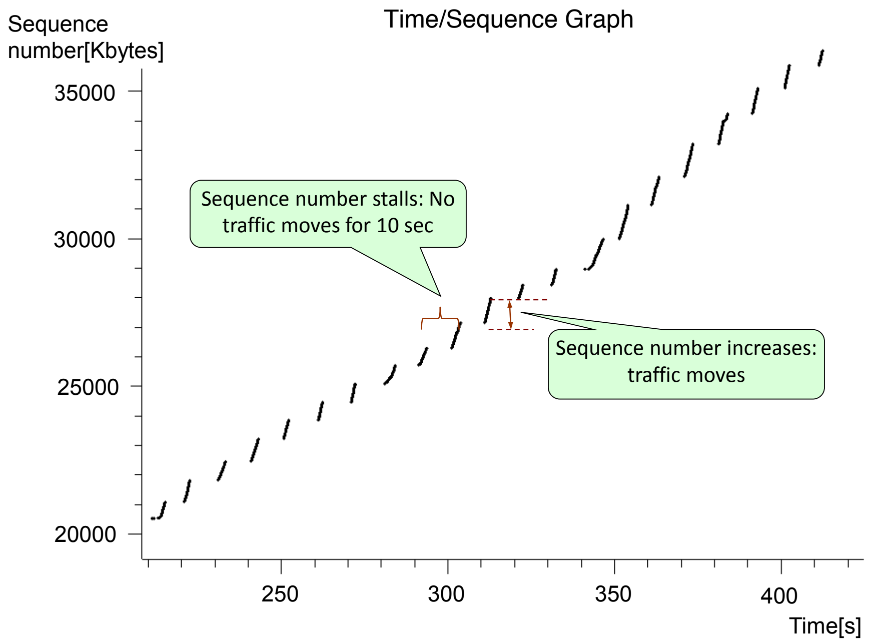


Figure 2.6: TCP sequence numbers with Netflix video trace over 4G networks

conclude that an ABR player sends a series of HTTP GET messages while playing a video and the traffic behavior of downloading videos varies depending on the OS, the hardware specification of the viewer's device and the network condition during playback.

2.5 Conclusions

The goal of this chapter is to understand how today's OTT video streaming work. We analyzed YouTube and Netflix video streaming on mobile devices (iOS and Android) over wireless networks (Wi-Fi, 3G and 4G) under varying network conditions. Through our experimental results, we prove that the video traffic behavior can vary depending on the network condition, the OS and the hardware specification of a viewer's device. The behavior is entirely based on the ABR heuristics implemented in an ABR player. According to our analysis, an ABR player dynamically changes bitrates based on various factors such as real-time available network bandwidth, amount of video remaining in the playout buffer and frame rate during playback, screen resolution and video rendering capabilities of the viewing device.

Table 2.2: Analysis of YouTube and Netflix video streaming on mobile devices[1]

	Operating systems	ABR	Num. of concurrent TCP connections while playing a video	Requested size of video per a TCP connection
YouTube	iOS	✓	1	A small segment of video requested in periodic messages
	Android	✓	1	Entire video file requested at once
Netflix	iOS	✓	2 (video and audio separate)	A small segment of video requested in periodic messages
	Android	✓	2 (video and audio separate)	Entire video file requested at once

[1] YouTube uses Apple's HLS, but it does not support Android devices before 4.0 (Gingerbread or Honeycomb). There are still inconsistencies and problems on Android 4.x and above. Netflix uses its own adaptive streaming technology for mobile devices. We note that these results are based on the experiments that we conducted in 2013 and 2014. Their streaming technologies and ABR heuristics may have changed.

Table 2.3: ABR technologies comparison chart[1]

	Adobe's HDS	Microsoft's SS	Apple's HLS	3GPP/MPEG DASH[2]
Video codec	H.264, VP6	H.264, VC-1	H.264	H.264 + others (agnostic)
Audio codec	AAC, MP3	AAC, WMA	AAC, MP3	AAC + others (agnostic)
Manifest file	.fmf	.ismc	.m3u8	.mpd
Package and segment format	.f4f, .fmf MP4 segments	.ismv MP4 segments	.ts MPEG-2 TS	.mp4 MP4 segments + MPEG-2 TS
File storage on server	Contiguous	Contiguous	Individual file per segment	Contiguous or individual files per segment
Segmentation and delivery	Adobe Interactive Server	Microsoft Internet Information Services	Multiple vendors. Standard HTTP or streaming servers	Multiple vendors. Standard HTTP or streaming servers
Playback	Flash, Air	Silverlight	Apple iOS, Quick Time X	3GPP-Rel 9 or MPEG clients
Protection	Flash Access	PlayReady	AES-128 encryption	Flexible (e.g., OMA [52] or UV [53])
Deployment on ordinary HTTP servers	No	No	Yes	Yes
HTML5 support	No	No	No	Yes
Typical segment duration	2 - 4 seconds	2 - 4 seconds	10 seconds	Flexible

[1] The content in the table is based on the report produced by the Internet Video Archive (IVA) group [54].

[2] HTML5 video players use MSE [39].

Table 2.4: The State of MPEG-DASH 2016[1]

Web browser	Operating system	MSE support	EME support[2]
Chrome 37+	Win 7+, OS X Yosemite+	mp4 AVC, webm VP9 [55]	CENC ClearKey [56], Widevine [57]
Chrome 37+	Android 4.4.4+	mp4 AVC, webm VP9	CENC ClearKey, Widevine
Edge	Win 10	mp4 AVC, webm VP9 (passthrough codec)	PlayReady
Firefox 42+	Win 7+, OS X Yosemite+	mp4 AVC	CENC ClearKey Adobe Primetime
Firefox 42+	Android 5.0+	mp4 AVC	-
Internet Explorer 11	Win 8.1	mp4 AVC	PlayReady
Internet Explorer 11	WinPhone 8.1	mp4 AVC	-
Opera 26+	Win 7+, OS X Maverick+	mp4 AVC, webm VP9	CENC ClearKey
Safari 8+	OS X Yosemite+	mp4 AVC, ts AVC	FairPlay (Netflix only)
Safari Mobile	iOS	-	-

[1] The content in the table is based on the report produced by the Streaming Media magazine [58].

[2] HTML5 web browsers use Encrypted Media Extensions (EME) to support digital rights management (DRM) for media copyright protection [59].

Part II

Intelligent Network Architecture for OTT Video Streaming

Chapter 3

Towards Dynamic Network Condition-Aware Video Server Selection over Wireless Networks

3.1 Introduction

Today's popular video content delivery systems deploy content delivery networks (CDNs). Video service providers such as YouTube and Netflix stream videos to viewers through their own CDNs or the CDNs provided by third parties such as Akamai [60] and Limelight [61]. When a viewer requests a video, a video service provider uses its own server selection algorithms in order to decide which video content server the viewer downloads the video from. The selection mechanisms and policies are designed for providing high availability, server load-balancing and minimizing the cost for delivering video contents to viewers [62; 63].

In this chapter, we analyze YouTube's video server selection algorithms. For an empirical study, we conducted our experiments while playing the videos on PCs and mobile devices (smartphones and tablets) over wireless networks (Wi-Fi and 3G) under varying network conditions. As we described in Figure 2.3, we use our VSPA tool to analyze video packets during our experiments. Through measurements, we found out that a viewer downloads a

YouTube video from the same video content server regardless of the hardware specification, the OSs and the video players running on the viewers' devices. Instead, the network attachment point regarding a viewer's public IP address is considered as a key factor for the video server selection algorithm of YouTube.

YouTube's video server selection mechanism takes into account various factors such as viewer proximity, server load and popularity of video content [63]. During our measurements, we found that YouTube selects one among three or four available content servers when a viewer requests a video. Most interestingly, we discovered that the video content server assigned by YouTube may provide less reliable streaming with lower TCP performance than other available content servers during playback. After carefully analyzing the measurements, we surmise that YouTube's DNS-based location awareness algorithms are causing this problem. YouTube delivery cloud typically assigns a video content server that is geographically close to a viewer [62; 63; 64]. However, the network conditions between a viewer and a video content server can be unstable, even though the server is located near the viewer.

We propose using the round-trip time (RTT) between an edge node of a wireless network and video content servers to discover a better video content server when a video is requested. The edge node can be an ISP router in a Wi-Fi network, a Radio Network Controller (RNC) node in a 3G network or a packet data network gateway (P-GW) in a 4G network. In our proposed architecture, the edge node caches IP addresses of video content servers while videos are delivered through the node. When a viewer requests a video, it first sorts the available content servers containing the requested video. Then, it measures TCP establishment time to each content server and selects one that has the shortest RTT. To show the feasibility of our approach, we compare the performance of YouTube streaming in our testbed setup (Figure 2.3). The evaluation shows that a video content server chosen by our proposed RTT-based video server selection algorithm outperforms the distance-based algorithms by providing more reliable streaming during playback. It showed better TCP performance 146 times out of 200 experiments (73%) over Wi-Fi networks and 162 times out of 200 experiments (81%) over 3G networks.

The remainder of the chapter is organized as follows. In Section 3.2, we elaborate on the

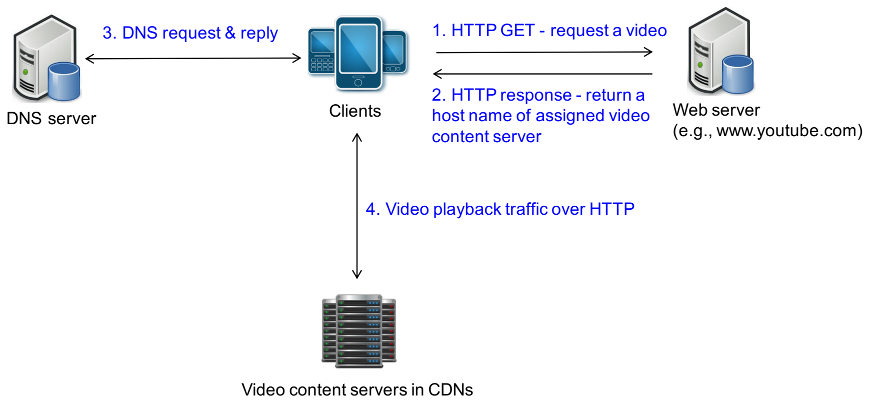

3. DNS request & reply
DNS server
Clients
1. HTTP GET - request a video
2. HTTP response - return a host name of assigned video content server
Web server
(e.g., www.youtube.com)
4. Video playback traffic over HTTP
Video content servers in CDNs

Figure 3.1: Video content server selection in OTT video streaming

analysis of YouTube's video server selection algorithms. In Section 3.3, we focus on finding problems that YouTube using location-based video server selection algorithms may assign a non-preferred video content server to a viewer. Our proposed solutions are described in Section 3.4. We evaluate our proposal in Section 3.5 and look at related work in Section 3.6. Finally, we summarize our conclusions in Section 3.7.

3.2 An analysis of YouTube video server selection algorithms

Figure 3.1 briefly describes a general CDN-based streaming platform. A video player running on a viewer's device sends an HTTP GET message (e.g., `http://www.youtube.com/watch?v=videoid`) to a web server that hosts the domain names such as `www.youtube.com` and `www.netflix.com`. The HTTP GET message contains a unique video identification and OS information of the device such as Windows, Mac OS X, Linux, iOS and Android. Then, the web server maps the video identification to the host name of the video content server that can stream the video based on their own server selection algorithms. It returns the host name (e.g., `v1.lscache1.c.youtube.com`) to the viewer. The host name is resolved to an IP address by the viewer via a DNS query to a DNS server. Finally, the viewer sends another HTTP GET message to the assigned video content server to download the video over HTTP or HTTPS. In our testbed setup (Figure 2.3), we monitored the network response times from the viewer's device to the web server and the selected video content server and found that the response times are different.

According to recent studies [62; 63; 64], YouTube typically allocates multiple IP addresses to a single host name, and picks one of video servers that is geographically close to a viewer. A viewer may be assigned to a farther location in order to avoid high traffic load on a particular video content server. In addition, YouTube uses an HTTP redirection mechanism to dynamically redirect a viewer's access to a non-busy video content server. In this case, the assigned video content server sends an HTTP 302 message asking to download the content from another server at the beginning of playback.

Using the testbed setups in Figure 2.3, our analysis takes into account various conditions such as requesting videos on different devices such as PCs and mobile devices (Table 2.1), using various applications running on diverse OSs (Windows, Mac OS X, Linux, iOS and Android) over Wi-Fi and 3G networks. During the measurements, we played hundreds of randomly selected YouTube videos from a diversity of genres (e.g., movie, music video, live concert and sports), popularity, length (from ten minutes to one hour) and video quality (SD and HD). Section 3.2.1 through Section 3.2.4 show our baseline analysis of YouTube's video server selection algorithms.

3.2.1 Requesting videos on different devices over Wi-Fi networks

We requested the 200 videos on the desktops and the selected mobile devices (Table 2.1) at the same time and from the same place over Wi-Fi networks under the same network condition. During the measurements, the devices were connected to the same Wi-Fi network attachment point. We used the YouTube mobile application for mobile devices (iOS and Android) and Chrome browsers for PCs where YouTube's Flash or HTML5 players are installed. We found that the addresses of video content servers chosen by YouTube remain the same regardless of the hardware specifications and the OSs running on viewers' devices.

3.2.2 Requesting videos on the same devices over Wi-Fi networks under varying network conditions

We played the 200 videos on the same desktop and mobile devices over the same Wi-Fi network under different network conditions: *stable* and *unstable*. In order to create unstable network conditions, we intentionally injected additional traffic to the network using a common network-testing tool, `Iperf` [65]. We also placed home networking devices that cause interference at 2.4 GHz, such as baby monitors and cordless phones, between the viewer and the Wi-Fi access point. The RTT between the viewers and the video content servers was 16 ms on average under stable network conditions, while it was 566 ms under fluctuating network conditions. Our experimental results show that network conditions between a viewer and a Wi-Fi access point do not influence the video server selection algorithms of YouTube. We found the same video IP addresses, regardless of the local network conditions.

3.2.3 Requesting videos on the same devices via different wireless network interfaces

We requested the 400 videos on the same desktops and mobile devices at the same time and from the same place over Wi-Fi and 3G networks. We observe that the viewer via a 3G network connected to a different video content server, compared to the viewer via a Wi-Fi network. However, it is difficult to confirm that YouTube considers the radio interfaces for the video server selection process. When we analyzed the user-agent information in the HTTP GET message, we did not find any differences between Wi-Fi and 3G networks. Instead, YouTube considers DNS-based location awareness [63]. When a video request occurs, the local DNS server operated by a network service provider asks the YouTube DNS server for the address of a video content server that the viewer downloads the video from. Based on the IP address of the DNS resolver, YouTube assigns a video content server that is geographically close to the viewer. Hence, the addresses of video content servers can be different, even though viewers accessing via Wi-Fi and 3G networks request an identical video from the same time and place. For example, we requested a thousand randomly selected YouTube videos on PCs and mobile devices via different network service providers (e.g., AT&T, Verizon, Columbia University and Time Warner Cable) around the Columbia University. We collected a total of 8,194 IP addresses of video content servers, and found that viewers accessed different sets of video content servers via different networks for the same video content.

3.2.4 Requesting videos on the same devices from the same place over Wi-Fi and 3G networks during a 24 hour period

A hundred YouTube videos were requested on the same desktops and mobile devices from the same place over Wi-Fi and 3G networks. We played the videos every ten minutes for 24 hours. For each video content server, we calculated the frequency of how many times it was selected by YouTube when the viewer requested the video. Our experimental results indicate that one or two video content servers are much more frequently selected than others that also contain the same content. In the experiments, for example, most of the video traffic (99.9% via Wi-Fi networks and 85.4% via 3G networks) came from one or two different video content servers for the same video content.

3.3 YouTube often assigns video content servers with long RTTs

In this section, we point out that YouTube's server selection algorithms frequently assign video content servers with long RTTs to viewers, even if there exist other servers that deliver the same video content with shorter RTTs. Based on our extensive measurements, we surmise that YouTube's location-aware video server selection algorithms cause this unwanted behavior. According to other studies [62; 63; 64], YouTube typically assigns a video content server that is geographically close to a viewer. The authors approximately found the locations of YouTube video content servers based on the RTT-based measurements and the analysis of DNS names of video content servers. However, our measurements show that the RTT between a viewer and a video content server may not be the shortest even if it is the geographically closest.

3.3.1 Finding locations of YouTube video content servers

We analyzed the geographical locations of YouTube video content servers. We first used the IP-to-location database [66]. The experimental results indicate that most of the 8,194 collected YouTube video content servers (more than 97%) were located in Mountain View, CA. However, as Torres et al. [63] discussed, the IP-to-location database does not return accurate YouTube server locations. The main reason is that the companies may be hiding the real locations of internal IPs for security reasons. They may also be on a large network (central and OSPF-routed) such as using the same ASN. To prove this, Torres et al. [63] measured the RTTs to the YouTube video content servers from several ISPs, and showed that there was a lot of variation, even though the database reported that the servers were located in the same place - Mountain View, California. Instead, according to Adhikari et al. [67], YouTube video content servers are distributed over more than 45 cities in 25 different countries around the world.

To avoid this problem, we used `traceroute` to estimate the locations of the last hop routers between the viewers and the YouTube network. Our measurements show that in most cases there was not much of a difference (less than 2 ms) between the RTTs to the last hop router and to the video content server from the viewer. Therefore, it is reasonable to assume that the locations of the actual YouTube's video content servers are geographically close to the last hop routers. During the experiments, we collected the addresses of video content servers while requesting a thousand random videos (popular and unpopular videos, regardless of genre) from four places in Manhattan, New York: the Columbia University, Upper West Side, and high population density areas such as Times Square and Penn. Station where we expect network congestion during busy hours. In the experiments, we found that most of the last hop routers of video content servers assigned by YouTube were located in New York (68.5%) and California (17%), and some routers were placed in Michigan (5.2%), Georgia (4.2%), Massachusetts (3%) and Florida (2.1%).

3.3.2 Measuring RTTs between video content servers and viewers

In order to examine the network conditions between viewers and video content servers assigned by YouTube, we measured RTTs to video content servers from viewers when they established TCP connections to the servers. We selected a hundred random videos, and conducted the following experiments:

- **Collecting addresses of video content servers:** For each video, we first obtained IP addresses of video content servers that contain the same video content. The IP addresses were collected while we requested a hundred random videos every five minutes

Table 3.1: A hundred YouTube videos were requested over Wi-Fi and 3G networks during busy hours (13:00 - 15:00 and 19:00 - 20:00 EST)

	Ratio (%) of being assigned to a non-preferred video server by YouTube			
Networks	Columbia University	Upper West Side	Times Square	Penn. Station
Wi-Fi	97 % (Columbia network)	97 % (Verizon)	96 % (Time Warner Cable)	70 % (Time Warner Cable)
3G	44 % (AT&T)	33 % (AT&T and Verizon)	73 % (AT&T and Verizon)	47 % (AT&T)

for three days on PCs from the four selected places in New York. We observed four or five unique IP addresses of video content servers for each video. They are mostly located in New York and California.

- **Comparing RTTs:** We measured the RTT between the viewer and the video content server assigned by YouTube. From a common place and time reference, we compared it with the RTTs between the viewer and other collected servers that contain the same video content. The RTTs were measured when the video player established TCP connections with the video content servers.

We define two terms to identify video content servers: *preferred* and *non-preferred*. The preferred video content server is a server that shows the shortest RTT among others that can deliver the same video content when the video is requested. The non-preferred video content servers are others, not defined as a preferred. We calculated the ratio of how many times YouTube provided a non-preferred video content server out of the total number of requests. Table 3.1 shows the experimental results. For Wi-Fi networks, we observed high ratios of being assigned to a non-preferred video server from each location. For 3G networks, we found that high popularity density areas such as Times Square and Penn. Station typically show such high ratios. During the measurements, the average and standard deviation of the RTT were 13 ms and 9 ms for Wi-Fi networks while they were 63 ms and 21 ms for 3G networks. As a result, the ratio proves that YouTube frequently assigns a non-preferred video content server to a viewer over Wi-Fi and 3G networks.

3.3.3 Video content servers with long RTTs to viewers may degrade video QoE

After carefully analyzing the measurements, we conjecture that YouTube's location-aware server selection algorithms causes non-preferred video content servers to be chosen. One may assume that this behavior is caused by YouTube's server load-balancing policies. If this assumption is true, we would have observed different video content servers being selected over time in our measurements. However, we found that the IP addresses of video content servers collected during the busy hours in Table 3.1 were almost the same with the IP addresses that were assigned during the non-busy hours. The last hop routers of non-preferred video content servers were located near New York. This suggests that YouTube's server selection algorithms take into account viewer-proximity more rather than the server load.

We note that this unwanted behavior may degrade video QoE. For example, Equation 3.1 indicates the TCP average throughput [68] in terms of packet loss and RTT. TCP flows with shorter RTTs gain a congestion window (CWND) advantage in the slow start phase. When a loss occurs, for example, the slow start begins from its initial CWND. With a short RTT, the CWND reaches the slow start threshold faster than a TCP flow with a longer RTT. Therefore, a viewer may often experience rebufferings when the RTT is long and the network conditions are fluctuating.

$$\mathrm{TCP}_{\text{avg. throughput}} = \frac{1.22}{\sqrt{\mathrm{P}_{\text{loss}}}} * \frac{\mathrm{MSS}}{\mathrm{RTT}} \tag{3.1}$$

Consequently, depending on network conditions in transit networks or server-side loads in CDNs, video servers that are farther away geographically from a viewer may temporarily provide a better viewing experience than video servers that are closer. We will elaborate on the experimental results in the following section.

3.4 RTT-based video server selection algorithms

The key idea is how to assign an appropriate video content server to a viewer when there is a video request. To achieve this, we could use a client-based mechanism that takes into

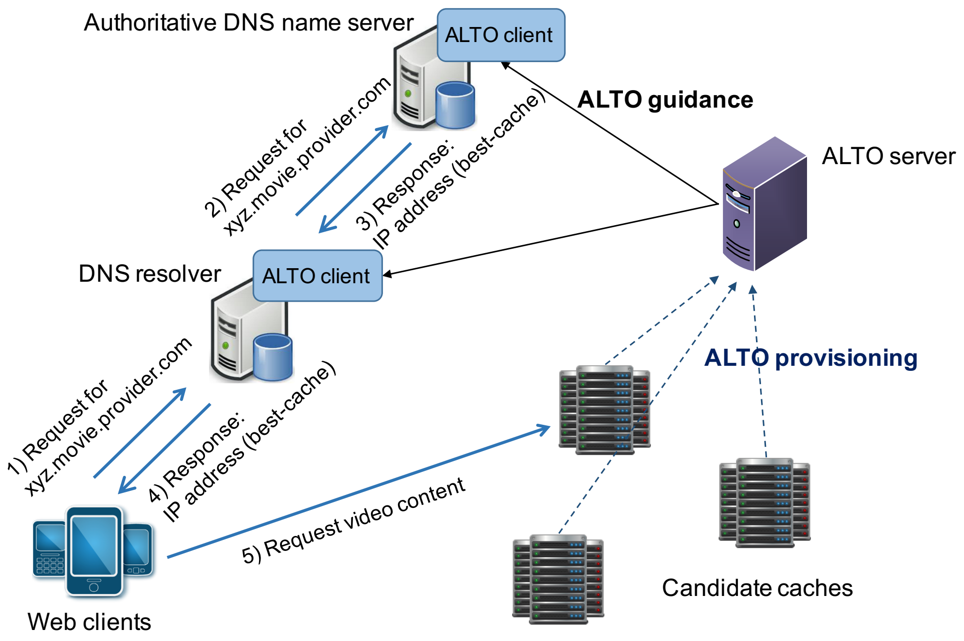

Authoritative DNS name server
ALTO client
ALTO guidance
ALTO server
2) Request for xyz.movie.provider.com
3) Response: IP address (best-cache)
DNS resolver
ALTO client
ALTO provisioning
1) Request for xyz.movie.provider.com
4) Response: IP address (best-cache)
5) Request video content
Candidate caches
Web clients

Figure 3.2: ALTO-guidance within CDN request routing (DNS example)

account the network conditions between the viewer and video content servers. According to Balachandran et al. [69], however, the viewer may not be able to effectively track the network conditions due to the lack of direct knowledge of access networks and up-link bandwidth constraints. Niven-Jenkins et al. [70] and Jan Seedorf [71] have introduced the use cases for Application-Layer Traffic Optimization (ALTO) with CDNs. For example, Figure 3.2 shows the request routing in CDN interconnection (CDNI) using ALTO [71]. In this scenario, a CDN provider hosts an ALTO server that provides network map and cost information such as geographical coverage, dynamic server load and packet loss rates to ALTO clients (DNS resolver and server in Figure 3.2). Based on the ALTO information and the IP address of the end user, the ALTO client chooses the best CDN server among the several candidates. In this way, the ALTO system can be used for the video streaming providers to enhance their video server selection in CDNs.

We propose to use a corresponding edge node of a wireless network in order to assist a viewer connecting to a better video content server. The edge node can be an ISP router in a Wi-Fi network, a RNC node in a 3G network and a P-GW in a 4G network. In our proposed architecture, the edge node performs like an ALTO client. Instead of receiving the network map and cost information from the ALTO servers operated by the ISPs or the CDN providers, however, it directly measures the RTT to each video content server for video server selection algorithms when a video is requested. The procedure is organized into two parts: *a*) Caching addresses of video content servers when viewers download videos through the edge node; *b*) Discovering a preferred video content server based on the comparison of RTTs between the edge node and the analyzed video content servers. The RTTs are measured while the edge node establishes TCP connection to the servers.

3.4.1 Caching addresses of video content servers

An edge node records addresses of video content servers when viewers watch videos through the edge node. It locally caches a hash-based database that maps the video identification to the addresses of assigned video content servers. The list of video content servers can be categorized based on time, locations of assigned video content servers and network conditions such as average TCP throughput, packet loss rates and RTTs which are measured while videos are delivered to the viewers.

3.4.2 Discovering a preferred video content server

RTTs between an edge node and collected video content servers measured when a video is requested are considered as key factors to find a preferred video content server. For example, Figure 3.3 shows a simplified mobile video streaming transaction for our proposed video server selection process.

1) When a video is requested, a web server returns the address of a selected video content server using its own server selection algorithms (step 1 and 2).

2) Before handing the address over to the viewer, the edge node examines if other video content servers cached on the list are able to provide more reliable streaming (step 3). It first searches a group of video content servers on the list that contain the requested video content. Secondly, it chooses a set of video content servers in the group that were recently used in the previous video sessions. Then, the edge node measures the current RTTs to the chosen video content servers and compares it with the one to the

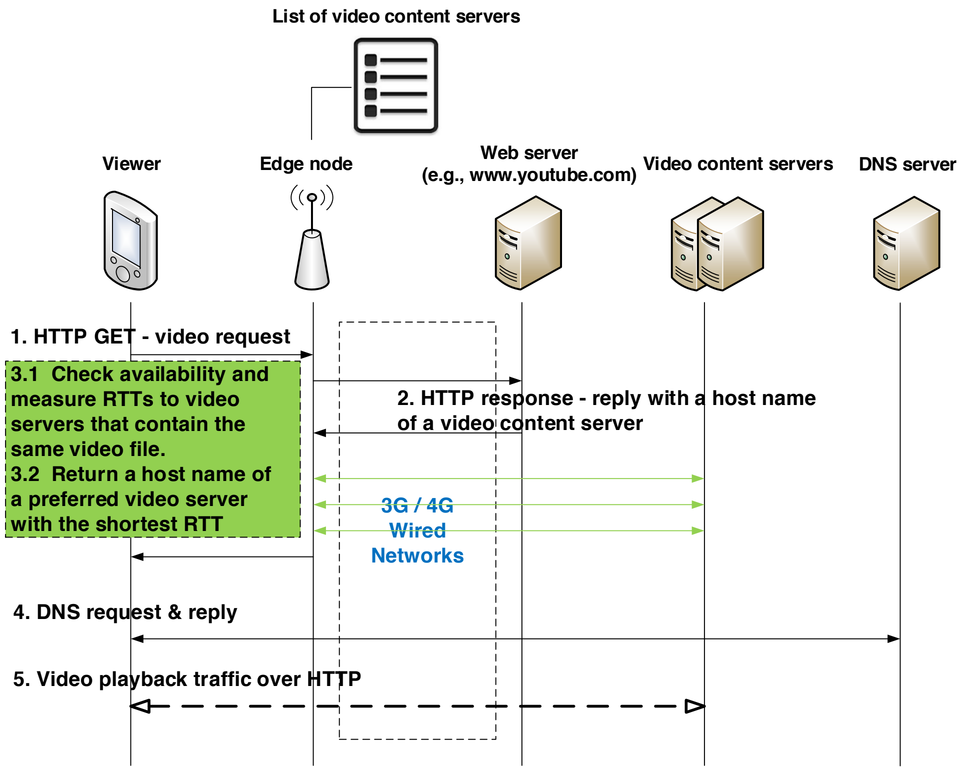

List of video content servers
Viewer
Edge node
Web server
(e.g., www.youtube.com)
Video content servers
DNS server
1. HTTP GET - video request
3.1 Check availability and measure RTTs to video servers that contain the same video file.
3.2 Return a host name of a preferred video server with the shortest RTT
2. HTTP response - reply with a host name of a video content server
3G / 4G
Wired
Networks
4. DNS request & reply
5. Video playback traffic over HTTP

Figure 3.3: Selecting a preferred video content server at a corresponding edge node

server assigned by the video service provider. The RTT can be measured during the TCP establishment. The edge node returns an address of a preferred video content server that shows the shortest RTT from the edge node.

3) Finally, the player resolves the host name and requests the video to the selected video content server (step 4 and 5).

Our proposed method causes a slightly longer start-up latency for a viewer to start downloading a video (2.1 seconds on average in our experiments), but the experimental results below show that it can enhance video QoE with higher TCP performance while playing a video.

3.5 Evaluation

To show the feasibility of our proposal, we have implemented our Video Streaming Packet Analyzer (VSPA) tool acting as an HTTP proxy server. We described the tool in Section 2.4. It manipulates HTTP headers to redirect a viewer's access to a preferred video content server chosen by our RTT-based video server selection algorithms. It can also analyze TCP traffic performance while downloading videos over cellular and Wi-Fi networks. The specific experimental setup is described below: We played a hundred random YouTube videos (360p resolution, video bitrate 0.5 Mb/s) on PCs and mobile devices from two different places (Columbia University and Penn. Station) over Wi-Fi and 3G networks. For each experiment, two viewers simultaneously requested the same video via the same network. Viewer A downloaded the video from the video content server selected by YouTube, and Viewer B accessed the video content server chosen by our RTT-based algorithm. Using `netem`, a networking emulation tool [72], we emulated network congestion between the viewers and VSPA. We injected additional 2 ms packet delay, 5% packet loss, 5% packet duplication and 5% packet re-ordering rates during the experiments.

Video QoE experiments - Considering network delivery issues, we monitor rebufferings to analyze the QoE of viewers. In order to measure this, we calculated the accumulated video bytes as time elapsed and compared it with the required downloading data rate. Figure 3.4 shows one of our video QoE experimental results over Wi-Fi networks. In the experiment, the length of the video was 253 seconds. The dotted blue line indicates the required data rate to play the video without any viewing interruption. During the experiment, Viewer B (green line) took only 222 seconds to complete downloading the entire video content while Viewer A (red line) took 290 seconds. Since the red line for Viewer A falls below the blue line, this indicates that Viewer A experienced rebufferings during playback. We conducted the same experiment twenty times over Wi-Fi networks from each place. Viewer A experienced more rebufferings than Viewer B thirteen times out of twenty experiments. Viewer B experienced a slightly higher number of rebufferings two times and the same number of rebufferings during the rest of the experiments. However, Viewer B with our dynamic server selection algorithms experienced less rebuffering time compared to Viewer A, by an average of 15 seconds.

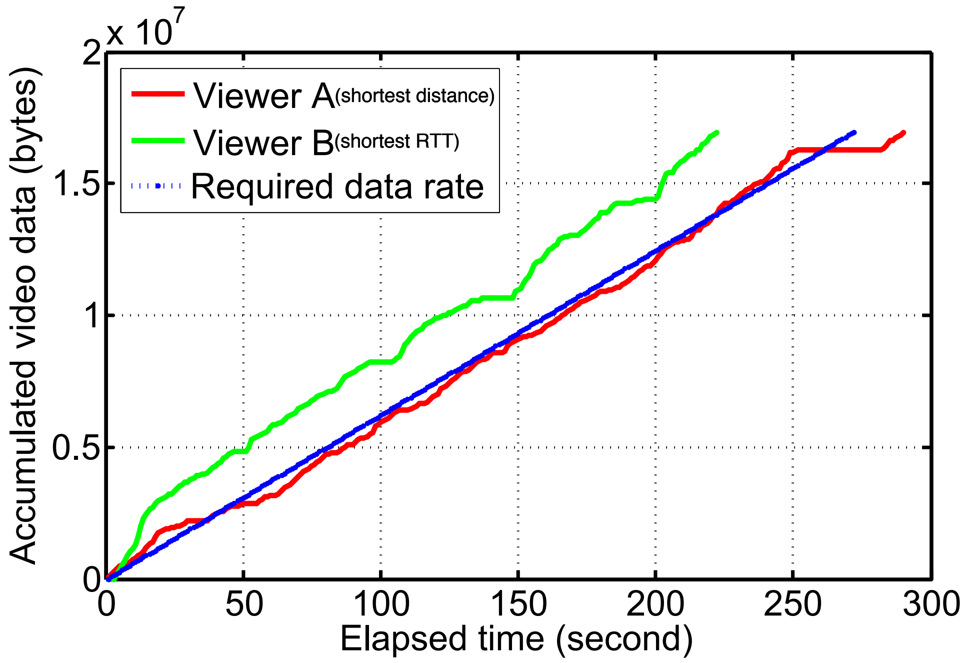


Figure 3.4: Monitoring downloaded video bytes while playing a sample YouTube video on two PCs over a Wi-Fi network in the HTTP proxy server-based testbed (one download)

Comparing packet inter-arrival times while downloading a video - We measured video packet inter-arrival times until the same video content was completely downloaded from the two video content servers. Figure 3.5 compares the CDFs: the solid line represents the results between Viewer A and the video content server selected by YouTube, and the dotted line indicates the results between Viewer B and the video content server that showed the shortest RTT when the video was requested. This experiment was conducted over 3G networks on the Columbia University during the time period of 13:00 and 13:30 EST. The experimental results indicate that our RTT-based video server selection algorithm provided shorter inter-arrival times while downloading the videos compared to YouTube's video server selection algorithms.

Comparing TCP throughput - We further analyzed TCP performance from the two different places (Columbia University and Penn. Station). From each place, we requested one hundred videos both on PCs and mobile devices over Wi-Fi and 3G networks during afternoon hours (12:00-17:00). We captured TCP/IP and HTTP packets using the VSPA,

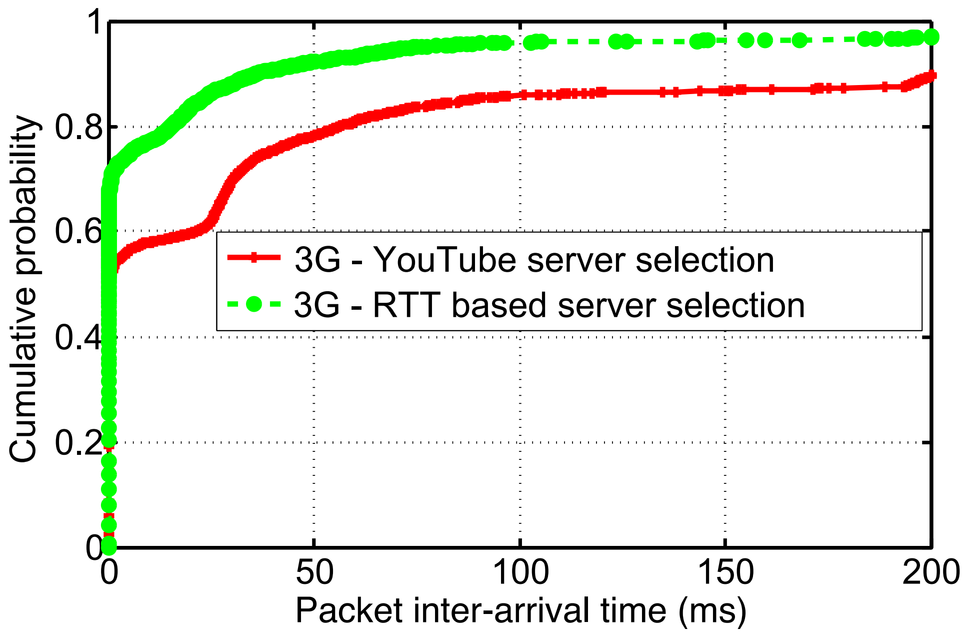


Figure 3.5: CDFs of video packet inter-arrival times measured while downloading a video over a 3G network in the HTTP proxy server-based testbed

and analyzed the dump files using a TCP trace tool [73]. Our proposed algorithm showed the higher TCP throughput 146 times out of 200 experiments (73%) over Wi-Fi networks and 162 times out of 200 experiments (81%) over 3G networks. During the rest of the experiments, our solution shows similar or slightly lower performance.

3.6 Related work

Several researchers have investigated video server selection algorithms that consider the geographical locations of video servers. Torres et al. [63] found that a variety of factors such as load-balancing, variations across DNS servers within a network and video popularity may affect the video content server selection process for YouTube. Adhikari et al. [64; 74; 67; 75] investigated the YouTube infrastructure by collecting video traces at ISP backbone networks. Analyzing the YouTube video distribution architecture, they found that YouTube deploys a large number of video caching servers that vary in size and geographical locations in order to reduce cost and improve the end-user performance. Saxena et al. [76] analyzed how three video service providers (YouTube, Dailymotion and Metacafe) distribute their video streaming services and how the server selection takes into account viewers' geographical locations and video characteristics such as age and popularity.

Our approach differs from the prior work in two aspects: Noticeably, in many cases, we found that YouTube assigns a non-optimal video content server to a viewer. We suggest that edge nodes of wireless networks should directly determine the best available video content servers based on network conditions. We propose using the TCP RTT when videos are being requested. The challenges of our approach is to install additional functions such as building a database to store video information and measuring RTTs for all video connections on the edge nodes. This causes the scalability concerns for large networks. In addition, our approach may cause a long start-up delay depending on the network conditions when the video is requested. This may degrade QoE of viewers. We will discuss the impact of start-up latency on video QoE in Chapter 7.

As we described in Section 3.4, ALTO [71] can be used for the dynamic content server selection in CDNs. It implements several ALTO clients and ALTO servers to find the best content server based on various network map and cost information such as server load and packet loss rates. Our future work is to empirically compare the performance between ALTO-based platform and our approach for building a dynamic CDN server selection mechanism for OTT video streaming services.

3.7 Conclusions

We have analyzed the YouTube video server selection algorithm. We proved that the geographical location based on the network attachment point of a viewer is considered important when YouTube assigns a video content server to a viewer. Our proposed dynamic network conditions-aware approach that considers current RTTs to find a better video content server achieves higher TCP performance than distance-based server selection algorithms, which improves QoE of viewers during playback.

Chapter 4

Towards Dynamic QoS-aware OTT Video Streaming

4.1 Introduction

ABR streaming is designed to provide viewers with the positive QoE for given network conditions. However, an ABR player may select an inappropriate bitrate during playback due to the lack of direct knowledge of access network performance, frequent user mobility and rapidly changing channel conditions. To resolve this issue, we may periodically estimate network capacity between a client and a video content server or a network access point, but bandwidth constraints often limit the frequency of end-to-end feedback [69]. Alternatively, SDN may enable video service providers to partially provision network resource in collaboration with network operators. However, the SDN infrastructure in WANs has not been widely deployed [77].

Providing a seamless viewing experience is important for network operators to increase the number of subscribers in their networks. To provide reliable streaming, we propose to build a dynamic QoS-aware video streaming platform in a 4G network. From the viewpoint of network operators, our proposed platform does not require any technical support from video service providers. In this chapter, we first attempt to characterize the inefficiency of today's ABR streaming approach and then improve the video content delivery in a 4G network.

During our experiments, we observed that the ABR players frequently discard a large number of successfully downloaded video packets. This unwanted behavior occurs when the video player changes the quality level in unstable network conditions. For example, when a timeout occurs during a download (Figure 2.1), the player terminates the open TCP connection and requests lower bitrate by establishing a new TCP connection. The packets in flight coming through the closed TCP connection will be discarded at the player. In addition, this happens when a viewer abandons the video during playback. The video player also discards some video packets when a viewer moves the playback slide bar before completely downloading a requested video. Our analysis shows that the average video packet discard is 10.1%, and that the discard may exceed 35% of the complete content. Needless to say, this behavior consumes network resources and causes additional mobile data usage paid by clients.

Our solution is to address this issue using the packet data network gateway (P-GW) in a 4G network. The P-GW provides connectivity from a 4G user to external packet data networks such as the Internet. It is responsible for performing policy enforcement, user IP-address allocation, packet filtering and charging. In our proposed platform, the selective packet discarding mechanism in P-GW drops the potentially wasted video content in advance before it is delivered to a viewer. It prevents unnecessary mobile data usage paid by viewers and waste of the limited network resource over the air interface. In order to improve video QoE for viewers, our proposed QoS rules in P-GW are designed to dynamically manipulate QoS parameters such as Aggregate Maximum Bit Rate (AMBR) that controls the maximum possible data rates, based on network conditions between a viewer and an Evolved Node B (eNodeB), a base station in a 4G network. By throttling TCP throughput, our QoS rules assist a video player to choose a proper bitrate under fluctuating network conditions. Our contributions can be summarized as follows:

1) We discover the underlying causes of video packet discard on HTTP-based mobile video streaming (Section 4.2).

2) Using network operator resources, we improve the existing OTT video delivery system. Our evaluation shows up to 21% saving down-link bandwidth over the air interface,

and the proposed mechanism enhances the video watching experience by reducing rebufferings during playback (Section 4.3 and 4.4).

The remainder of the chapter is organized as follows. In Section 4.2, we focus on finding problems that cause the discarding of video data, and our proposed solutions are described in Section 4.3. We evaluate our proposal in Section 4.4 and look at related work in Section 4.5. Our discussion is addressed in Section 4.6. Finally, we summarize our conclusions in Section 4.7.

4.2 Poorly designed video players waste network bandwidth

Our analysis on ABR streaming indicates that an ABR player sends a sequence of HTTP GET messages while playing a video (Chapter 2). During the measurements, we discovered that a large number of video packets can be discarded during a download. This unwanted behavior occurs when a video player terminates an open TCP connection before completely downloading the requested video segment.

As an example, Figure 4.1 shows a simplified video traffic flow diagram between a viewer and a YouTube video content server. When a viewer plays a video, the video player sends an HTTP GET message (step 7, TCP source port A) to download the video file. The video packets (step 9 and 10) are successfully delivered to the video player. However, before receiving the next video packet (step 12), the video player closes the port number A and transmits another HTTP GET message via a new TCP connection with the source port number B (step 11). We often find this behavior when a timeout occurs during a download (Figure 2.1). In the meantime, those video packets that were sent by the video content server prior to noticing the termination continue to arrive at the TCP port A. A TCP RST packet is sent to the video content server each time the video player receives a video packet via the terminated TCP connection. Consequently, the video packets (step 12 to 14) are discarded and not stored in the playout buffer. While experimenting, we found out that the following three cases cause this problem:

1) When a timeout occurs during a download, an ABR player terminates the open TCP connection and sends a new HTTP GET message. The player may receive the seg-

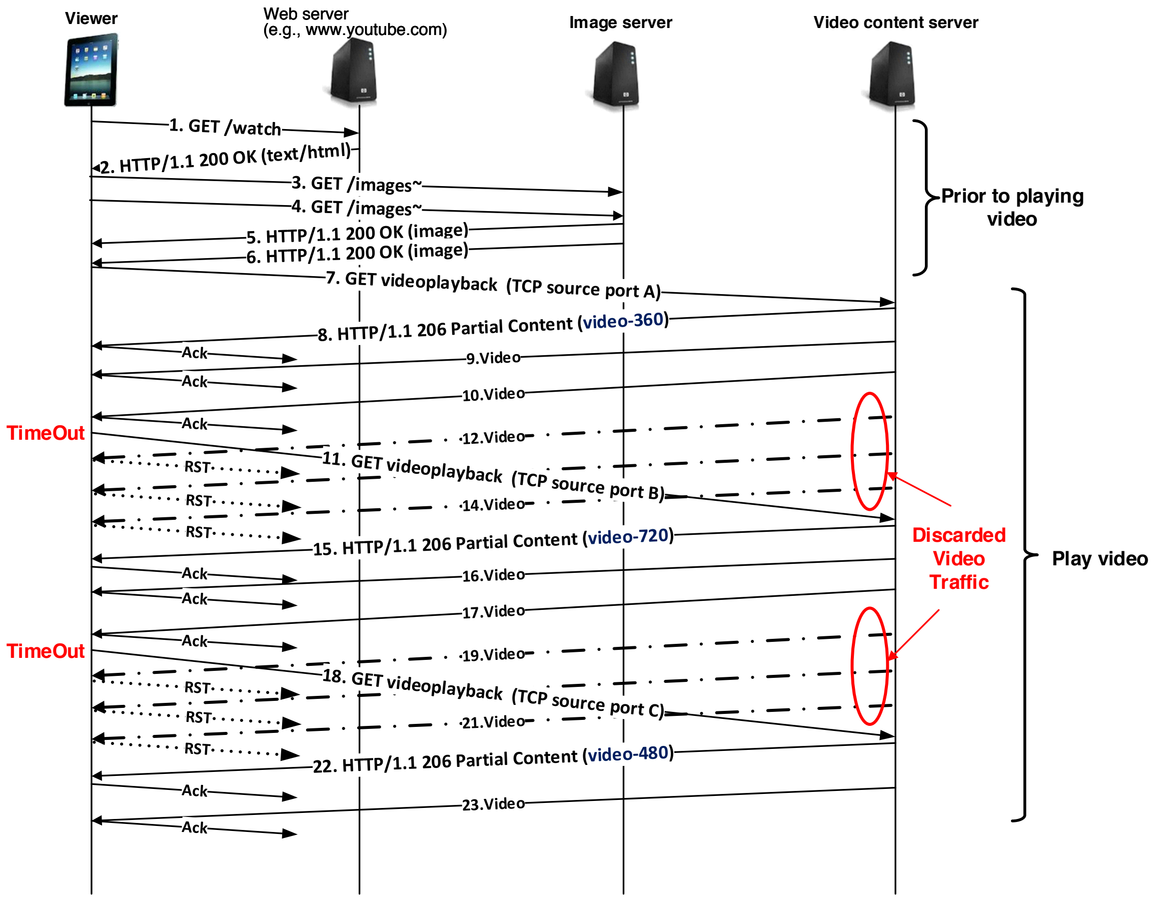

Viewer
Web server (e.g., www.youtube.com)
Image server
Video content server
1. GET /watch
2. HTTP/1.1 200 OK (text/html)
3. GET /images~
4. GET /images~
5. HTTP/1.1 200 OK (image)
6. HTTP/1.1 200 OK (image)
7. GET videoplayback (TCP source port A)
8. HTTP/1.1 206 Partial Content (video-360)
Ack
9.Video
10.Video
TimeOut
11. GET videoplayback (TCP source port B)
12.Video
RST
14.Video
15. HTTP/1.1 206 Partial Content (video-720)
16.Video
17.Video
18. GET videoplayback (TCP source port C)
19.Video
21.Video
22. HTTP/1.1 206 Partial Content (video-480)
23.Video
Prior to playing video
Play video
Discarded Video Traffic

Figure 4.1: Video packet discard occurs when a timeout occurs during a download

ments encoded at the previously requested bitrate before the newly requested GET message arrives at the video content server. When such events occur, the video packets through the terminated TCP port will be discarded.

2) When a viewer moves a playback slide bar while playing a video, an ABR player immediately terminates the open TCP connection and sends a new HTTP GET message that contains a newly requested range of bytes of the video. The player will no longer accept the video packets that continue to arrive at the closed TCP connection.

3) When a viewer abandons the video during a download, the requested video segments will be discarded at the video player.

4.2.1 Calculating discard ratio

$$\text{Discard ratio} = 1 - \frac{\text{Goodput}}{\text{Total throughput}} \tag{4.1}$$

Experimental setups - Using Equation 4.1, we calculated the discarded video traffic ratio. The specific experimental setups are:

- One hundred YouTube and Netflix videos were played on mobile devices via Wi-Fi, 3G and 4G networks in our testbed (Figure 2.3). During the experiments, the video players selected a bitrate of either among HQ (360p) or HD (720p) based on their own ABR heuristics.

- Using `netem` [72], we artificially manipulated packet delay (avg. 50 ms ± 10 ms variance distribution), packet loss rate (avg. 5%), packet duplication rate (avg. 3%), packet corruption rate (avg. 3%) and packet re-ordering rate (avg. 5%) between mobile devices and our VSPA tool. Using `iperf` [65], we also generated a heavy TCP traffic to the same network to overload the network.

- We manually moved the slide bars of the video players during a download. For each experiment, we moved the bar ten times to the unbuffered point and abandoned the videos during playback.

Table 4.1 shows the experimental results. We did not experiment with the iPhone 3G running iOS 4.1.2 for Netflix because Netflix only supports iOS 5 or later. Our analysis shows that for YouTube, Android suffers form a lower discard ratio than iOS. That is mainly because the YouTube video player for iOS sends more HTTP GET messages with new TCP connections than the video player for Android, in order to download the redundant video content for potential re-play activities 2.4.1. For Netflix, iOS shows a lower discard ratios compared to Android. As we stated before, the video player for iOS periodically requests a small chunk of video. On the other hand, the player for Android requests the entire video file at one go. When it changes the quality level, it requests the entire file (from the current playing point to the end) for the newly assigned bitrate. This causes a large number of packets in flight that are discarded at the player. In addition, we found out that the discard ratio is affected by hardware performance of a viewer's device. For Netflix,

Table 4.1: Average and standard deviation of discard ratio (%) while playing YouTube and Netflix videos on mobile devices over Wi-Fi, 3G and 4G networks under fluctuating network conditions

Device	OS version	YouTube	Netflix
iPad 3	iOS 6.1.2	11.77 % (1.23)	0.13 % (0.09)
iPhone 4S	iOS 6.1.2	11.25 % (1.22)	0.5 % (0.48)
iPhone 3G	iOS 4.1.2	13.01 % (9.02)	Not available
Nexus 7	Android 4.1.2	1.79 % (0.45)	11.11 % (9.1)
Nexus S 4G	Android 4.1.1	9.23 % (1.81)	1.413 % (0.68)

Nexus S 4G shows lower discard ratios compared to the Nexus 7. That is because only the low video bitrates (smaller size than high bitrate) are selected on the Nexus S due to the small size of the display screen.

4.2.2 Summary of key observations

Through our experimental results, our found out that discard ratio is closely related to the performance of viewers' devices and the network conditions while playing videos. For instance, the large number of video packets which are sent by the video content server before receiving a TCP RST packet will be discarded when the RTT between the server and the viewer is long and the receiver TCP window size is large (Figure 4.2). As we described before, the more HTTP GET messages via new TCP connections an ABR player sends, the more delivered video packets is likely to be discarded. Also, if the network is congested, the HTTP GET messages may get lost or retransmitted, which will further increase the discard ratio.

4.3 Improving OTT video content delivery in 4G networks

Wireless network resources such as radio spectrum and backhaul transport between the base station and the core network are limited and expensive. As described in Section 4.2, current OTT video players may waste a large amount of network resources and cause additional

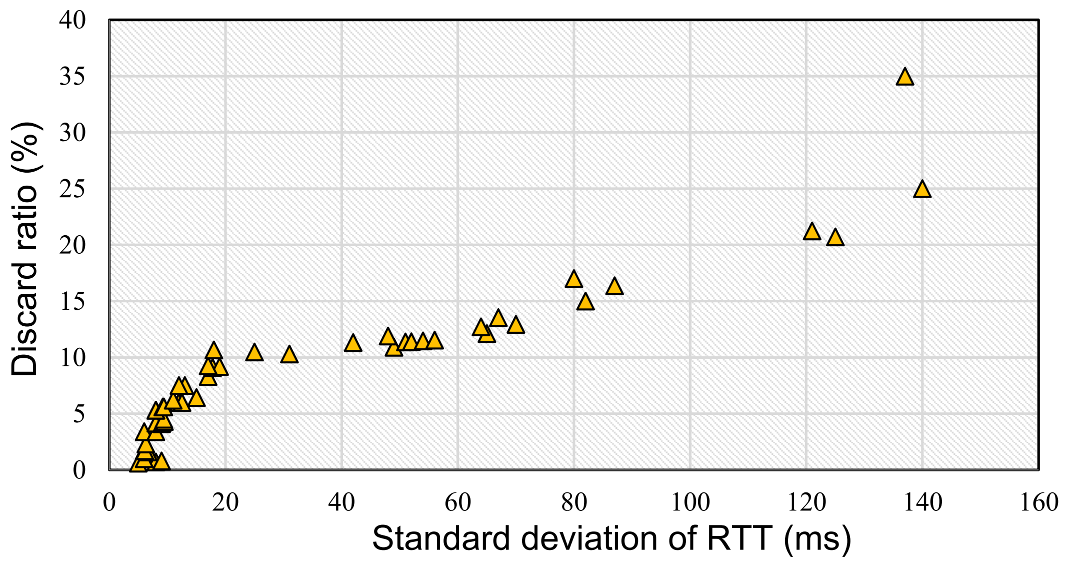


Figure 4.2: RTT vs. discard ratio (%)

mobile data usage paid by the clients. In order to resolve the inefficiency, we propose a dynamic QoS-aware video streaming in 4G networks.

4.3.1 QoS in 4G networks

We first describe a brief background of how QoS in 4G networks would be implemented on bearers[1] between a client and P-GW. The QoS level determines how an IP packet flow is handled at eNodeB when it experiences congestion, influencing scheduling policy, queue management and rate shaping. There are two types of bearers; the dedicated bearer and the default bearer. A default bearer is established when a client is connected to a 4G network, and several dedicated bearers can be added when it needs QoS-enabled services such as VoIP and video streaming.

There are two types of dedicated bearers; the Guaranteed Bit Rate (GBR) type and the non-Guaranteed Bit Rate (non-GBR) type. In a GBR mode, it provides minimum and maximum guaranteed data rate per an Evolved Packet System (EPS) bearer using GBR and Maximum Bit Rate (MBR) parameters. In a non-GBR mode, on the other hand, a bearer provides a best-effort packet delivery. Even though non-GBR bearers do not provide

[1]A bearer means a virtual pipe line connecting two or more points in a 4G network.

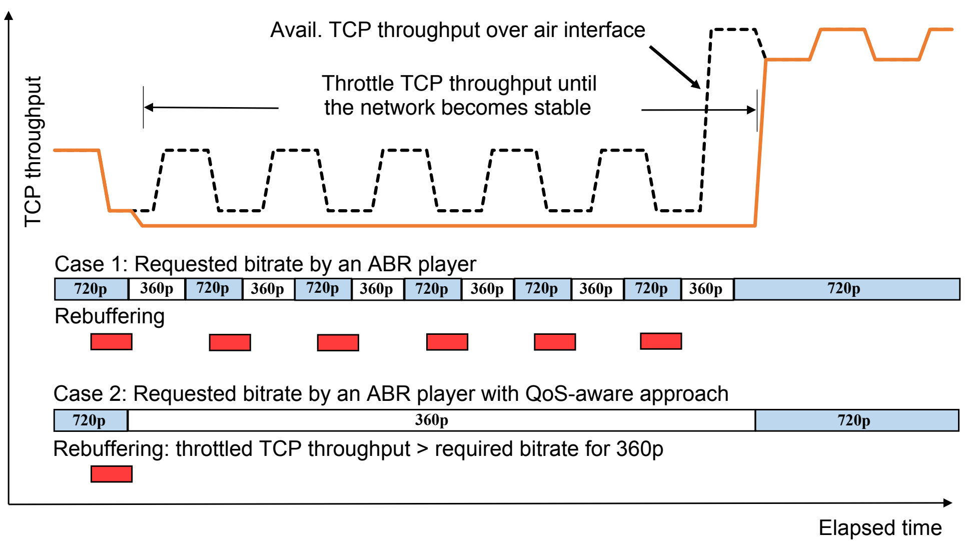

Avail. TCP throughput over air interface
Throttle TCP throughput until the network becomes stable
TCP throughput
Case 1: Requested bitrate by an ABR player
720p
360p
Rebuffering
Case 2: Requested bitrate by an ABR player with QoS-aware approach
Rebuffering: throttled TCP throughput > required bitrate for 360p
Elapsed time

Figure 4.3: The impact of controlling TCP throughput on video QoE

a guaranteed data rate, LTE still enables managing QoS by using the A-AMBR and the UE-AMBR parameters:

- A-AMBR: This indicates the maximum possible data rate for all best effort services for all clients connected to a specific access point name (APN).

- UE-AMBR: This represents the maximum possible data rate for all of best effort services for a particular client. It prevents a client from taking all the available bandwidth from the other 4G clients over the same air interface.

The 3GPP standards have defined nine QoS class identifiers (QCIs) in total which are characterized by priority, packet delay budget and packet error loss rate. According to the standardized QCIs in 4G networks [78], buffered video streaming is assigned to QCI 6, 8 and 9, which indicate non-GBR type, 300 ms packet delay tolerance and 10^{-6} acceptable packet error loss rate.

4.3.2 Dynamic QoS-aware video content delivery in 4G networks

When network conditions are fluctuating, ABR players may repeatedly switch bitrates while playing a video. This may result in frequent rebufferings and a large amount of packet discard, as shown in Figure 4.3 (Case 1). To solve this problem, we propose differentiated QoS

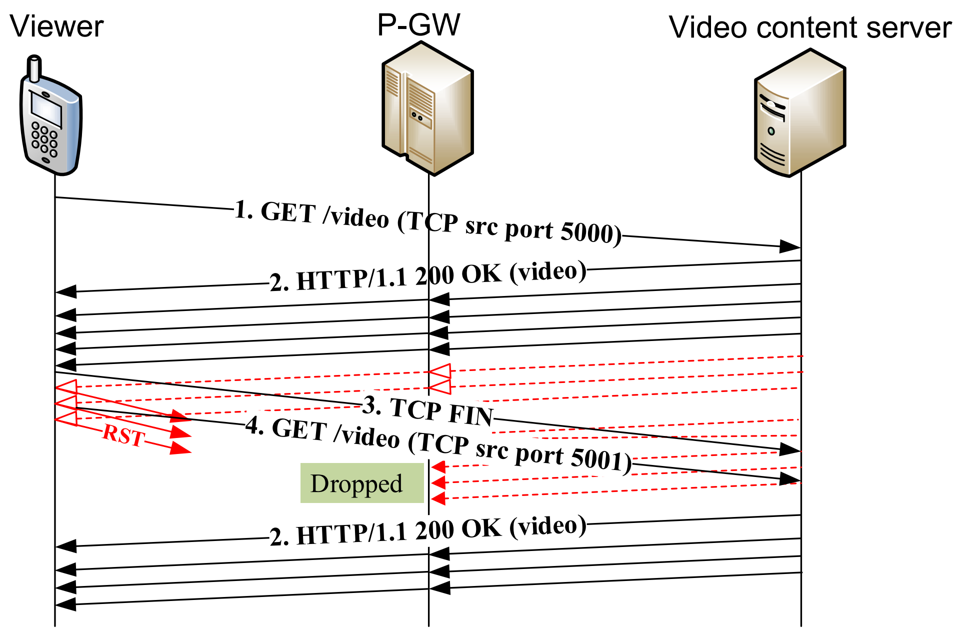

Viewer
P-GW
Video content server
1. GET /video (TCP src port 5000)
2. HTTP/1.1 200 OK (video)
3. TCP FIN
4. GET /video (TCP src port 5001)
RST
Dropped
2. HTTP/1.1 200 OK (video)

Figure 4.4: P-GW selectively drops the potential wasted video packets

solutions that dynamically change QoS parameters based on network conditions between viewers and eNodeBs. The proposed architecture requires no changes by either a server or a client. It is designed to perform the following two objectives:

1) **Swiftly downgrading QoS parameters based on network capacity over the air interface -** An ABR player specifies the requested video bitrate in the HTTP GET message. P-GW can inspect the URL and obtain the requested bitrate information selected by the player. For example, it typically requires at least 0.8 Mb/s for HQ (360p) videos and 5.4 Mb/s for HD (720p) videos. In our proposed architecture, P-GW controls the maximum possible data rate of the video streaming flow under varying network conditions. This prevents frequent bitrate changes by the player and assists the player in selecting an appropriate bitrate for a smooth streaming. As a result, this can improve QoE of viewers (Case 2 in Figure 4.3). We describe the experimental results in Section 4.4.

2) **Discarding potential wasted video content in advance before being delivered to viewers -** In the previous section, we described that an ABR player sends

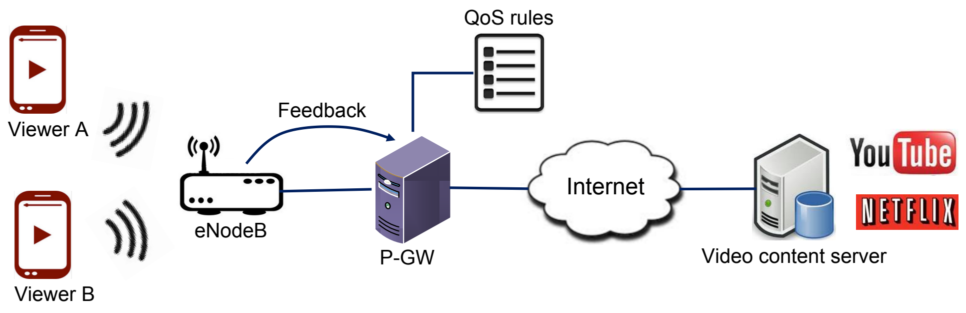


Figure 4.5: Testbed setups for evaluation

a TCP RST segment each time it receives an unexpected video packet via the terminated TCP port. We intend to drop the unnecessary video packets in advance before delivering them to the viewer over the air interface. In our proposed architecture, P-GW acts as a firewall that performs TCP header inspection and discards the unwanted traffic. When it captures the TCP FIN or RST segments sent from the video player, it starts discarding the video packets destined for the closed TCP port (Figure 4.4). This has the advantage of saving downstream bandwidth from the P-GW to the viewer.

4.4 Performance evaluation of the dynamic QoS-aware video streaming platform

In this section, we evaluate our dynamic QoS-aware video streaming platform. We measure the QoE of viewers and compare the discard ratio while playing YouTube and Netflix videos on mobile devices over Wi-Fi networks.

Building a testbed in Wi-Fi - Instead of using simulators such as MATLAB [79] and OPNET [80], we have designed a testbed using Wi-Fi to take realistic OTT video streaming traffic into account. As shown in Figure 4.5, in our prototype, a Wi-Fi access point and a proxy server, respectively, act as an eNodeB and P-GW. All the video packets between a viewer and a video content server pass through the proxy server.

We designed a set of QoS rules (Algorithm 1) to control the video streaming flows. These

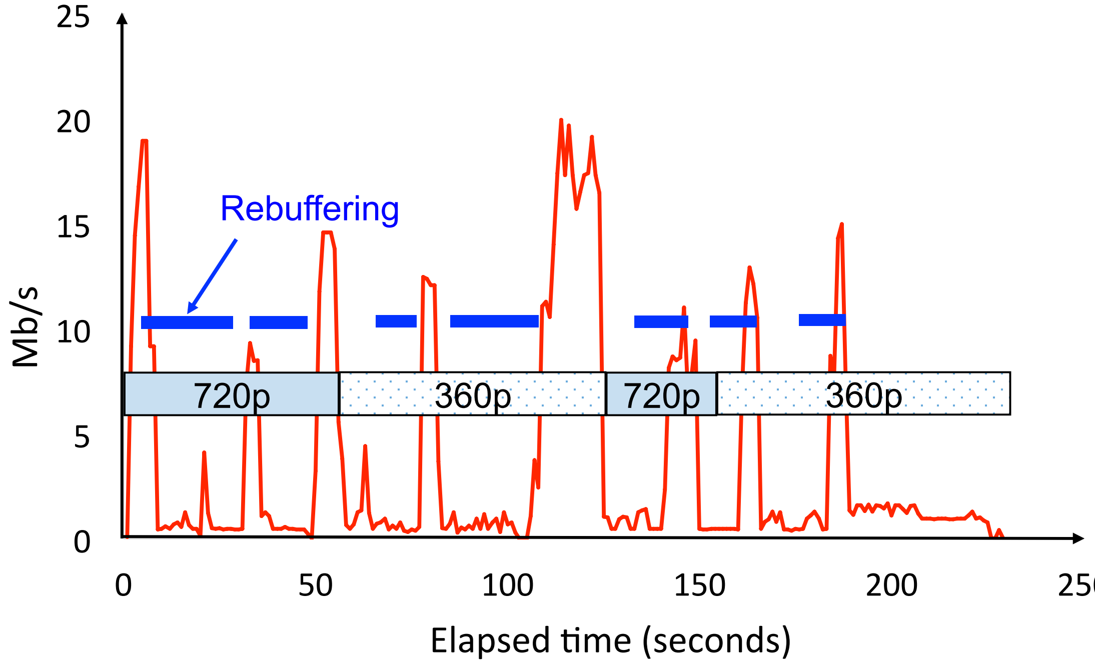
Algorithm 1

```
 1: if an HTTP GET msg received from a video player then
 2:     if $BR_{req.} \geq BW_{avail.}$ or
            $SNR_{avg.} \leq N_{thr.}$ then
 3:         (Step 1) Throttle TCP throughput of the video streaming flow until the network
    conditions become stable
 4:     end if
 5: end if

 6: if TCP RST or FIN received from a video player then
 7:     $SET_{closed} \leftarrow TCP_{srcport}$
 8: end if

 9: if a video pkt received from a video server then
10:     if $TCP_{dstport}$ in $SET_{closed}$ then
11:         (Step 2) Discard the video packet
12:     else
13:         Pass the video packet to the video player
14:     end if
15: end if
```

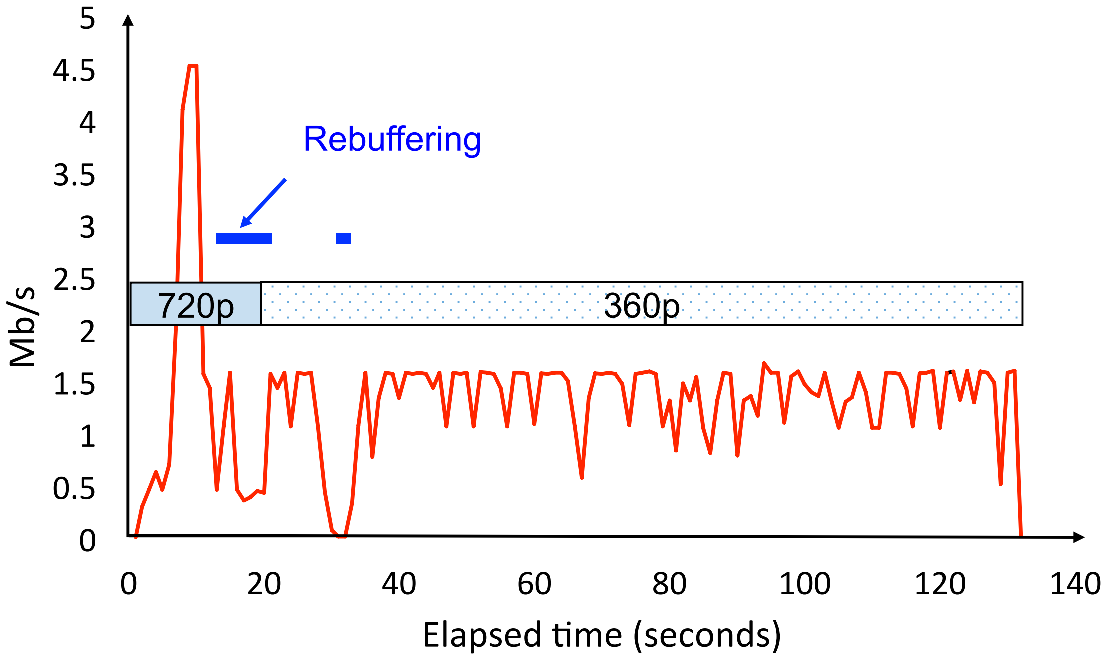
rules are designed to implement our proposed QoS-aware video streaming platform in 4G networks. Let $BR_{req.}$ be the requested bitrate selected by the video player. $BW_{avail.}$ and $SNR_{avg.}$ respectively denote the available bandwidth to the video stream and the signal-to-noise ratio (SNR) over the air interface. During the experiments, we calculated the average of SNR every five seconds, and compared the value with the predefined SNR noise threshold ($N_{thr.}$) to decide if the network was fluctuating or not. We note that the proxy server acting as P-GW only throttles TCP throughput when the video player requests an inappropriate bitrate for the given network conditions. It will decrease the maximum allowable TCP throughput on the video streaming flow, which leads the video player to switch to lower bitrates quickly.

The testbed was set up as follows: A hundred YouTube and Netflix videos were randomly selected. During the experiments, the average playing time of each video was about 10 minutes. Two viewers (Viewer A and B) on iPads requested the same video in the same network. We only applied our QoS algorithm to Viewer B and compared the performance against the baseline measured through Viewer A. The video players dynamically selected bitrate among 360p, 480p and 720p based on their own ABR heuristics. We installed a Linksys WRT54GL (802.11b/g - 54 Mb/s) Wi-Fi access point [81], and installed an open-source firmware, DD-WRT [82], on it. We wrote a script on the access point to periodically (every five seconds) report feedback on the network conditions to the proxy server. The feedback contains the transferred RX and TX bytes (used to calculate the available bandwidth) and SNR (dBm) on the air interface. To make the network fluctuate, we periodically turned on and off home networking devices (baby monitors and cordless telephones) that cause Wi-Fi interference at 2.4 GHz. To load the network, we also intentionally added the TCP traffic using a network testing tool `iperf` [65]. During the experiments, we measured the SDN on the access point. The average SNR was 47 dBm in the clean environment, but it went down to 18 dBm with interference. Based on the measurements, the SNR noise threshold ($N_{thr.}$) was set to 25 dBm. When the network condition is unstable (step 1 in Algorithm 1), using `netem` on the proxy server we set the maximum available TCP throughput 0.2 MB/s, 0.4 MB/s and 0.7 MB/s for 360p, 480p and 720p bitrate, respectively. The proxy server discarded the packets destined for the closed the TCP connection using `netfilter` and `iptables` [83] (step 2 in Algorithm 1).

Improving QoE of viewers - To evaluate the video QoE, we measured how long the viewer experienced rebufferings while watching a video. Viewer B with our QoS-aware approach experienced an average of 32 seconds less rebuffering compared to Viewer A. For example, Figure 4.6 shows the TCP throughput while playing the same YouTube video on the two iPad 3 devices in our testbed. We measured the TCP throughput until the video players on Viewer A and B fully downloaded the video files under the fluctuating network conditions. As depicted in Figure 4.6a, Viewer A experienced many rebufferings during playback (72 seconds out of 225 seconds). During the experiment, it sent 100 HTTP GET messages in total and the discard ratio was 11.6%. On the other hand, while employing our QoS-



(a) Viewer A without a dynamic QoS-aware approach



(b) Viewer B with a dynamic QoS-aware approach

Figure 4.6: TCP throughput while playing a YouTube video on iPad 3

Table 4.2: Discard ratio (%) on average while playing YouTube and Netflix videos on mobile devices over Wi-Fi networks under fluctuating network conditions

Devices	YouTube		Netflix	
	Viewer A	Viewer B	Viewer A	Viewer B
iPad 3	13.54 %	0.87 %	0.16 %	0.01 %
iPhone 4S	12.72 %	0.03 %	0.5 %	0.38 %
iPhone 3G	20.72 %	0.14 %	Not Avail.	Not Avail.
Nexus 7	2.16 %	0.49 %	14.86 %	0.13 %
Nexus S 4G	8.54 %	0.01 %	11.25 %	0.15 %

aware algorithm (Figure 4.6b), the TCP throughput of the video streaming on Viewer B was adjusted to better cope with the measured fluctuations in the channel quality. After the TCP throughput was congested at time 9 due to the baby monitor and the cordless phone, it took only 10 seconds for the video player B to change the quality. As the low bitrate was selected, it played the video with much fewer rebufferings. Consequently, Viewer B experienced rebufferings only 15 seconds out of 132 seconds until it fully downloaded the entire video file. Unlike Viewer A, it only sent 12 HTTP GET messages, and showed a discard ratio of 0.41%.

We may achieve this by improving ABR heuristics in an ABR player. For example, given the fluctuating network conditions, a more conservative bandwidth adjustment approach can avoid the frequent bitrate changes as shown in Figure 4.6a. We elaborate the details in Chapter 6.

Saving bandwidth over the air interface - We analyzed a TCP dump file for each experiment, and calculated the discard ratio to compare the performance. As shown in Table 4.2, employing our dynamic QoS-aware algorithm yields lower discard ratio. For instance, our proposed solution reduced the discard ratio up to 20.58% (case of iPhone 3G and YouTube), compared to the baseline.

4.5 Related work

Several researchers have characterized HTTP-based video streaming. In 2011, Finamore et al. [84] focused on analyzing the differences between the network traffic patterns when accessed from PCs over wired networks and from mobile devices over Wi-Fi networks. They showed that the video delivery mechanism of YouTube is more efficient for PCs than for mobile devices due to the limited capabilities of the mobile devices. In 2008, Zink et al. [85] analyzed YouTube traffic in a university campus network. By analyzing TCP/IP and HTTP packets, they characterized the duration and popularity of YouTube videos, and access patterns for YouTube video streaming. Based on their measurements, they proposed proxy-caches for video streaming to save network traffic and enhance the QoE of viewers.

In 2011, Rao et al. [86] identified the streaming strategies used by YouTube and Netflix when using Wi-Fi. They showed that the streaming strategies vary depending on the video players and the types of container (e.g., `ogg`, `mkv`, and `avi`) used for delivering video content to a viewer. Hoque et al. [87] conducted a measurement study of three popular video streaming services (YouTube, Dailymotion and Vimeo) on mobile devices over Wi-Fi and 3G networks. They analyzed the energy efficiency of the five different video streaming techniques used by the mobile video streaming services. In 2013, Liu et al. [47] analyzed and compared the performance of YouTube video streaming between Android and iOS mobile devices. They showed that Android and iOS use different approaches for downloading a video. After analyzing the traffic patterns of YouTube and different buffer management methods, they found out that iOS devices receive more duplicate YouTube video content than Android devices do. The duplicate video content affects the discard ratio as described in Chapter 4.2. In 2012 and 2013, Huang et al. [88; 89] have shown that many factors such as the size of a video chunk, dynamic TCP congestion control algorithm and competing flows in the same network make it hard to pick a proper bitrate on viewers. To resolve the issue, they have introduced playout buffer-based rate adaptation for HTTP-based video streaming.

In addition to studying the characteristics of ABR streaming, we focus on finding the root cause of video packet discard on mobile devices over Wi-Fi, 3G and 4G networks. Noticeably, in some cases, a significant amount of video content may be discarded by a video player after transferring content over the bandwidth limited air interface, resulting in undesirable waste of resources. To mitigate the misuse of network resources, we strengthen the 4G architecture evolved with our selective packet discarding mechanism. We also designed a dynamic QoS algorithm to improve video QoE.

4.6 Discussion

The general proxy-based approaches [90] are similar in spirit to our approach. In a 4G network, we can achieve this using Deep Packet Inspection (DPI) on P-GW such as Cisco's multimedia core platform [91] instead of deploying additional proxy servers between the mobile network and the video content servers in CDNs. In addition, our solutions can be compared to various split-TCP approaches that separate the volatile wireless link from the more-stable long-haul link [92]. Under unreliable wireless network conditions, the middlebox relays the video data to the viewer at a steady rate (by controlling the TCP throughput) in order to prevent frequent bitrate changes at the video player. Our future work is to empirically compare the performance of these techniques depending on various network conditions.

The limitation of our approach is that the proposed solution may not work properly for an encrypted HTTP connection. To detect video streaming traffic, we may use a DPI-based method that matches regular expressions based on the Server Name Indication (SNI) information [93]. However, it is difficult to monitor all the TCP source port numbers in HTTPS GET messages generated from the viewers' devices during a playback for our selective packet discarding mechanism. In this case, using Deep Packet Inspection of Secure Socket Layer (DPI-SSL) on a P-GW would be a possible option to address this issue [94; 95].

4.7 Conclusions

This chapter explored and analyzed two of the most popular OTT streaming services (YouTube and Netflix) on mobile devices (iOS and Android) over three wireless networks (Wi-Fi, 3G and 4G). While delivering a video to a viewer over HTTP, we observed that a noticeable amount of video packets gets discarded without being stored in the video playout buffer, after the successful delivery to the viewer's device. The discarded video content occurs when a TCP connection is repeatedly terminated and established. In such cases, the video packets that arrived via the terminated TCP connection get discarded.

To reduce the waste of network traffic and enhance video QoE for viewers, we propose a dynamic QoS-aware video streaming platform in 4G networks. Based on the feedback of network conditions over the air interface, P-GW is designed to assist a video player in selecting a proper bitrate under fluctuating network conditions, by dynamically throttling the maximum allowable TCP throughput on the video streaming flow. By monitoring TCP/IP and HTTP packets in real time, it also enables to discard the unnecessary video packets in advance before being delivered to the viewer. Our experimental results show that the proposed solution can save a significant downlink bandwidth (up to 21% improvement) over the air interface, and provide a better viewing experience on mobile devices.

Chapter 5

Towards QoE-aware Video Streaming using SDN

5.1 Introduction

Video service providers use CDNs to speed up the delivery of their contents to viewers. Today's CDN-based streaming assigns a geographically close content server to a viewer. However, even if the content server is located near the viewer, it does not always guarantee stable network conditions (Chapter 3). To improve video QoE, they use ABR technologies where an ABR player automatically adjusts bitrates based on the network condition. However, this client-side mechanism is not helpful in discovering the bottleneck that degrades the video quality during playback.

To mitigate the issue, we suggest to implement an SDN-based video streaming platform. Figure 5.1 shows our proposed architecture. Network operators deploy their own SDN controllers in their network domains and obtain network information on WAN routers in real time using OpenFlow [96]. Our video optimization server communicates with the SDN controllers via northbound APIs and video service providers to update video information (e.g., bitrate setting and addresses of video content servers in CDNs). The optimization server finds the best available content server depending on network conditions provisioned via the SDN controllers when a viewer requests a video. Based on various QoE metrics reported from video players, it enables dynamically changing routes in WANs using MPLS-

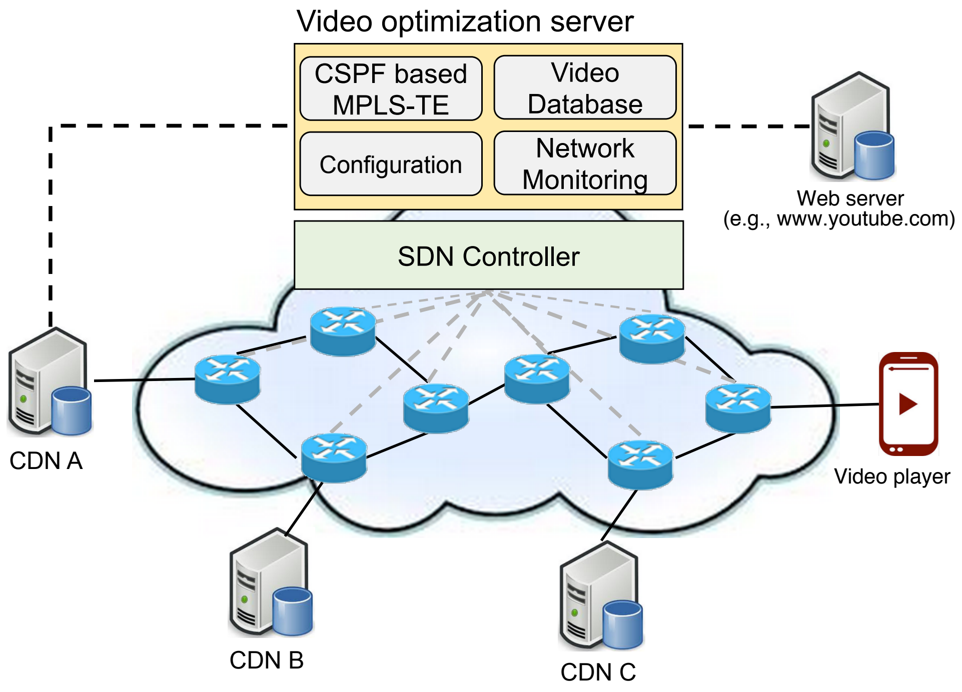


Figure 5.1: Video QoE-aware streaming platform using SDN

TE. When rebuffering events occur, for example, it sends queries to the SDN controllers to analyze network conditions (e.g., TCP throughput and packet loss rate) on the streaming flow and change the route if applicable.

We implement our SDN solutions using Junos Space SDK [26] that can monitor and control networking devices of Juniper Networks. In our testbed, we have created a lightweight plug-in in an HTML5 video player to monitor various QoE factors (e.g., rebuffering status and video bitrate) to analyze user-perceived experience when a video is playing. Our WAN traffic monitoring system is designed to communicate with the SDN controllers using RESTful APIs to visualize the network information in real time.

The remainder of the chapter is organized as follows. In Section 5.2, we briefly address problems on existing OTT video delivery systems. Our proposed SDN platform is described in Section 5.3. We explain our implementation in Section 5.4 and evaluate our solution in Section 5.5. We look at related work in Section 5.6. The challenges of our approach are addressed in Section 5.7. Finally, we summarize our conclusions in Section 5.8.

5.2 Problems on existing OTT video delivery system

OTT video delivery can be challenging because viewers, video service providers (e.g., YouTube and Netflix) and network operators (e.g., ISPs) involved do not have a global view of the end-to-end network condition. In this case, a video service provider does not have access to both the transit ISPs and the last mile network that actually reaches the viewer. Once a viewer is connected to a content server in a CDN operated by a third party, it is difficult for the service provider to track the network condition during playback. In addition, the content server is rarely switched to another node after the video starts. Thus, it is possible for the viewers to experience frequent rebufferings until the end of the video if the network is unstable. Even if the viewers pay for HD videos, they can end up watching low bitrates due to the Internet-side or CDN-side network problems.

In order to mitigate these problems, today's OTT video service providers take advantage of ABR technologies where a video player automatically adjusts bitrates depending on the network conditions (Chapter 2). Even though the streaming technologies are designed to provide smooth streaming, it does not resolve the root cause of the congestion. For instance, if the main problem is due to the link congestion in wide area networks (WANs) or the content server's malfunction, changing the bitrate is not the best way to improve video QoE. Furthermore, according to our analysis of video selection algorithms in Chapter 3, a geographically close content server is assigned to the viewer. Even though a cache server located near a viewer typically provides fast delivery, the network condition can be unstable and in such case, other content servers that are located further away may be able to provide a more reliable streaming experience.

5.3 QoE-aware video streaming using SDN

We leverage SDN to assist video service providers in selecting the best content servers when viewers request videos. In addition, we propose a Constrained Shortest Path First (CSPF) path selection algorithm over MPLS in order to find the best route for each streaming flow in WANs. As shown in Figure 5.1, our proposed SDN-based video streaming platform consists of a video optimization server, a video web server, distributed content servers in

CDNs and a viewer. Taking into account scalability and performance issues, operators may deploy multiple SDN controllers in their network. Using an SDN controller, our video optimization server (as an SDN application) monitors network conditions and updates the routing tables of WAN routers in the network. A typical utility scenario proceeds as follows:

1) A viewer sends a video request to a video web server (e.g., www.youtube.com).

2) The web server sends our video optimization server a list of available content servers that can stream the requested video at the moment.

3) Our SDN application analyzes the network conditions of each connected link on the paths to the viewer. The proposed measurements for video streaming include available bandwidths, packet loss rates and jitter. It chooses the best available content server and stores the connection information such as IP addresses of the viewer and the selected content server, an assigned MPLS label and selected video bitrate in the video database.

4) Once the connection is established, the video player running on the viewer's device periodically reports video QoE metrics to the video content server (Section 5.3.1).

5) When a rebuffering occurs, our video optimization server tries to pinpoint a bottleneck (Section 5.3.2). It is designed to find the best available routing path based on the CSPF algorithm over MPLS (Section 5.3.3).

6) We dynamically change the content server if all available paths from the assigned node experience congestions. In this case, the first assigned content server sends an HTTP redirection message to the video player, and has the viewer connected to another available content server that can provide the content with higher networking performance. The address of newly assigned content server can be obtained directly from the video optimization server. Once the viewer is connected to the new content server, the video player continues to play the rest of the video.

5.3.1 Application-level video QoE metrics

Video service providers typically do not have any access to last mile networks (e.g., local ISPs) of viewers. We propose to measure end-to-end network conditions between a video player and a content server. In our proposed architecture, the content server is designed to periodically receive various QoE measurements directly from within the video player in order to analyze user-perceived video quality.

Existing QoS metrics such as packet loss rate, goodput, delay, jitter and throughput are used to indicate the impact on the video quality from the network operator's point of view, but do not present the user-perceived video quality. Moreover, it is difficult to use the traditional Peak Signal to Noise Ratio (PSNR) [97] where received frames and referenced frames of an original video are compared to measure video QoE. However, such frame-to-frame comparison mechanism does not directly reflect the common QoE metrics such as video start-up latency, rebuffering rate and playout buffer status [98; 99].

5.3.2 Pinpointing a bottleneck using SDN

Our video optimization server is capable of catching rebufferings based on the feedback directly from a content server that obtains the QoE metrics from a video player. It is straightforward to find the bottleneck link in an SDN-enabled network. When a rebuffering occurs, it first obtains the flow information from the video database (Figure 5.1) such as source and destination IP addresses, routing paths (selected MPLS labels) and requested video bitrate. Then, it sends queries to collect the current data rate of the video streaming flow on each connected link on the path (e.g., obtaining network statistics of an individual flow using OpenFlow [96]). There is a recommended downloading bitrate that represents the amount of bitrate required to play the selected bitrate without any viewing interference. For example, YouTube requires 2.5 Mb/s for 720p and 725 kb/s for 360p. If a link provides lower data rate than what is required for the current streaming flow, we define this link as a bottleneck, which may be the cause of rebufferings on the viewer's device.

Table 5.1: Required TCP throughput for CSPF-based path selection algorithms

Selected bitrate	Required TCP throughput
1080p	5 Mb/s
720p	2.5 Mb/s
480p	1 Mb/s
360p	725 kb/s
240p	325 kb/s

5.3.3 Dynamic network condition-aware path optimization with SDN

We use MPLS-TE over SDN to control video streaming flow [100; 101]. MPLS enables ISPs to provide QoS in layer 3 networks. In an MPLS network, routers perform packet-forwarding decisions based only on the labels assigned on data packets instead of inspecting an IP address of each packet. Different MPLS labels are assigned to corresponding Labeled Switch Paths (LSPs). Typically, those labels are attached to IP packets or removed from the packets at Label Switch Routers (LSRs) and label swapping can be performed on the intermediate routers. From a QoS standpoint, MPLS-TE allows network operators to efficiently manage different kinds of data streams based on service plans and speed up network traffic flow. According to recent studies [100; 101], the MPLS-TE architecture can be more flexible and simpler on a SDN platform compared to the traditional implementation, by separating the control plane from the data plane. In this chapter, we apply it to the video streaming use case and build a prototype to show the feasibility of developing MPLS-TE over SDN.

We implement a CSPF algorithm over MPLS-TE in order to select the best available route from a content server to a viewer. It runs the shortest path algorithm after selecting links that meet a given set of constraints. In our case, we take account of three constraints (current available TCP bandwidth, packet loss rate and jitter) that are typically considered important for video streaming. When a viewer experiences a rebuffering, for example, our optimization server collects network conditions on the connected links and runs the CSPF algorithm with the required bandwidth in Table 5.1 to find the best available LSP. We consider packet loss rate ($< 5\%$) for buffered video streaming and put more weight on

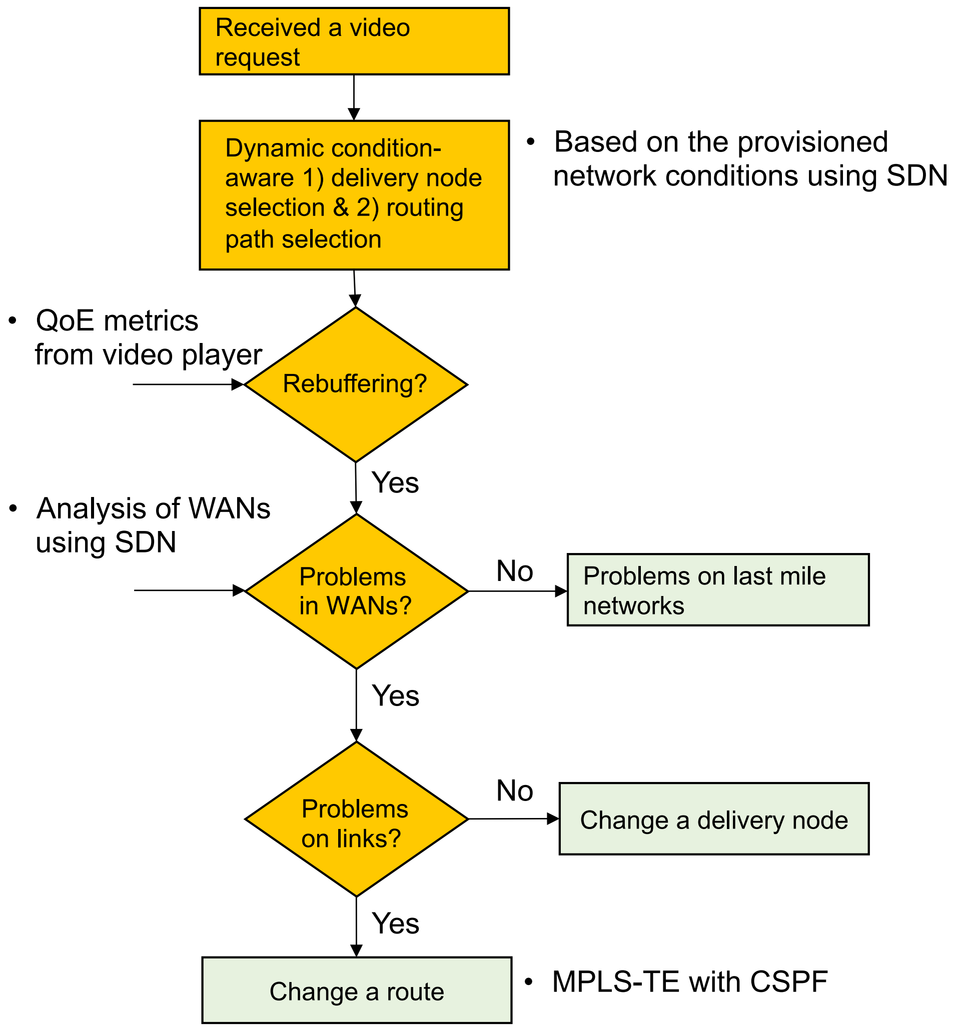

Received a video request
Dynamic condition-aware 1) delivery node selection & 2) routing path selection
• Based on the provisioned network conditions using SDN
• QoE metrics from video player
Rebuffering?
Yes
• Analysis of WANs using SDN
Problems in WANs?
No
Problems on last mile networks
Yes
Problems on links?
No
Change a delivery node
Yes
Change a route
• MPLS-TE with CSPF

Figure 5.2: A simplified flowchart of a decision tree

packet jitter (< 20 ms) for UDP-based live streaming.

Basic CSPF algorithms where a set of video streaming requirements is considered to select the best LSP may encounter load-balancing problems on WAN links. For instance, if multiple viewers request to change routes from the same time and place, the current CSPF algorithms may lead all the viewers to take the same LSP. If the selected link is running at 80% - 90% utilization and sudden spikes of network traffic arise (e.g., during busy hours), the link may become overwhelmed and start to drop packets, which eventually leads to poor video QoE. Taking into account the load-balancing on WAN links, our CSPF algorithm has the following rules:

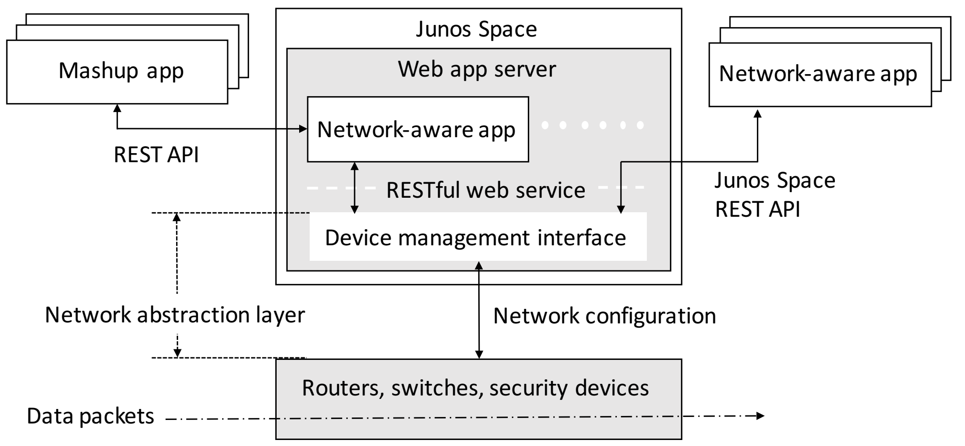


Figure 5.3: Junos Space architecture

1) Prune WAN links that do not satisfy the required bandwidth, packet loss rate and jitter.

2) If multiple LSPs that meet the requirements are available, select the LSP with the lowest link utilization.

3) If several LSPs have the same link utilization, select the LSP with the smallest number of hops.

In summary, Figure 5.2 shows our simplified flowchart of a decision tree in our proposed SDN-based video streaming platform.

5.4 Implementation

As a proof of concept, we have implemented our SDN-based video streaming architecture using Junos Space that is a comprehensive network management solution developed by Juniper Networks [26; 102]. It provides a centralized management plane across Juniper's switching, routing and security networking devices (Figure 5.3). The platform allows third parties to control the devices through standards-based Representational State Transfer (RESTful) APIs. Our proposed platform is designed to improve network utilization and user-perceived video quality under dynamic network conditions. In order to achieve this, we have implemented server and client-side applications over SDN.

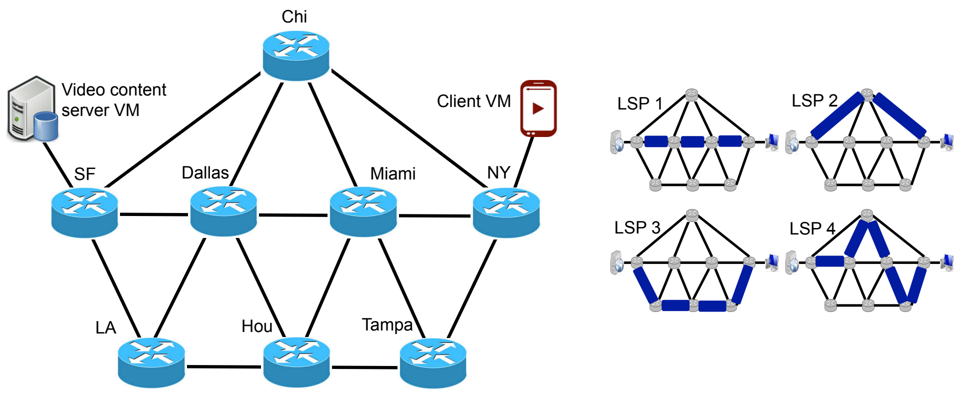


Figure 5.4: Implementing a testbed using Junos Space and WAN routers of Juniper Networks

- **Server-side application**: This is a video optimization server that determines the best routes and updates MPLS labels in real time based on our CSPF-based algorithm. It communicates with the SDN controller using RESTful APIs. It also provides GUI in order to visualize network topology and networking statistics.

- **Client-side application**: This is a lightweight plug-in embedded in an HTML5 video player. It is designed to identify the video bitrate selected by a viewer and periodically report user-perceived quality to a content server. The QoE metrics include the player's state (e.g., playing, paused and finished) and the status of video playout buffer while downloading a video. The information is periodically delivered to the connected content server over HTTP POST messages.

Figure 5.4 shows our testbed network[1]. A network control machine using the Junos Space SDN platform is connected to eight Juniper edge routers. The routers use MPLS to deliver video packets. Each router references the MPLS short label to decide a LSP route of traffic flow instead of performing an IP address lookup. The video packets are delivered from the server located in SF to the viewer located in NY via one of the four predefined LSPs.

[1]All edge routers are placed in the lab in New Jersey. We use the different location labels and emulate WAN network conditions.

5.5 Evaluation

Due to the difficulties of creating real WAN traffic and in order to test our routing algorithms extensively in various scenarios, we have created a simulation tool that reflects the same network topology as in our testbed. In our simulated network, video packets are delivered from virtual hosts (video servers and viewers) via the links that are connected among virtual WAN routers. The video player running on the client-side has been designed to adjust bitrates based on downloading TCP throughput of streaming flow in the network.

We assume that links with 100 Mb/s bandwidth capacity are running between 80% and 90% utilization during busy hours. In order to simulate real network conditions, we take into account recent mobile streaming statistics [103] indicating that most viewers watch low or medium bitrates, and about 1% of total mobile subscribers watch high definition (e.g., 720p and 1080p) videos. Based on the information, we inject background traffic flows on each link that follow a Poisson distribution where 200 viewers on average request a video per minute from each router and 99% of total streaming flows generate 0.5 Mb/s on average and 1% of total flows consume 2.5 Mb/s bandwidth on average. In order to show the feasibility of our approach, we measured the following two scenarios:

1) **Non-ABR streaming**: A non-ABR video player does not switch bitrates during a download. Without our solution, a viewer continues to watch a video with 1080p via LSP 2 that has minimum number of hops among LSPs. In QoE-aware streaming, the player downloads the same video via dynamically changing LSPs based on our CSPF algorithm.

2) **ABR streaming**: An ABR player automatically adjusts bitrates based on network conditions (Chapter 2). Without our solution, the player switches bitrates but does not change LSPs. In QoE-aware streaming, the player downloads the 1080p video via dynamically changing LSPs. It only degrades bitrates if there are no available LSPs that meet the required TCP throughput (e.g., 5 Mb/s for 1080p).

We experimented each scenario five times and evaluated the performance. Figure 5.5 shows the available bandwidth on each LSP in Scenario 1 (one download). For the first 20 minutes, there are no good LSPs with more than 5 Mb/s of available bandwidth. Figure 5.6

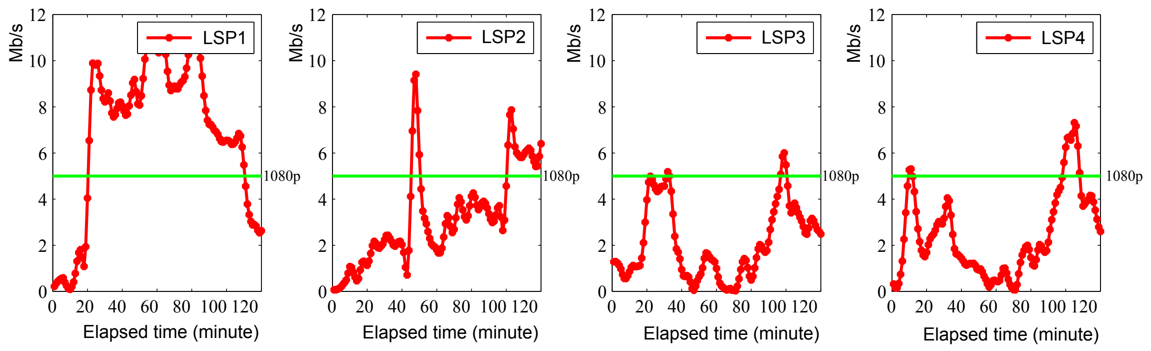


Figure 5.5: Available bandwidth capacity on LSPs in Scenario 1

shows the experimental results in Scenario 1. The abscissa represents the elapsed time and ordinate indicates the downloading data rate at client-side. We measured the data until the player completely downloaded the video content. We monitored the accumulated received bytes every minute and compared the downloading data rate with the required bitrate of the selected bitrate. We put a square box if the video player experienced bad networking conditions (downloading data rate < required bitrate) for at least five seconds during the sampling period. In such unstable network conditions, there is a high possibility of experiencing rebufferings at the client-side.

For non-ABR streaming without our solution, the video player had bad viewing experience for 52 minutes out of 100 minutes (the total length of square box) due to traffic, via LSP 2. It took 111 minutes to download the full size of the video. The video player with our QoE-aware mechanism over SDN experienced unstable networking conditions for only 21 minutes in total. At the beginning, the SDN controller switched the path from LSP 2 to LSP 3 since it was the best one among all the others, and then it changed to LSP 1 at time $t = 39$ minutes to provide a fast delivery. As shown in Figure 5.6b, it took only 63 minutes to download the entire size of the video.

For ABR streaming, we played a 1080p movie lasting 100 minutes long. We measured how often the video player switched bitrates while playing the video. Figure 5.7 shows our experimental results. The ordinate represents the video bitrate selected by the video player. As we see in Figure 5.7a, the video player often changed bitrates and suffered from more periods with bad viewing experience, compared to the one in Figure 5.7b. In

Table 5.2: ABR video bitrates chosen as fraction of time and period of bad viewing experience while playing a video with 100 minutes of length

	Static route	Our QoE-aware solution
Avg. bad viewing experience period out of 100 minutes	10.4 minutes	4.6 minutes
1080p	69%	77%
720p	20%	16%
480p	11%	7%

this experiment, the video player with a static route played 1080p, 720p and 480p bitrate for 50 minutes, 24 minutes and 26 minutes, respectively. On the other hand, the video player with our QoE-aware mechanism downloaded the 1080p video most of the time. We conducted the same experiment a hundred times and calculated the statistics. As shown in Table 5.2, our QoE-aware mechanism over SDN reduces the bad viewing experience by 5.8 minutes on average, and provides higher bitrates while the video player downloaded the content.

5.6 Related work

Traditionally, QoS routing has been studied to compute the best network routes for the requested QoS parameters and improve the network resource utilization. Shigang et al. [104]. have addressed the overview of QoS routing algorithms such as source routing, distributed routing and hierarchical routing. They have presented the strengths and the weaknesses of different routing algorithms. Vitoria [105] has described a practical architecture for implementing a QoS-enabled IP network. In the article, she has addressed several IP and QoS technologies such as MPLS, IPSec, traffic shaping and VPNs in WANs to build end-to-end QoS for VoIP.

Several researchers have investigated an application-aware SDN platform and WAN routing control using SDN. Zafar et al. [106] focus on a mobile application detection framework. They use a traffic classification technique based on machine learning to identify the application types in SDN. Ali et al. [100] have introduced MPLS-TE and MPLS VPNs with OpenFlow. They have shown a demo where MPLS control plane features are implemented on an SDN platform. Saurav et al. [101] have demonstrated application-aware aggregation and traffic engineering in a packet circuit network. Using a NOX SDN controller [107], they dynamically controlled packet flows based on different application types.

Michael et al. [108] have shown SDN-based application-aware networking for YouTube video streaming. They have conducted a performance test of several path selection mechanisms such as round-robin, bandwidth-based, deep packet inspection-based and application-aware in an SDN-enabled network. The Aricent group [109] has introduced application-aware routing with SDN as a business model. They address that SDN-based routing control allows service providers to lower operating expenses and improve the overall end-user experience.

5.7 Discussion

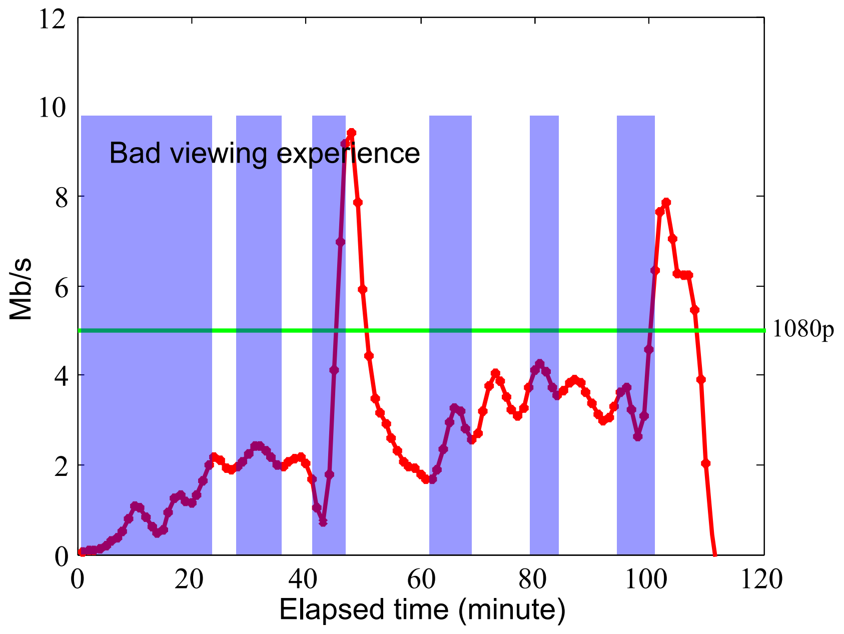
Our approach differs from the prior work in two aspects: First, we focus on video streaming. Unlike other approaches that introduce application-aware SDN platform, we have designed our solution from the perspective of video service provider. With the support of SDN platform, we dynamically control video streaming routes in WANs and change video content servers based on real-time network conditions. Secondly, we have implemented our solutions using a commercial off-the-shelf SDN platform, Juniper network's Junos Space [26], to show the feasibility of our approaches.

Some researchers address challenges and limitations on building the SDN platform in WANs [110; 111; 112; 113]. First, it is necessary to create a standardized northbound API above the SDN controllers. In our proposed architecture, ISPs operate their own SDN platform and share various networking information in real time using the northbound APIs such as RESTful. Therefore, they need to decide what types of networking parameters the controllers should pass through the APIs or how the controllers communicate among each others. Secondly, an application-aware routing approach has scalability concerns. Unlike traditional routing protocols that compute the best paths for a given destination using simplistic metrics such as hop count and cost, the application-aware approach assigns different paths for a given application such as video or VoIP. To achieve this, the SDN controllers need to track all the traffic characteristics, analyze network conditions, compute the best paths and share the routing information with others in real time. This may cause a large volume of signaling messages in WANs and may take a long time to decide the routing paths for all the individual flows for large networks.

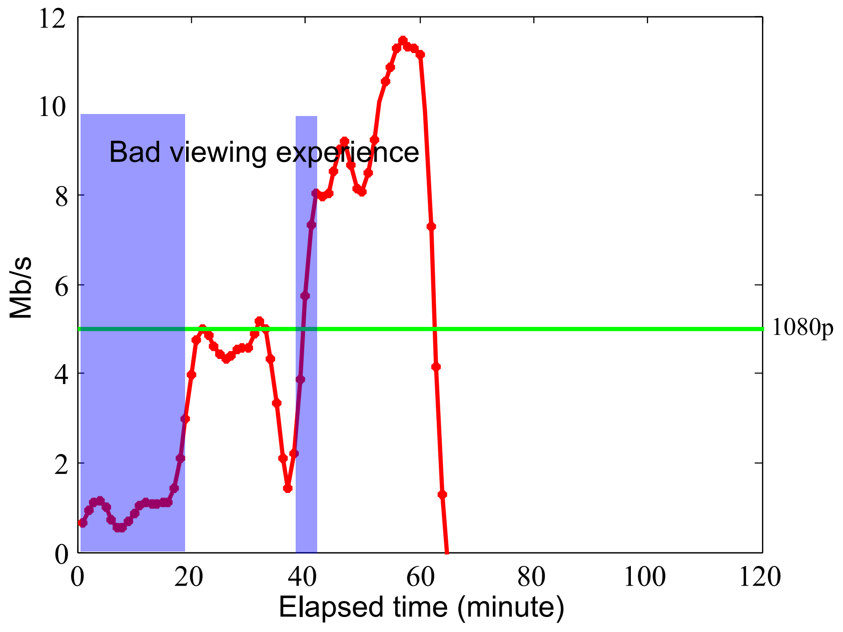
5.8 Conclusions

In today's OTT video delivery platform, it is difficult to track QoE of viewers once a content server has been connected to a viewer. Without changing routing paths and content servers, only switching bitrates at client-side may not resolve the bottleneck problems that degrade video QoE. For instance, it is possible that the routing paths between the assigned content server and the viewer experience congestion at the moment.

To resolve this issue, we propose to use MPLS-TE over SDN in WANs. To monitor watching experience of a viewer in real time, we propose to measure video QoE metrics (e.g., rebuffering status and video player state) directly from within the video players during a download. Based on the end-to-end feedback, our video optimization server selects the best available content server that can stream the content with more reliable network conditions than others presently and dynamically change routes among WAN routers using MPLS-TE. In our testbed setups, our proposed QoE-aware mechanism shows 56% improvement on enhancing viewing experience especially during busy hours. It selects better routing paths to provide higher bitrates during playback.

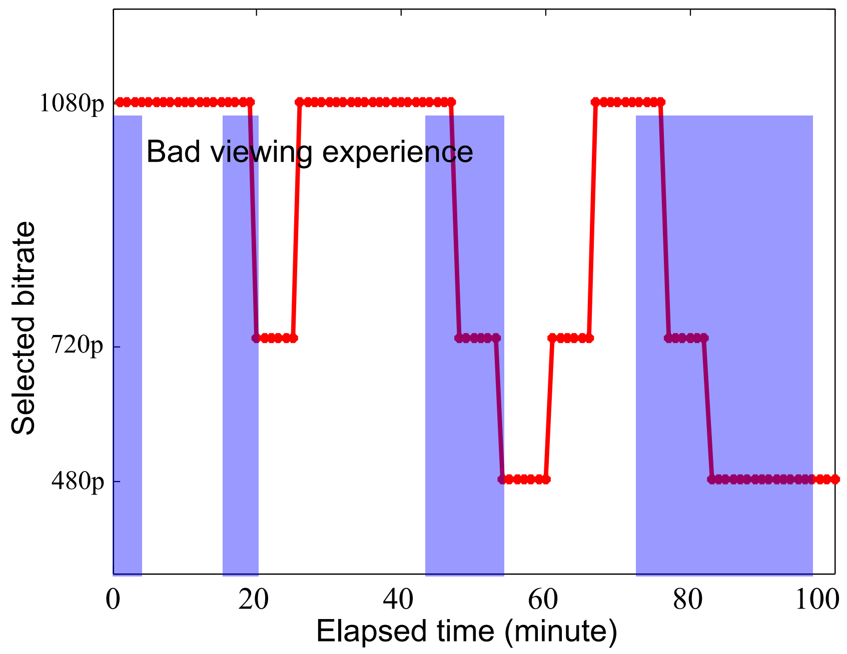


(a) Non-ABR streaming with a static route

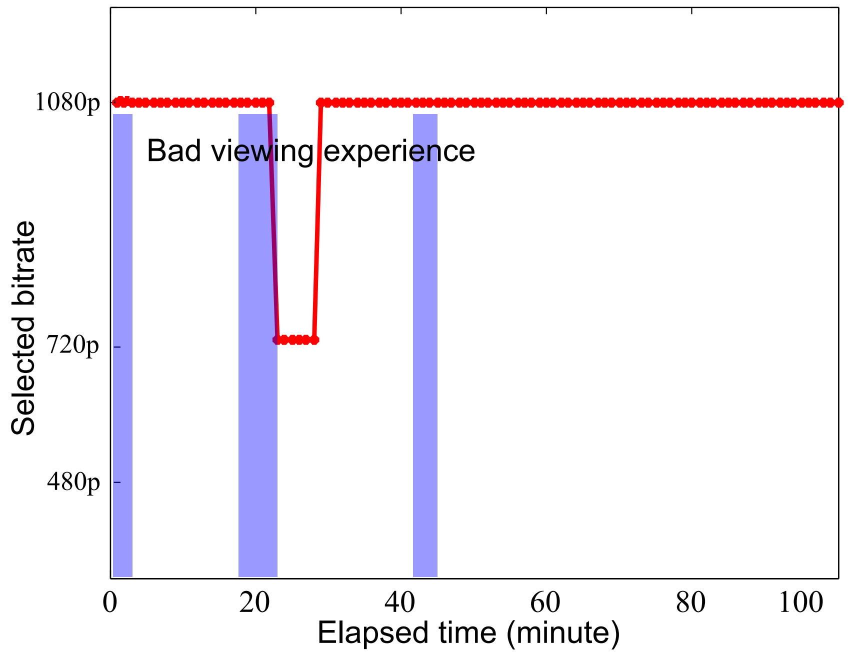


(b) Non-ABR streaming with our QoE-aware solution

Figure 5.6: Monitoring TCP throughput and period of bad viewing experience in non-ABR streaming



(a) ABR streaming with a static route



(b) ABR streaming with our QoE-aware solution

Figure 5.7: Monitoring TCP throughput and period of bad viewing experience in ABR streaming

Part III

ABR Streaming Heuristics

Chapter 6

An Empirical Evaluation of Playout Buffer Dimensioning in ABR Streaming

6.1 Introduction

We analyzed the playout buffer size in ABR video streaming and its impact on video QoE. As we described in Chapter 2, an ABR player can avoid frequent bitrate changes and rebufferings by storing video data in the buffer up to its maximum buffer size. Therefore, it may be reasonable to assume that a larger buffer always achieves a better viewing experience. To test this hypothesis, we first implemented several Microsoft's Smooth Streaming (SS) players [114] with different playout buffer sizes, and compared the ABR performance under the same controlled network conditions. Through our experimental results, we observe that not only the remaining playout buffer level but also its maximum buffer size affects bitrate switching behaviors in ABR streaming. Even though the current buffer level is the same, for instance, the player selects bitrates differently depending on its maximum playout buffer size. To figure out how much it can affect QoE of the viewers, we conducted a survey using an online crowdsourcing platform. More than 200 participants watched our short video clips that show distinct bitrate switching behaviors referenced by different playout buffer sizes, and scored their viewing experience for evaluation. Our QoE survey reveals that, in general, a large buffer outperforms a small buffer by causing fewer bitrate changes and suffering from fewer rebufferings under slowly varying network conditions. But, interestingly, we also observe that a small buffer can achieve higher QoE than a large buffer, especially under fast varying network conditions. When available throughput increases after low bandwidth periods, a small buffer reaches higher bitrates more quickly than a large buffer that pauses downloading the high bitrates until the player consumes all the low bitrates in its large buffer.

Based on these findings, we suggest an ABR player to not only change bitrates but also switch its maximum playout buffer size adaptively depending on the remaining buffer occupancy. The key idea is to store video segments with high bitrates as much as it can when network bandwidth is sufficient, and reach the high bitrates quickly while the network stabilizes after congestion. Our experimental results show that instead of using a fixed buffer size (e.g., small or large), an ABR player dynamically switching between small and large buffers can offer 15% higher played bitrate, 70% of fewer bitrate changes and 50% shorter rebuffering duration under varying network conditions.

The remainder of the chapter is organized as follows. Our motivation is described in Section 6.2. In Section 6.3, we analyze the relationship between playout buffer size and bitrate switching behavior in ABR streaming, and our QoE survey results are presented in Section 6.4. Our proposed solutions are described in Section 6.5. We look at the related work and summarize our conclusions in Section 6.6 and 6.7, respectively.

6.2 Motivation

The purpose of using ABR technologies is to provide smooth streaming in the highest possible bitrate. In our earlier technical report [115], we analyzed ABR performance of two video streaming services (Netflix and Redbox Instant) under varying network conditions. Even though they use the same ABR technology (Microsoft's SS), we observed different behaviors in bitrate switching. This is a result of each streaming service using their own ABR configurations such as different size of playout buffer and segment duration. To com-

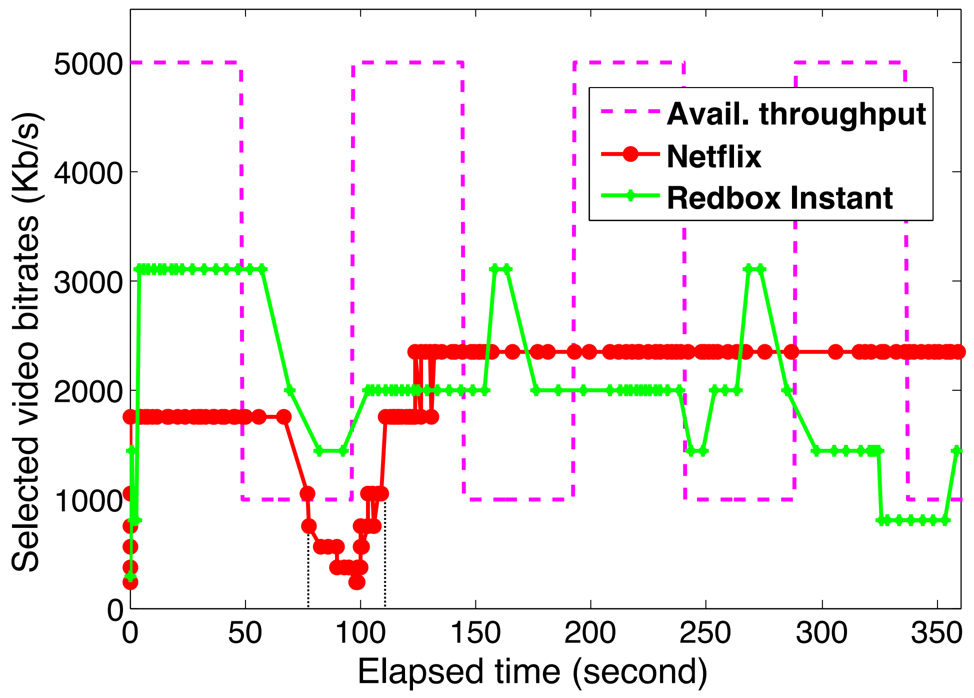


Figure 6.1: Video bitrate changes during playback under fluctuating network conditions

pare their bitrate switching behaviors, we arbitrarily throttled and increased the available bandwidth in the network between the video players and the Internet access point. The experimental results are shown in Figure 6.1. Under the fluctuating network conditions in the experiments, the Netflix player causes fewer bitrate changes than the Redbox Instant's player. We also observed Netflix downloading lower bitrates, even though there is enough network bandwidth available at elapsed time between $t = 80\,s$ and $t = 110\,s$. This inspires our following questions: what factor could cause this different behavior? Although their video segments are similar in size ($4\,s$ for Netflix and $5\,s$ for Redbox Instant), we noticed apparent differences in their playout buffer sizes. We tracked the playout buffer sizes while the videos were downloaded. We observed that Netflix uses a $245\,s$ buffer size while Redbox Instant uses $29\,s$. So, could the answers be related to different size of playout buffer in the players?

According to other work [116; 14], remaining buffer space is important for an ABR player in deciding between increasing, decreasing or keeping the current bitrate for the next segment. For example, if there is enough content in the buffer, the player will not necessarily downgrade the bitrate due to the risk of changing bitrates under fluctuating network conditions. But if the buffer is running low, the player will quickly switch to a lower bitrate in order to avoid potential rebufferings. After considering this behavior, however, we are still left with these questions: if this is indeed the main reason, why do some video streaming services continue to use small buffers? Are there any trade-offs between playout buffer size and video QoE?

In this chapter, we investigate the role of playout buffer size in ABR streaming and try to answer the following questions:

- How different sizes of playout buffers affect bitrate switching behaviors in ABR streaming?
- Does a large buffer always achieve higher QoE than a small buffer?
- What factors can be used to analyze the impact of playout buffer size on video QoE?

In some cases, measuring network QoS parameters (e.g., downloading throughput, video packet jitter and latency) is useful in representing the impact on video quality level from the network operators' perspective. However, these measurements cannot accurately pinpoint the video quality perceived by the viewer. So as a way to analyze QoE of viewers, we measure various playback events (e.g., bitrate changes and rebufferings) directly from within video players using existing ABR streaming platform instead of using QoS metrics.

6.3 Analysis of the role of playout buffer size in ABR streaming

In ABR streaming, a playout buffer is used to store video data ahead of playing time to avoid unnecessary bitrate changes and rebufferings when networks experience congestion. Our first step is to understand the role of playout buffer size embedded in an ABR player. In order to analyze the fundamental of ABR heuristics related to the playout buffer size, we implemented customized ABR players and a streaming server using Microsoft's SS platform.

Table 6.1: Video bitrate settings - *Big Buck Bunny*

No.	Bitrate	Resolution	File size
1	2,962 kb/s	1280×720	221 MB
2	2,056 kb/s	992×560	157 MB
3	1,427 kb/s	768×432	113 MB
4	991 kb/s	592×332	81.7 MB
5	688 kb/s	448×252	60.2 MB
6	477 kb/s	368×208	45.2 MB
7	331 kb/s	284×160	34.8 MB
8	230 kb/s	224×128	27.6 MB

6.3.1 Testbed setups

Building ABR players and server: We use Microsoft's Internet Information Services (IIS) SS APIs [117], also known as Microsoft's SS player development kit, to build an ABR player on Microsoft's Silverlight platform [114]. As a test video, "Big Buck Bunny 720p HD" was chosen, which is available from the Microsoft official website [118] and last 10 minutes 35 s. Microsoft expression encoder was used to encode this video into eight different bitrates. The settings for each bitrate are described in Table 6.1. The playback duration of a segment is two seconds. Using the tool kit, we developed four ABR players, customized with different playout buffer sizes: $BS = 20\,s$, $BS = 30\,s$, $BS = 40\,s$ and $BS = 100\,s$, where $BS = t\,s$ denotes the maximum playout buffer size with t s. In other words, the player can store video data up to t s in the playout buffer. During the experiments, the players downloaded the test video from our IIS SS web server on Windows desktops. In order to control network condition as much as possible, we placed the ABR players and the server in the same campus network. We note that the RTT is less than 2 ms between the players and the server. Using the IIS logging platform, we retrieved various playback statistics such as rebufferings, initial start-up delay, bitrate changes and remaining buffer level in real time. In addition, we analyzed TCP/IP and HTTP packets generated from the players using

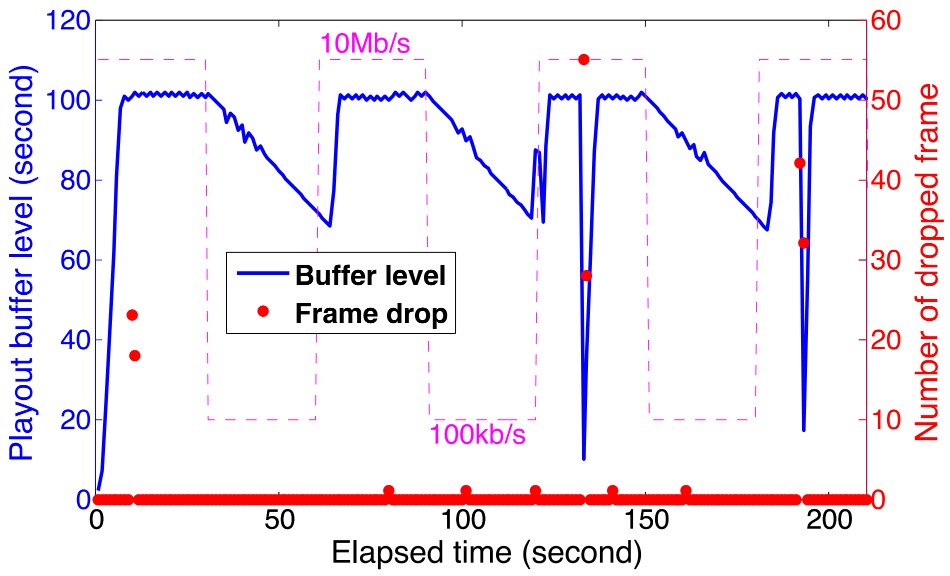


Figure 6.2: Monitoring a playout buffer level and frame drop ratio under changing network condition with large bandwidth variation

`Wireshark`, to compare the requested bitrates that are contained in HTTP GET messages with the actually played bitrates during a download.

Emulating networking conditions: During the experiments, we control the available throughput using networking emulation tools (e.g., NetLimiter [119] and Fiddler [120]), allowing full control of incoming and outgoing Internet bandwidth over applications. We emulate the following network conditions:

- **Long-term *High* and *Low* cycle:** We periodically throttle and increase available bandwidth in the network for long periods of time (e.g., $10\,s$, $20\,s$, $30\,s$ and $50\,s$). It is common for mobile users such as 3G and 4G to confront such long-term bandwidth changes. For example, nomadic users in mobile networks may experience slowly varying or disconnected network conditions especially during handover between base stations [121]. Further, the same situation can occur when a mobile user on high-speed railway is passing through a tunnel. The user will experience the lack of available bandwidth until the mobile device finds a strong signal again.
- **Short-term *Up* and *Down* spikes:** Short-term spikes (e.g., a few seconds of avail-

able bandwidth variation) are common in practice over wireless networks, in particular Wi-Fi [14]. It is commonly caused by home networking devices that cause signal interference at 2.4 GHz (e.g., baby monitors, microwave ovens and cordless phones) or by channel contention caused by neighboring Wi-Fi access points [122]. From the application's point of view, this problem appears as short-term *Down* spikes where available bandwidth is repeatedly throttled down for few seconds and then becomes stable again.

The above network conditions are set up to observe the bitrate switching behaviors along with different playout buffer sizes. We emulate fluctuating network conditions by throttling (down to 400 kb/s) and increasing (up to 2,100 kb/s) the available bandwidth in the network.

6.3.2 Analysis of experimental results

In this section, we present our experimental results and summarize key findings. As a result, an ABR player is shown to have a distinct behavior of bitrate switching depending on its maximum playout buffer size.

Analysis methodology and metrics: As shown in Figure 6.3 through 6.6, we plotted the bitrates selected by the ABR players and remaining buffer levels during the download. For example, in Figure 6.3a, the dotted line represents the available incoming throughput on the viewer's device in accordance with the pre-defined *High* (2,100 kb/s) and *Low* (400 kb/s) bandwidth cap. The selected bitrates are marked every time the players requested the video segments during playback. Figure 6.3c shows the remaining playout buffer level as time elapsed[1], and each mark is plotted every second. Each experiment has been conducted thirty times and the average playback statistics are calculated in Table 6.2 and 6.3. We calculated the average played bitrate (fourth column in Table 6.3) and bitrate changes (the last column in Table 6.3) using the following equations:

$$\text{Avg. played bitrate (kb/s)} = \frac{\sum_{\mathrm{i}=1}^{\mathrm{n}} \mathrm{B_i} \times \mathrm{D_i}}{\sum_{\mathrm{i}=1}^{\mathrm{n}} \mathrm{D_i} + \mathrm{T_{total}}} \tag{6.1}$$

[1]We used the Microsoft's Internet Information Services (IIS) SS APIs [117] to track the playout buffer level in real time.

Table 6.2: Distribution (%) of played bitrates during playback

Cycle	Max. BS	2,056 kb/s	1,427 kb/s	991 kb/s	688 kb/s	477 kb/s	331 kb/s	230 kb/s
High	20 *s*	0	11.1	42.2	31.1	6.7	4.4	4.4
(10 s)	30 *s*	0	11.6	7	55.8	11.6	4.7	9.3
Low	40 *s*	0	11.2	7.2	56.3	11.5	4.6	9.2
(10 *s*)	100 *s*	0	11.8	7.1	50.6	21.2	4.7	4.7
High	20 *s*	8	27.4	10.4	17.9	15.4	6.5	14.4
(30 *s*)	30 *s*	2.4	37.8	18.7	15.8	10.5	5.7	9.1
Low	40 *s*	0.5	43.4	11.3	19.3	12.3	5.7	7.5
(30 *s*)	100 *s*	0.5	38.3	12.4	18.7	15.8	4.8	9.6
High	20 *s*	0	14.4	8.1	8.1	4.5	13.4	51.4
(20 *s*)	30 *s*	0	18.6	11.5	11.5	7.1	17.5	33.7
Low	40 *s*	0	17.8	13.3	13.3	16.9	13.8	24.9
(50 *s*)	100 *s*	0	16.9	16	11.6	16	16.9	22.7
Down	All	0	40.3	52.9	3.4	0	0	3.4
Up	20 *s*	0	0	0	0	31	29.3	39.7
	30 *s*	0	0	0	7.1	35.7	16.1	41.1
	40 *s*	0	0	0	16.6	28.4	14.2	40.8
	100 *s*	0	0	0	17.1	31.1	12.1	39.7

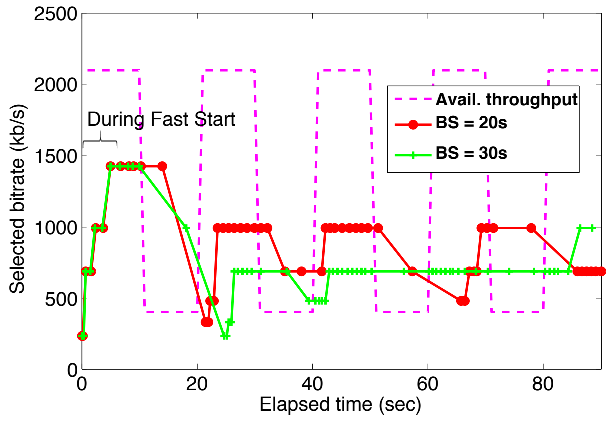
$$\text{Avg. BR changes (kb/s)} = \frac{\sum_{\mathrm{i}=2}^{\mathrm{n}} |\mathrm{B_i} - \mathrm{B_{i-1}}|}{\text{Total number of bitrate changes}\,(\mathrm{n}-1)} \tag{6.2}$$

B_i denotes i_{th} bitrate that actually played during playback. The duration of each B_i is represented as D_i, and T_{total} indicates the total duration of rebufferings in video session. Equation 6.2 shows how much bitrate increases or decreases in kb/s on average whenever a bitrate changes during a download. We note that the results from Equation 6.1 and 6.2 do not reflect the impact of frequency and duration of bitrate switching. For instance, let's suppose that there is a case where a bitrate changes from 100 kb/s (30 *s*) ➜ 200 kb/s (30 *s*) in

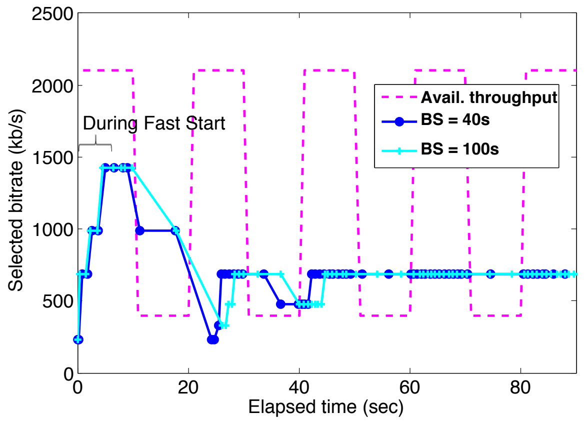
Table 6.3: Video segment downloading statistics

Cycle	Max. BS	Num. of segments	Avg. bitrate (kb/s)	Num. of bitrate changes : length (second)	Avg. variance (kb/s)
High	20 *s*	55	872	13:7	376
(10 *s*)	30 *s*	59	680	9:9.9	363
Low	40 *s*	64	672	8:10	354
(10 *s*)	100 *s*	66	672	7:10.1	341
High	20 *s*	115	856	27:7.9	442
(30 *s*)	30 *s*	120	968	21:9.6	391
Low	40 *s*	128	952	21:10.2	324
(30 *s*)	100 *s*	131	952	20:11.1	321
High	20 *s*	123	504	32:7.2	271
(20 *s*)	30 *s*	129	616	31:7.4	273
Low	40 *s*	129	648	27:8.5	282
(50 *s*)	100 *s*	130	648	26:9	278
Down	All	75	1,128	4:30	408
Up	20 *s*	65	328	9:12.7	146
	30 *s*	70	496	9:12.2	172
	40 *s*	74	552	10:11.5	194
	100 *s*	78	552	12:11	194

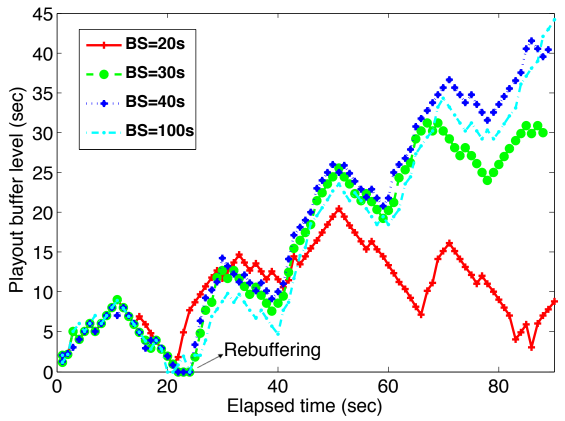
60 s. There is another case where the bitrate changes from 100 kb/s (10 *s*) ➜ 200 kb/s (10 *s*) ➜ 100 kb/s (10 *s*) ➜ 200 kb/s (10 *s*) ➜ 100 kb/s (10 *s*) ➜ 200 kb/s (10 *s*) in 60 *s*. For both cases, the average played bitrate and number of bitrate changes are the same. Regarding video QoE, however, the first case is better because of the smaller number of bitrate changes. To clearly represent the differences, the total number of bitrate changes and the average length of played time before the bitrate switches are taken into consideration in the fifth column in Table 6.3.



(a) [***High*** (10 *s*) **-** ***Low*** (10 *s*)] Selected bitrates with $BS = 20\,s$ and $BS = 30\,s$



(b) [***High*** (10 *s*) **-** ***Low*** (10 *s*)] Selected bitrates with $BS = 40\,s$ and $BS = 100\,s$



(c) [***High*** (10 *s*) **-** ***Low*** (10 *s*)] Remaining buffer level (second) as time elapsed

Figure 6.3: Monitoring selected bitrates and remaining buffer levels under *High* (10 *s*) - *Low* (10 *s*)

Exp. 1) *High* (10 s) and *Low* (10 s) cycle: First, we observe that a rebuffering appears only at the end of the first cycle around elapsed time $t = 20\,s$ in Figure 6.3c. After that, no additional rebufferings occur for the rest of the playback. We observe this behavior because the video player rapidly reaches the high bitrates in the first few seconds (e.g., less than $5\,s$ based on Figure 6.3a and 6.3b), regardless of the remaining playout buffer level. Since it downloads high bitrates before saving enough video data in the buffer, the player most likely goes through rebufferings after the available bandwidth is throttled at $t = 10\,s$. For this reason, all players experience short rebufferings at $t = 20\,s$ as shown in Figure 6.3c.

Based on the *High* (10 s) and *Low* (10 s) results in Table 6.3, we observe that a large buffer shows fewer bitrate switches and lower average bitrate changes (the last column in Table 6.3). According to Table 6.2, the player with $BS = 20\,s$ downloaded more segments with 991 kb/s compared to other players with $BS = 30\,s$, $40\,s$ and $100\,s$. The players with large buffers tried to fill up the buffers quickly by requesting lower bitrates (e.g., 688 kb/s or 477 kb/s) after experiencing the throttled network conditions. Since buffer is time-based, in other words, the low bitrates allows the player to fill seconds of buffer space more quickly for the same bandwidth. We observe this behavior at every bandwidth cycle (around time $t = 20\,s$, $40\,s$, $60\,s$ and $80\,s$). It explains why the players with large buffers downloaded more segments but provided lower average played bitrate (fourth column in Table 6.3). For instance, the large buffer downloaded two segments ($B = 477\,kb/s$) while the small buffer was downloading the single segment with 991 kb/s for the same period of time. This reflects the experimental results in Figure 6.3c where the remaining buffer levels of three cases, except $BS = 20\,s$, increase as time elapsed. The players with large buffers tried to fill up the buffers as much as they could with lower bitrate segments during the *High* period.

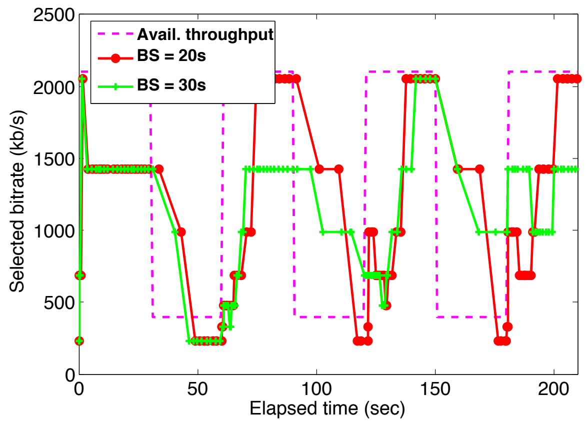
There are two possible reasons to why large buffers download such low bitrates during playback. The first is that 10 s of *High* period may not be sufficient for the large buffers to reach the `Upper` threshold[2]. On the other hand, 10 s of a high-bandwidth time period is enough for $BS = 20\,s$ to save data above the `Upper` threshold in the buffer and allow the player to increase the bitrate more quickly than the players with large buffers (Figure 6.3a). The

[2]According to Chapter 2, an ABR player takes into account three pre-defined thresholds (`Upper`, `Lower`, and `Panic`) when it changes state (`Buffering` or `Steady`) and picks the best available bitrate during playback.

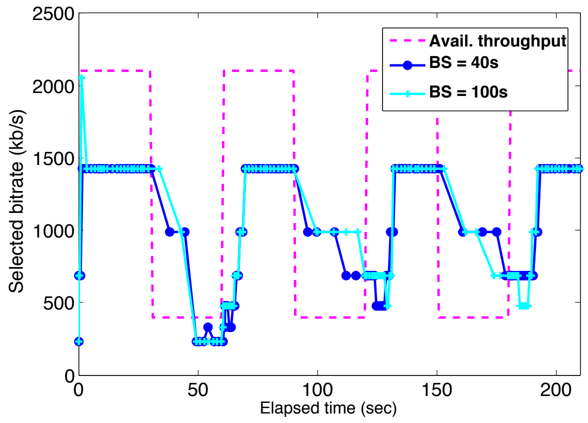
second is that the bandwidth estimator may prevent the player from increasing bitrate despite the large remaining buffer. As described in Figure 2.1, the player increases the bitrate when the estimated network bandwidth is larger than the next-higher bitrate. During the experiments, we find that a large buffer uses longer time periods to measure the network bandwidth than a small buffer. In the middle of the measurement by the bandwidth estimator, the available throughput is suddenly throttled. The average download speed during the period is lower than the next-higher bitrate and the player does not increase the bitrate.

Exp. 2) *High* ($30\,s$) and *Low* ($30\,s$) cycle: During the experiment, we find that the player with $BS = 20\,s$ leads to a lower average played bitrate and a higher number of average bitrate changes, compared to the players with larger buffers (Table 6.3). The reasons can be explained as follows. We first observe that the player with $BS = 20\,s$ often requested 2,056 kb/s during the *High* period (Figure 6.4a). As we pointed out in Exp. 1, $30\,s$ is sufficient for the player to fill up the data above the `Upper` threshold and decide whether to increase or decrease the bitrate regarding the current playout buffer level. We clearly observe this behavior in Figure 6.4c. The player with $BS = 20\,s$ gets into `Steady` state around elapsed time $t = 70\,s$, $130\,s$ and $190\,s$, showing a relatively flat line on each edge. During `Steady` state, it periodically (every two seconds) requested a video segment with 2,056 kb/s. In the middle of the download, the available bandwidth was throttled around elapsed time $t = 90\,s$ and $150\,s$. To downgrade the bitrate, the player had to wait until the requested segment with 2,056 kb/s was completely downloaded. However, the player consumed most data in the buffer before it downloaded the entire segment. As a result, it failed to fill up the buffer with new data due to the lack of available bandwidth. This behavior led to repeated rebufferings at every cycle (around elapsed time $t = 120\,s$ and $180\,s$ in Figure 6.4c). When it experienced such rebufferings, it directly switched to the lowest bitrate (233 kb/s). Overall, this is the cause of the lower average played bitrate and the higher bitrate changes for $BS = 20\,s$. Even if the players with $BS = 40\,s$ and $BS = 100\,s$ request for 1,427 kb/s, the $30\,s$ of *High* period is not long enough to reach 2,056 kb/s (Figure 6.4b) owing to the same reasons described in the previous experiment.

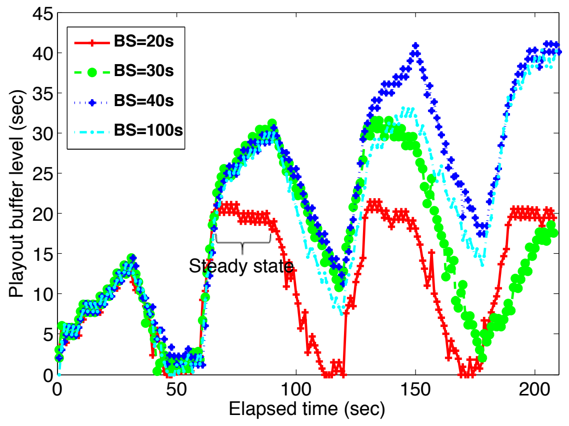
Further, we find that the Microsoft's SS player requests the high bitrates at the beginning of



(a) [***High*** $(30\,s)$ **-** ***Low*** $(30\,s)$] Selected bitrates with $BS = 20\,s$ and $BS = 30\,s$



(b) [***High*** $(30\,s)$ **-** ***Low*** $(30\,s)$] Selected bitrates with $BS = 40\,s$ and $BS = 100\,s$



(c) [***High*** $(30\,s)$ **-** ***Low*** $(30\,s)$] Remaining buffer level (second) as time elapsed

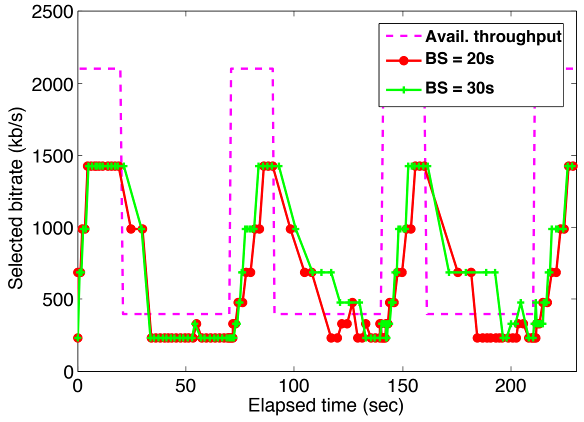
Figure 6.4: Monitoring selected bitrates and remaining buffer levels under *High* $(30\,s)$ - *Low* $(30\,s)$

Table 6.4: Compare average elapsed times (second) from 230 kb/s to 1,427 kb/s in Exp. 2

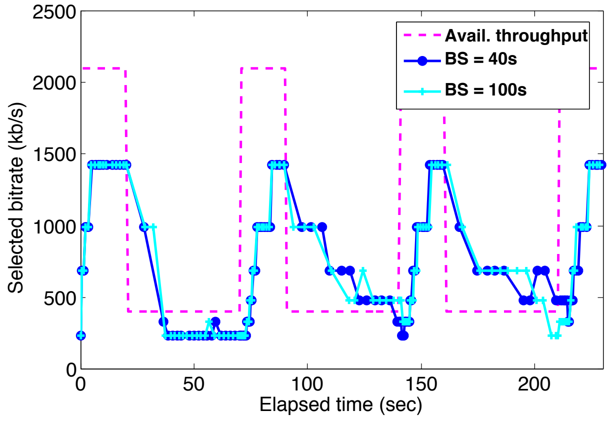
Maximum BS	First cycle	Rest of the cycles
20 s	5.08 s	15.29 s
30 s	4.19 s	14.06 s
40 s	4.07 s	13.02 s
100 s	4.33 s	12.38 s

a playback. This causes rebufferings around elapsed time $t = 50\,s$ as shown in Figure 6.4c. To verify this, we compared the elapsed times until the bitrate changed to 1,427 kb/s from 230 kb/s at the beginning of each *High* period. The experimental results in Table 6.4 show that the elapsed time for the first bandwidth cycle is relatively shorter than the times for next cycles. Also, we observe that the increased speed of the remaining buffer level in the first cycle is relatively slower than the speed in other bandwidth cycles (Figure 6.4c). This indicates that the ABR player is reaching higher bitrates quickly at the beginning of a playback while it focuses more on filling up the playout buffer by requesting low bitrates during the rest of *High* cycles. This behavior may cause unnecessary rebufferings at the beginning of a playback. As an example, Figure 6.3c shows a single rebuffering event occurring only at the end of the first cycle. Then no rebufferings appear again throughout the rest of the cycles.

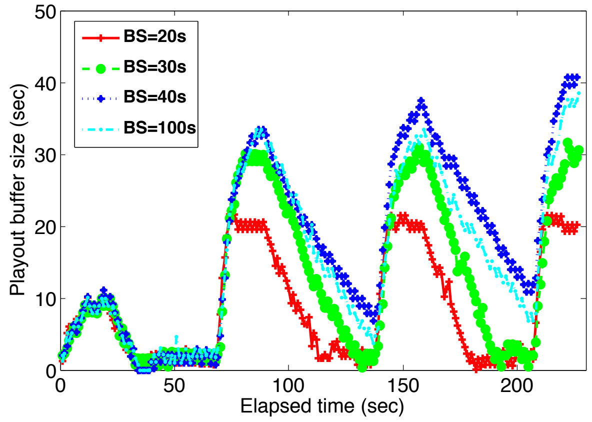
Exp. 3) Asymmetric *High* (20 s) and *Low* (50 s) cycle: In Figure 6.5a, the player with $BS = 30\,s$ gradually downgrades the bitrate since it has already stored enough data in the buffer. However, to maintain a full buffer, the player with $BS = 20\,s$ decreases the bitrate a bit more quickly, responding to the suddenly throttled network conditions. On the other hand, when the network condition switches from *Low* to *High* period, the player with $BS = 30\,s$ gets to the high bitrate in slightly shorter time than the player with $BS = 20\,s$. This contrasts with the previous experiments where a large buffer is reluctant to increase bitrate to fill up its buffer by requesting lower bitrates. The main reason is that the player has already stored enough video data with lower bitrates during the 50 s of *Low* period. The total amount of video content in the buffer is sufficiently large for the player to promptly



(a) [***High*** $(20\,s)$ **-** ***Low*** $(50\,s)$] Selected bitrates with $BS = 20\,s$ and $BS = 30\,s$



(b) [***High*** $(20\,s)$ **-** ***Low*** $(50\,s)$] Selected bitrates with $BS = 40\,s$ and $BS = 100\,s$



(c) [***High*** $(20\,s)$ **-** ***Low*** $(50\,s)$] Remaining buffer level (second) as time elapsed

Figure 6.5: Monitoring selected bitrates and remaining buffer levels under *High* $(20\,s)$ - *Low* $(50\,s)$

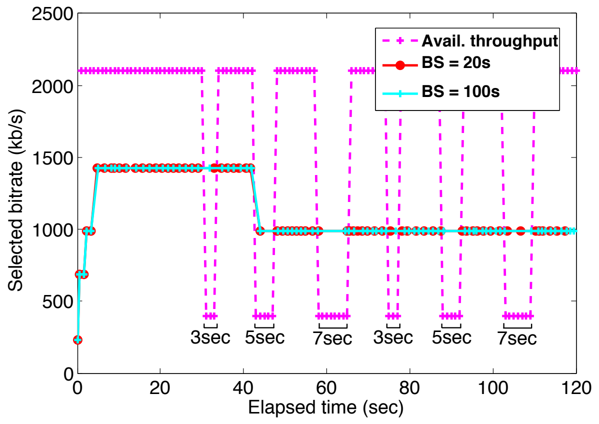
increase bitrate when the available bandwidth is suddenly increased. It will play the high bitrate after it consumes all the low bitrate segments.

Exp. 4) Short-term *Up* and *Down* spikes: We created short-term spikes of available bandwidth, where positive (*Up*) or negative (*Down*) spikes occur for a few seconds (e.g., $3\,s$, $5\,s$ and $7\,s$ as shown in Figure 6.6a and 6.6b). For *Down* spikes, all video players show the same behavior regardless of its playout buffer size (Figure 6.6a). They do not rapidly throttle down the bitrate since the remaining buffer levels are sufficiently large during the *Down* spikes. For *Up* spikes, the player with a large buffer (e.g., $BS = 40\,s$ and $BS = 100\,s$) is more likely to download a higher bitrate. The reason for this behavior is that the large buffer has stored enough video data in the first $30\,s$ at the beginning (Figure 6.6c), and the measured network throughput by the bandwidth estimator is larger than 688 kb/s or 991 kb/s. Thus, it does not need to keep the low bitrate responding to the *Up* spikes. On the other hand, the player with a small buffer (e.g., $BS = 20\,s$ and $BS = 30\,s$) does not try to take the risk (Figure 6.6b). To keep a full buffer, a small buffer seldom increases the bitrate during the *Up* spikes. This behavior causes a slightly increased number of bitrate changes for a large buffer (e.g., $BS = 40\,s$ and $BS = 100\,s$), as described in Table 6.3.

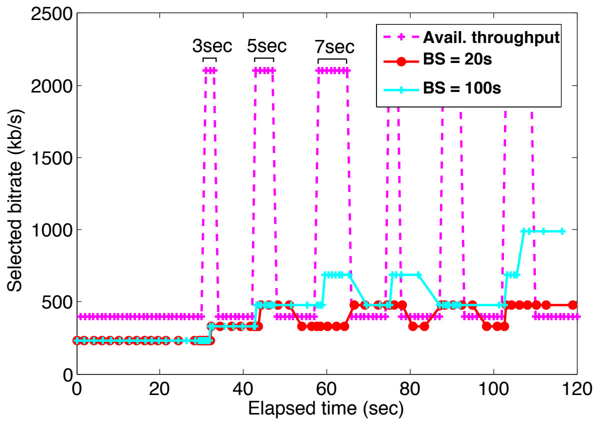
During the *Up* spike experiment, we find that all video players experience about $3\,s$ of rebuffering at the beginning of the playback, regardless of its playout buffer size (Figure 6.6c). Unlike the prior experiments where the available bandwidth is sufficiently large, this experiment shows that the throttled network conditions at the beginning prevent the players from quickly filling up initial buffers and increasing bitrates. At the beginning of a playback, this leads to a series of short rebufferings between elapsed time $t = 1\,s$ and $t = 15\,s$ in Figure 6.6c.

6.4 The impact of playout buffer size on video QoE in ABR streaming

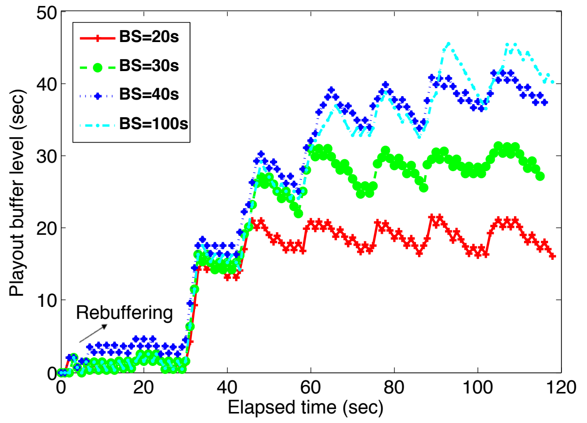
Through Section 6.3, we observe that a bitrate switching behavior in ABR streaming varies based on the remaining buffer level and the maximum playout buffer size in the player. We



(a) [***Down* spikes**] Selected bitrates with $BS = 20\,s$ and $BS = 100\,s$



(b) [***Up* spikes**] Selected bitrates with $BS = 20\,s$ and $BS = 100\,s$



(c) [***Up* spikes**] Remaining buffer level (second) as time elapsed

Figure 6.6: Monitoring selected bitrates and remaining buffer levels under *Down* and *Up*

use an online crowdsourcing platform to analyze the impact of playout buffer size on human perception of video quality.

6.4.1 Online crowdsourcing platform

We performed a video QoE survey using Amazon's Mechanical Turk [123]. The survey was classified by Columbia University's IRB as human subjects research (IRB-AAAO5906). The Mechanical Turk system gives employers (called *requesters*) a way to post human intelligence tasks on the Internet for employees (called *workers*) to tackle in exchange for a wage. For our QoE experiments, the participants were required to watch our recorded videos and took a survey about their viewing experience during playback.

Video sources: One possible way of monitoring video QoE is to operate our own streaming server and require participants to download videos from the server using ABR players with different playout buffer sizes. But this can cause unexpected results due to uncontrolled last mile network conditions of the participants during playback. So instead, we first recorded the video screen in our laboratory while the players played the test video (Table 6.1) under the same network conditions. Every video was recorded as high resolution and each recorded video reflected distinct playback behavior (e.g., rebufferings and bitrate changes) regarding the maximum playout buffer size that we figured out in Section 6.3. Then, we uploaded three sets of video files to the crowdsourcing platform. During the survey, one set was randomly assigned to each participant, and the sequence of the videos was also randomly ordered. All participants were required to download the videos locally on their computers before the survey and after watching the videos, they answered questions regarding their viewing experience using general video players (e.g., PotPlayer, GomPlayer and VLC). Table 6.5 and Figure 6.7 show the bitrate switching behaviors and video downloading statistics of the recorded videos used in our QoE survey.

Participants: 215 people participated in our QoE survey. They are categorized into two groups. 177 people in the first group were randomly recruited via Amazon's Mechanical Turk. In order to avoid inaccurate data from participants who did not pay attention during the playback, we included questions that only the participants who followed the instructions

Table 6.5: Video downloading statistics for QoE experiments

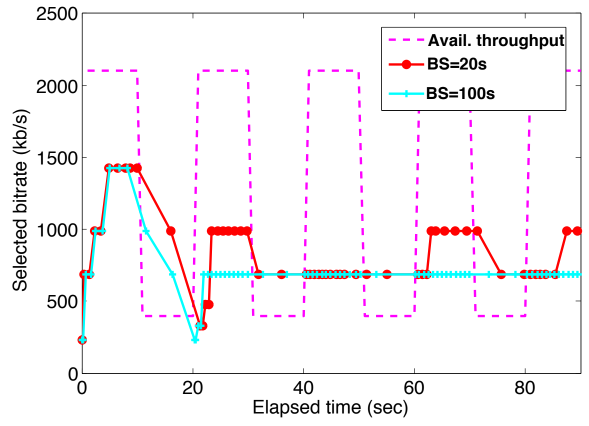
No.	Max. BS	Num. of segments	Avg. bitrate (kb/s)	Num. of bitrate changes : length (second)	Avg. variance (kb/s)
Set 1	20s	56	816	11:10	375
	100s	71	696	8:11.3	301
Set 2	20s	114	832	29:7	420
	30s	117	920	20:10.4	325
	100s	127	917	18:11.5	321
Set 3	20s	62	296	10:11.2	126
	100s	81	568	11:10.9	194

Table 6.6: MOS for QoE experiments

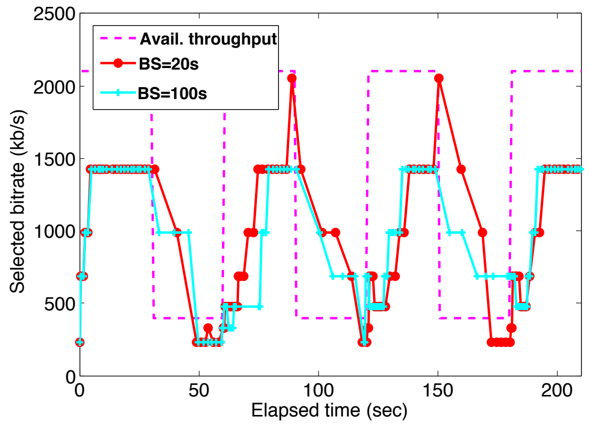
Score	Description
5	Perfect, watched without any viewing interference
4	Good, but imperfections are perceived
3	Slightly annoying
2	Annoying
1	Very annoying, nearly impossible to watch

and watched the videos carefully could answer. Those who failed to give correct answers to these specific questions were not included in the total number of participants. The second group is consist of 38 engineers, and some of them specialize in video streaming. This group gave us more comprehensive feedback on our video QoE experiments through our website[3]. In addition to the MOS question described in Table 6.6, we asked these participants which factor (e.g., rebufferings and bitrate changes) interrupted their viewing experiences the most during playback.

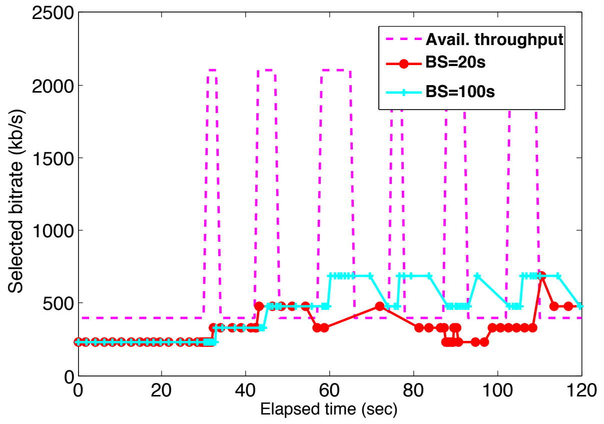
[3]We created a website for the QoE survey. It is no longer available.



(a) **[Set 1]** Under *High* (10 *s*) - *Low* (10 *s*)



(b) **[Set 2]** Under *High* (30 *s*) - *Low* (30 *s*)



(c) **[Set 3]** Under short-term *Up* spikes

Figure 6.7: Video sources for QoE experiments: selected bitrates with $BS = 20\,s$ and $BS = 100\,s$

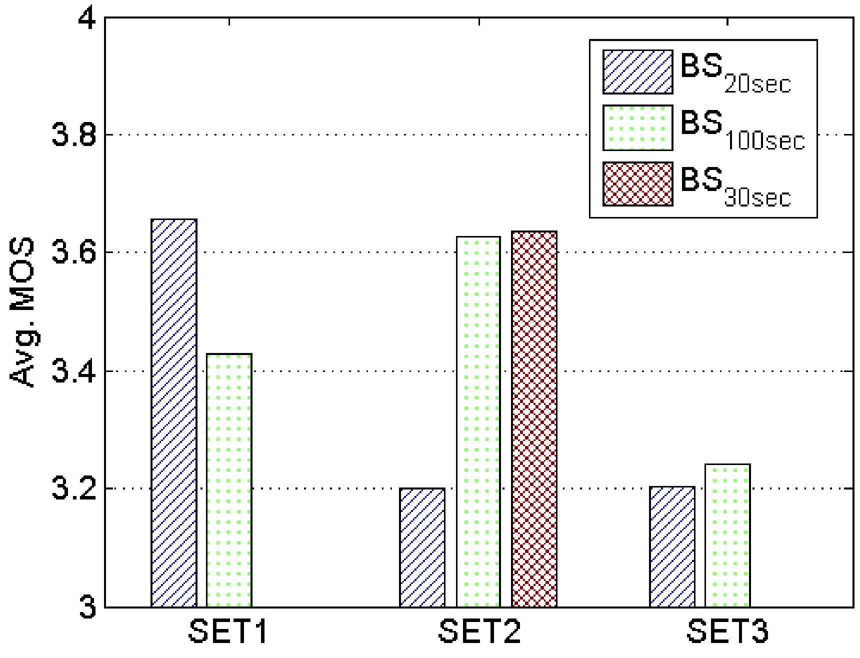
6.4.2 QoE survey results

We first note that there is no significant difference between the results collected from the two groups. Through our QoE survey, we focus on finding the root cause of the participants' disturbed viewing experience and the impact of playout buffer size under the given network conditions.

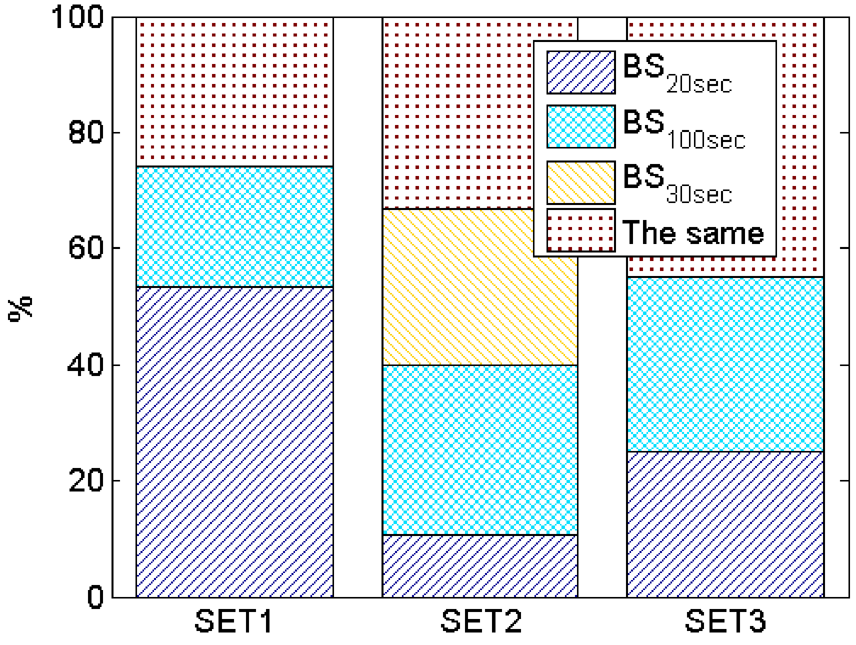
Set 1) *High* $(10\,s)$ and *Low* $(10\,s)$ cycle: Data was collected from 62 participants. In Set 1, there are two different playout buffer sizes ($BS = 20\,s$ and $BS = 100\,s$). As described in Figure 6.7a and Table 6.5, the video player with $BS = 20\,s$ tries to download a higher bitrate and switches bitrate more often, compared to the player with $BS = 100\,s$. There are no significant rebufferings for both cases (less than $1\,s$). A rebuffering only occurs right after the first cycle around time $t = 20s$ (Finding 1 in Section 6.4.3). After the first bandwidth cycle, the $BS = 20\,s$ configuration often switches between 688 kb/s and 991 kb/s while the $BS = 100\,s$ setting maintains 688 kb/s to the end. We assumed that the frequent bitrate changes by the $BS = 20\,s$ setting would lead more severe viewing interruption. However, interestingly, the bitrate changes caused by $BS = 20\,s$ are ignored by many participants, and they preferred $BS = 20\,s$ over $BS = 100\,s$ (Set 1 results in Figure 6.8b). In this case, a small number of bitrate changes in the middle of the playback did not degrade video QoE, and the higher bitrate provided by $BS = 20\,s$ led to slightly better viewing experience.

Set 2) *High* $(30\,s)$ and *Low* $(30\,s)$ cycle: 75 people participated in Set 2. Unlike the previous experiment, we uploaded three videos with $BS = 20\,s$, $30s$ and $100s$. For $BS = 20\,s$, the $20\,s$ of playout buffer size is not enough to avoid rebufferings during $30\,s$ of *Low* period. As a result, it causes $7\,s$ of rebufferings during the entire playback. The frequent bitrate changes also annoyed the participants, but the participants gave the lowest score to the video played with $BS = 20\,s$ due to the rebufferings, (Figure 6.8a). $BS = 30\,s$ and $BS = 100\,s$ cause a relatively shorter period of rebufferings (less than $0.5\,s$) and show similar behaviors regarding the number of bitrate changes and its average bitrate changes (Table 6.5). As shown in Figure 6.8a and 6.8b, most participants did not recognize the difference between $BS = 30\,s$ and $BS = 100\,s$.

Set 3) Short-term *Up* spikes: The total number of participants for Set 3 is 78. Both



(a) What is your score for viewing the video?



(b) Overall, which video offered you the best viewing experience in each set?

Figure 6.8: QoE survey results - MOS and preference in each set

$BS = 20\,s$ and $BS = 100\,s$ are too conservative to request high bitrates such as 1,427 kb/s and 2,056 kb/s (Figure 6.7c). The players cause short periods of rebufferings at the beginning (3 s for both $BS = 20\,s$ and $BS = 100\,s$) due to the throttled network conditions when they start downloading the video. Then, there are no more rebufferings until the end. This causes some participants to complain about the long start-up delay. During the experiments, $BS = 100\,s$ switches bitrates mostly between 688 kb/s and 477 kb/s while $BS = 20\,s$ switches between 477 kb/s and 331 kb/s, reacting to the short spikes (Figure 6.7c). According to our QoE survey (Figure 6.8b), the higher played bitrate caused by $BS = 100\,s$ results in slightly better video QoE. Compared to the previous experiments, the participants in Set 3 provide relatively low scores owing to the lower average experienced bitrates.

6.4.3 Summary of key observations

Table 6.7 briefly shows the comparison of the performance between small and large buffers based on the experimental results in Section 6.3. We note that the terms (small, large, `High`, `Medium` and `Low`) used in Table 6.7 are relative values and not absolute. The fast varying network conditions represent the cases where the duration of bandwidth shortage is less than the maximum size of playout buffer. On the other hand, the slowly varying network conditions indicate the cases where the duration of bandwidth shortage is longer than the maximum size of playout buffer. For instance, the player with $BS = 20\,s$ experienced frequent rebufferings under slowly varying network conditions such as *High* (30 s) and *Low* (30 s). In the short-term *Up* spike experiments, we observe frequent short rebufferings at the beginning of playback (Exp. 4 in Section 6.3) because the player requested the high bitrates that require higher bandwidth than what is available in the network. In general, we find no rebufferings for all buffer sizes under such short-term spikes. Below, we summarize our key findings:

Finding 1: The Microsoft's SS player may cause unnecessary rebufferings by requesting high bitrates at the beginning of a playback. As shown in Table 6.4, the ABR player tries to increase bitrates quickly at the beginning of a playback. This may cause unnecessary rebufferings in a short period of time when the network experiences congestion.

Table 6.7: Overview of playout buffer size experiments

Under fast varying network conditions: Small BS $>$ duration of bandwidth shortage				
BS	Rebuffering	Avg. bitrate	Num. of bitrate changes	Avg. variance
Small	No	`High`	`Medium`	`High`
Large	No	`Medium`	`Low`	`Medium`

Under slowly varying network conditions: Small BS $<$ duration of bandwidth shortage $<$ Large BS				
BS	Rebuffering	Avg. bitrate	Num. of bitrate changes	Avg. variance
Small	Yes	`Medium`	`High`	`High`
Large	No	`High`	`Medium`	`Medium`

Under short-term *Down* spikes Small BS $\gg$ duration of bandwidth shortage				
BS	Rebuffering	Avg. bitrate	Num. of bitrate changes	Avg. variance
Both	No	`Medium`	`Low`	`Low`

Under short-term *Up* spikes Small BS $\gg$ duration of bandwidth shortage				
BS	Rebuffering	Avg. bitrate	Num. of bitrate changes	Avg. variance
Small	No	`Low`	`Medium`	`Low`
Large	No	`Medium`	`Medium`	`Medium`

Finding 2: The time periods to estimate download speed by bandwidth estimators vary depending on the maximum playout buffer size. The bandwidth estimator measures available throughput during a download and its result directly affects the bitrate switching behaviors in ABR streaming. As we described in Section 6.3, the estimator in the player using a large buffer uses a larger window size to measure the throughput than the one with a small buffer. Therefore, it is likely to avoid frequent bitrate changes under fluctuating network conditions. However, it may also prevent the players from switching to higher bitrates quickly even if it has enough content in the buffer (Exp. 1 and 2 in Section 6.3).

Finding 3: A large buffer does not always achieve higher QoE especially when available throughput increases after bandwidth shortages. During the experiments, we show that a large buffer typically provides fewer bitrate changes and rebufferings than a small buffer. However, our study shows that a small buffer can achieve higher QoE by providing higher played bitrates more quickly than a large buffer when available throughput increases after bandwidth shortages. In this case, a small buffer can reach the `Upper` threshold more quickly than a large buffer. On the other hand, the large buffer tries to fill up its buffer by requesting lower bitrates to reach the `Upper` threshold. This switching behavior is followed by the large buffer causing lower average played bitrate during playback (Table 6.5), and resultantly degrading video QoE (Figure 6.8a and 6.8b).

6.5 Adaptive playout buffer size

Throughout Section 6.3 and 6.4, we observe that a large buffer reduces the number of rebufferings and achieves higher QoE (Set 2 in QoE survey) under slowly varying network conditions. On the contrary, during fast varying network conditions we find that a small buffer can achieve higher QoE by offering higher played bitrates, despite slightly higher amplitude and increased number of bitrate changes (Set 1 in QoE survey). As other researchers [19; 124; 125; 126] agree, our experimental results indicate that rebufferings should be avoided all times to enhance viewing experiences. Some papers [127; 128] address that constant bitrate is typically preferred to frequent bitrates changes. However, the number of bitrate changes can be less significant when the player provides higher bitrates quickly after bandwidth shortages. According to our QoE survey results (Figure 6.8), the viewers preferred this behavior rather than being stuck with lower bitrates for a long time.

Our experimental results tell us that playout buffer size directly affects bitrate switching behaviors. This inspires the following question: can we find an optimal buffer size? However, we believe that modeling an optimal buffer size for video streaming is difficult since various QoE metrics need to be monitored during the entire playback. For instance, it can vary depending on when or how long rebufferings appear, when or how much the bitrate is increased or decreased and how long the viewer watches the video. As Garcia et al. [129]

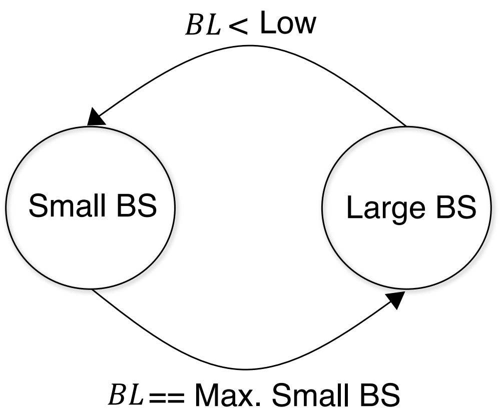


Figure 6.9: Adaptive playout buffer size - dynamically switching between small and large buffers during playback

summarize in their paper, there are no quantitative metrics that take into account all the above metrics for QoE evaluation. Instead, we propose that an ABR player should adapt its playout buffer size (e.g., between small and large buffers) while a video is being played. Therefore, the question is, how does the player track network conditions (e.g., fast or slowly varying) accurately and change the buffer size properly? For better channel capacity estimation, the player needs to precisely measure the incoming throughput and its variation during a download. But, due to estimating errors caused by the competing TCP flows [88] and TCP slow start [130], it is difficult to monitor the available throughput accurately with a bandwidth estimator. In addition, there are challenges to finding the optimal window size for the estimator to analyze the network conditions [130]. For these reasons, instead of relying on the bandwidth estimator, we suggest to monitor a playout buffer level in real time for the analysis of network conditions during playback. Other papers [131; 116; 132] propose that players should mainly use the remaining buffer occupancy to decide whether to increase or decrease the bitrate during playback. The key idea is to find the proper switching point for buffers of fixed size. However, their implementation and the parameters used in the switching mechanism can be very complicated depending on the player's maximum buffer size. As we observed in Section 6.3 and 6.4, for example, even though the remaining buffer level is the same, we may set up different ABR heuristics based on the player's maximum buffer size to better handle the network conditions.

Table 6.8: Buffer thresholds used in our testbed

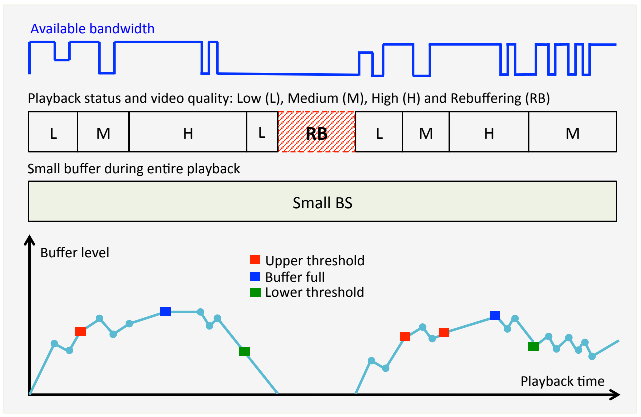
BS	`Panic`	`Low`	`Upper`	`Max`
Small	$7\,s$	$12\,s$	$17\,s$	$20\,s$
Large	$7\,s$	$15\,s$	$25\,s$	$100\,s$

In our proposed platform, an ABR player tries to reach high bitrates quickly and store the video segments as much as it can to protect against possible bandwidth shortage. To more easily describe our solution, we demonstrate the possible scenarios in Figure 6.10. The player first starts with a small buffer to reach the `Upper` threshold and provide high bitrates more quickly. When the playout buffer level becomes full, it switches to a large buffer to store more segments in the buffer to guard against possible bandwidth shortages later. The player maintains the large buffer since it typically provides fewer bitrate changes and rebufferings under fluctuating network conditions. When there is a long-term bandwidth shortage that causes rebufferings or the current buffer level is below the `Low` threshold, it changes the buffer size to small again. Figure 6.9 shows the FSM of buffer size switching behavior in our proposed platform.

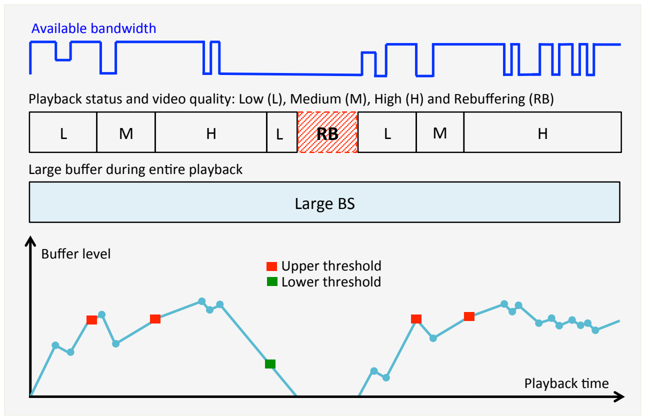
6.5.1 Evaluation

The existing Microsoft's SS platform does not allow to flexibly change the playout buffer size and the buffer thresholds while a video is being played. To show the feasibility of our approaches, we created our own video players that are capable of downloading video segments from the IIS streaming server, but they cannot actually play the video on the screen because they do not have any decoding functionalities used in Microsoft's SS. The players are designed to follow the same bitrate switching rules described in Figure 2.1.

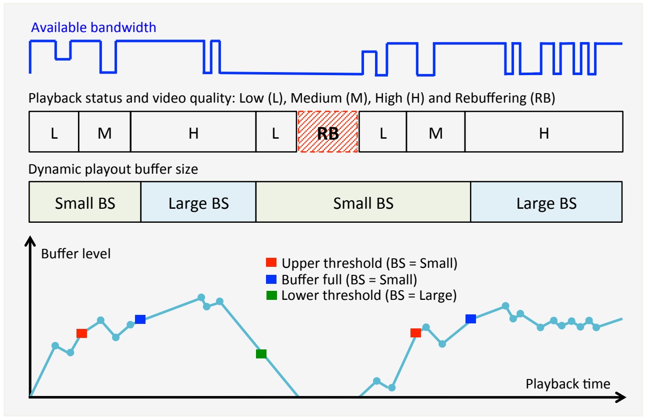
Figure 6.11 shows our testbed setups in Wi-Fi networks. During the experiments, we tested the three players with different playout buffer sizes (small, large or dynamically switching among two buffers) separately. The buffer thresholds for small and large buffers are described in Table 6.8. The bandwidth estimator for the player with the small buffer measured the network throughput over the last three segments while the estimator for the


Available bandwidth
Playback status and video quality: Low (L), Medium (M), High (H) and Rebuffering (RB)
L
M
H
L
RB
L
M
H
M
Small buffer during entire playback
Small BS
Buffer level
Upper threshold
Buffer full
Lower threshold
Playback time

(a) Small buffer


Available bandwidth
Playback status and video quality: Low (L), Medium (M), High (H) and Rebuffering (RB)
L
M
H
L
RB
L
M
H
Large buffer during entire playback
Large BS
Buffer level
Upper threshold
Lower threshold
Playback time

(b) Large buffer


Available bandwidth
Playback status and video quality: Low (L), Medium (M), High (H) and Rebuffering (RB)
L
M
H
L
RB
L
M
H
Dynamic playout buffer size
Small BS
Large BS
Small BS
Large BS
Buffer level
Upper threshold (BS = Small)
Buffer full (BS = Small)
Lower threshold (BS = Large)
Playback time

(c) Switching between small and large buffers during playback

Figure 6.10: Playback status and played bitrate changes depending on maximum playout buffer size

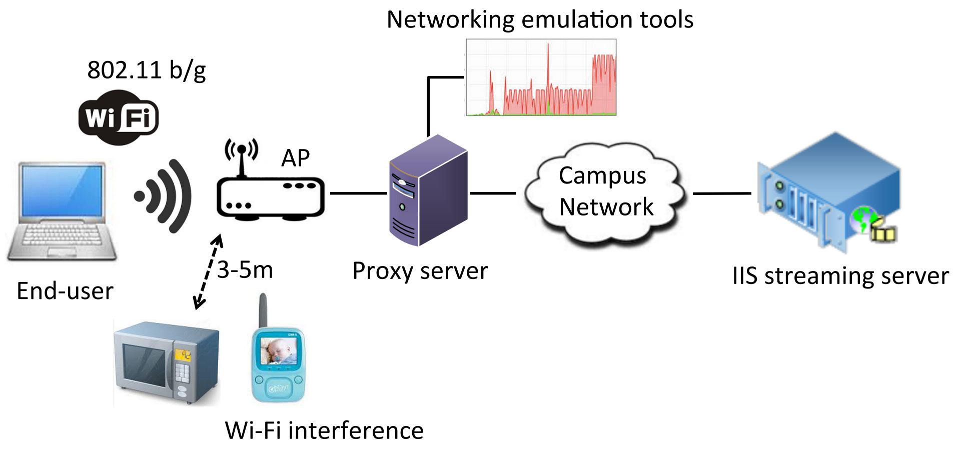


Figure 6.11: Testbed setups for evaluation

large buffer measured it over the last five segments during playback. We note that the buffer thresholds and parameters are used in existing Microsoft's SS players and obtained from the experimental results in Section 6.3 and 6.4. Depending on network conditions, the small and large buffers show distinct behaviors that we summarized in Table 6.7. The players switched among three different bitrates (`Low` - 331 kb/s, `Medium` - 688 kb/s, and `High` - 2,056 kb/s) during the download, and the playback duration of each segment is two seconds.

We emulated two different network conditions: short-term *Down* spikes and fast or slowly varying network conditions. To create the real short-term *Down* spikes, we periodically turned on and off home networking devices (e.g., baby monitors and microwave ovens) while the players downloaded the segments for 120 *s*. We separately tested each player ten times, and the average playback statistics are shown in Table 6.9. All players displayed similar behaviors regardless of their maximum buffer sizes - no rebufferings, few bitrate changes and downloading `medium` bitrates most against the short-term *Down* spikes. This reflects our experimental results in Section 6.3.

In addition, we measured the playback statistics while the players downloaded the video for 600 *s* under fast or slowly varying network conditions. During the download, we randomly selected and switched the network conditions every 120 *s* (e.g., fast ➜ slow ➜ slow ➜ fast ➜ fast). To emulate the varying network conditions, we periodically throttled (less than

`Low` bitrate) and released (higher than `High` bitrate) the available throughput in the network using NetLimiter [119] and Fiddler [120]. Under fast varying network conditions, the duration of bandwidth shortage did not last longer than the small buffer size ($< 20\,s$). For slowly varying network conditions, the bandwidth shortage duration was randomly set to between 20 to 70 s. Therefore, if the remaining buffer level is higher than the shortage duration, the large buffer may provide smooth streaming while the small buffer experiences rebufferings. Each player separately downloaded the video twenty times. As shown in Table 6.9, the small buffer caused longer rebuffering durations and more bitrate changes during the download. Both the large and adaptive buffers offered shorter duration of rebufferings since they accumulated a large amount of segments in the buffers before the available throughput was throttled. During the experiments, the adaptive buffer downloaded the `High` bitrate 4.5 s earlier than the large buffer when the available throughput increased after the bandwidth shortages. This results in more than 15% higher average downloaded bitrate.

We note that the experimental results can vary regarding the segment duration. As we described in Chapter 2.4, the players may not properly switch the bitrate on time with larger sized segments (e.g., five or ten seconds). For example, let's suppose that the network bandwidth is throttled while the player is downloading a ten second segment. The player cannot switch to lower bitrates until it completely downloads the entire segment or the time-out will occur (Figure 2.1). For this reason, more improvements can be achieved if the segment size is larger than two seconds used in the current testbed setups. Other buffer-aware ABR algorithms [131; 132] that try to find the proper switching point with the finite buffer size may provide more flexible control in bitrate switching. However, our solution is simpler to implement in practice by easily switching playout buffer size during playback. In our testbed setups, an ABR player adaptively changing its maximum playout buffer size depending on the remaining buffer occupancy outperforms the player with finite buffer size, by offering 15% of higher average played bitrate, 70% fewer bitrate changes and 50% shorter rebuffering duration.

6.6 Related work

We address the recent studies on the analysis of ABR streaming performance along with playout buffer size. Tian et al. [131] propose client-side video adaptation algorithms that use a buffer and a PID controller to better estimate network capacity and balance the needs for video rate smoothness and bandwidth utilization in DASH streaming. Huang et al. [116] suggest to find the best appropriate bitrate based on remaining buffer occupancy and use the bandwidth capacity estimation when it is necessary. Cicco et al. [132] propose ELASTIC (fEedback Linearization AdaptIve STreamIng Controller) that filters network capacity and computes the video bitrates to drive the buffer to a set point in bitrate switching.

The prior works agree on the fact that monitoring playout buffer occupancy is useful to estimate network conditions and improve the bitrate switching behaviors in ABR streaming. They track the buffer level to find the right switching point among bitrates in the buffer with fixed size. Our solution is to dynamically change the maximum buffer size depending on the current buffer level during playback. Similarly, we may achieve this by dynamically changing the predefined buffer thresholds (`Upper`, `Lower`, and `Panic`) in the fixed buffer size. Our future work is to empirically compare these techniques depending on various factors such as different segment durations and network conditions.

6.7 Conclusions

We evaluate the impact of playout buffer size in ABR streaming along with video QoE. Throughout our experimental results, we show that bitrate switching behaviors in ABR streaming is dependent not only on the remaining buffer occupancy during playback but also the maximum playout buffer size. Our analysis shows that a large buffer provides relatively higher QoE under slowly varying network conditions. As a key observation, we find that a small buffer can outperform a large buffer by offering higher played bitrates more quickly especially under fast varying network conditions. Based on these findings, we propose an ABR player to not only change bitrates but also switch its maximum playout buffer size adaptively depending on remaining buffer occupancy during a download. In our testbed, we show that an ABR player dynamically switching among small and large buffers can enhance video QoE by providing higher average played bitrate, fewer bitrate changes and shorter rebuffering duration, compared to the players with fixed buffer size.

Table 6.9: Experimental results in our testbed

Network conditions	BS	Number of downloaded segments			Rebuffering duration (second)	Avg. bitrate (kb/s)	Number of bitrate changes
		`Low`	`Medium`	`High`			
Short-term *Down* spikes	Small	8	32	20	0	548	5
	Large	9	30	21	0	556	4
	Adaptive	8	31	21	0	560	4
Fast or slowly varying networks	Small	91	21	149	78	580	20
	Large	61	11	209	38	762	6
	Adaptive	20	11	250	38	880	6

Part IV

Video QoE Monitoring Tool

Chapter 7

QoE Matters More Than QoS: Why People Stop Watching Cat Videos

7.1 Introduction

Today's popular video streaming services such as Netflix, Hulu and YouTube stream video contents to viewers over HTTP or HTTPS. To provide smooth streaming, they use adaptive bitrate (ABR) streaming technologies such as Apple's HTTP Live Streaming (HLS) [37], Microsoft's Smooth Streaming (SS) [16], Adobe's HTTP Dynamic Streaming (HDS) [17] and Dynamic Adaptive Streaming over HTTP (DASH) [38]. In ABR streaming, a video player dynamically adjusts video bitrates based on estimated network conditions, buffer occupancy and hardware specifications of viewers' devices, for example, distinguishing smartphones from desktops. Therefore, user-perceived video quality can vary depending on how appropriately the player selects the best available bitrate during a download. As an example, a viewer may experience frequent rebufferings, where the video is paused and then resumes playing repeatedly, when the player requests a higher bitrate than what is actually available in the network. It is also possible for the viewer to be stuck with a low bitrate during the entire playback if the network capacity is underestimated by the player. Hence, from over-the-top (OTT) video service provider's viewpoint, improving ABR heuristics is a key factor to enhancing video QoE.

To improve ABR streaming, it is important to analyze how changing ABR heuristics influences QoE. While traditional quality of service (QoS) based metrics, such as measuring TCP throughput, video packet delay and jitter, can be used to pinpoint network impairments, the metrics do not accurately reflect the viewer's watching experience. Thus, we believe that the QoE monitoring system should focus on application-layer events instead of transport-layer events. To achieve this, we suggest monitoring live playback events directly from within video players rather than network elements such as routers. As a proof of concept, we have developed YouSlow ("YouTube Too Slow!?"), a new QoE monitoring system for OTT streaming services. This lightweight web browser plug-in can monitor various playback events such as start-up delay, rebufferings and bitrate changes directly from within ABR players while viewers watch videos on the YouTube web site. So far, YouSlow has collected over 1,400,000 YouTube views from more than 1,000 viewers located in more than 110 countries.

In this paper, we evaluate various QoE metrics by analyzing video abandonment rates in YouTube. An abandonment occurs if a viewer closes the video during playback, either due to lack of interest or because they are annoyed by viewing interruptions such as long start-up delay, frequent rebufferings and bitrate changes. Below, we summarize our key findings and contributions:

- **Development of an analysis tool for video QoE:** YouSlow is designed to detect various playback events while a video is being played. Compared to prior approaches using survey-based metrics, YouSlow saves video researchers time and effort, particularly for large sample sizes. In addition, our QoE monitoring system allows viewers to track their viewing experiences such as average played bitrates and rebufferings in real time.

- **An analysis of video QoE in YouTube:** We observe that about 10% of viewers abandoned the videos when the pre-roll ads lasted for 15 seconds. We confirm that the initial buffering has more impact on the video abandonment than the rebufferings in the middle of a playback. Our analysis shows that viewers prefer constant bitrate to increasing bitrate during playback even if the abandonment rate is not significantly different. We show that tracking the rebuffering ratio during playback is useful to quantify abandonment rates for short videos. Our regression analysis using the rebuffering ratio and the number of rebufferings achieves an R-squared value of 0.94 in predicting the video abandonment rate in YouTube.

The remainder of this paper is organized as follows. Section 7.2 describes the overview of YouSlow and its implementation. Then, we present our analysis of YouTube in Section 7.3. Our QoE analysis report is described in Section 7.4. We address challenges of YouSlow platform in Section 7.5 and discuss future work in Section 7.6. Finally, we look at the related work and summarize our conclusions in Section 7.7 and 7.8, respectively.

7.2 YouSlow overview

YouSlow can monitor various playback events directly from within an ABR player for an analysis of video QoE. Currently, YouSlow only supports YouTube, but other players' JavaScript APIs such as Vimeo [133] could be easily added to YouSlow.

7.2.1 Implementation

YouSlow supports three different platforms: The Chrome web browsers, iOS and Android. We have recently released the beta versions of iOS and Android applications on the YouSlow web site[1]. For desktops and laptops, we created a lightweight Chrome plug-in, also known as a Chrome extension [134]. We distribute the YouSlow applications via Chrome web store[2]. The source code is available in GitHub[3].

Figure 7.1 shows the architecture of the Chrome plug-in for YouTube analysis. YouSlow runs in the background of the Chrome browser, and injects our QoE monitoring scripts into the web page whenever a viewer watches a video on the YouTube web site, `www.youtube.`

[1]YouSlow - `https://dyswis.cs.columbia.edu/youslow/`

[2]Chrome web store - `http://goo.gl/AIOED3`

[3]YouSlow GitHub - `https://github.com/leftdal/youslow`

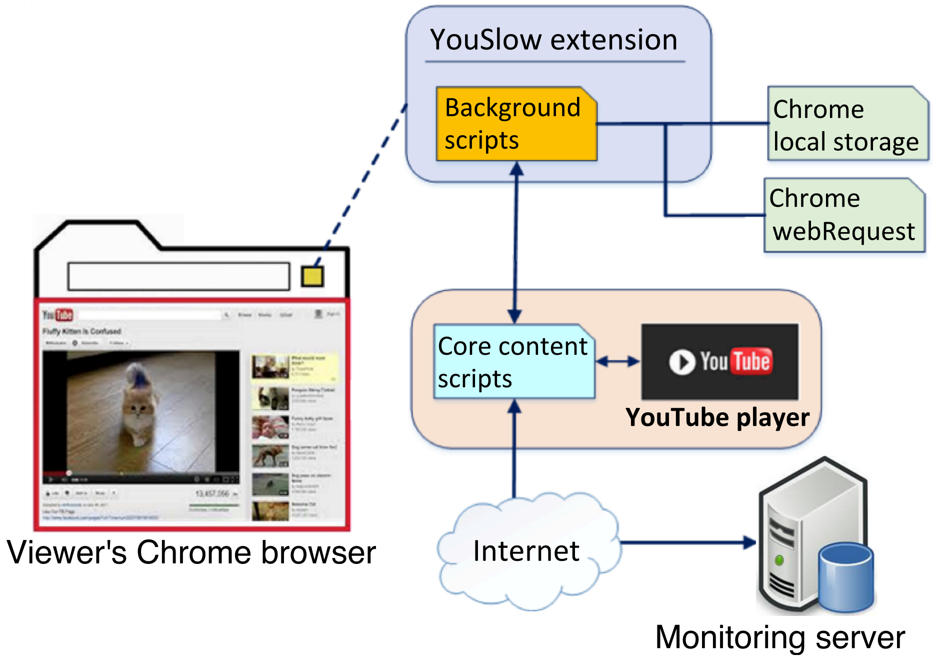


Figure 7.1: Chrome plug-in for YouTube analysis

`com`. The YouSlow scripts contain YouTube player's iframe and JavaScript APIs [135] to access and monitor playback events of HTML5 and Flash video players. When a viewer ends a video session, the extension automatically reports the measurements to our monitoring server[1]. The collected data is analyzed and then marked on Google maps. For privacy reasons, the extension does not collect any information regarding the viewer's YouTube account, video URLs or video titles. Through our monitoring system[1], viewers can monitor various metrics about their YouTube watching experiences, such as how often they experience rebufferings and what video bitrates they typically watch. Using this information, they may compare the performance of their own ISPs with other local ISPs. Additionally, YouSlow outputs can be useful to video service providers to improve their ABR streaming services. For example, they can monitor and compare the rebuffering statistics every time there is a change in their ABR heuristics.

7.2.2 What factors can YouSlow measure?

YouTube deprecated its JavaScript Player API and Flash <object> embeds on January 27th, 2015 [136]. YouTube mainly uses the iframe API to serve an HTML 5 video player using MPEG-DASH (Table 2.4) and avoids a Flash player for mobile devices that do not support Flash [135]. The iframe API allows viewers to control the YouTube player in the web browser: *play*, *pause* and *stop* videos, adjust the player volume, retrieve the information about playback speed and quality changes. YouSlow uses the following functions of the player API:

- **getPlayerState** returns the state of the player such as *unstarted*, *ended*, *playing*, *paused*, *buffering* and *video cued.*
- **getVideoLoadedFraction** shows the percentage of the video that the player shows as buffered.
- **getPlaybackQuality** retrieves the current playing video bitrate.
- **getAvailableQualityLevels** returns the list of video bitrates for the current video.

Additionally, we created the following functions for our YouSlow measurements:

- **getNumOfRebufferings** returns the number of rebufferings that occur in the middle of playback.
- **getRebufferingDuration** returns the total duration of rebufferings (in seconds) that occur in the middle of playback.
- **getResolutionChanges** returns the list of selected bitrates during playback.
- **getPlaybackTime** returns the total amount of time (in seconds) a viewer stays in the video session.
- **getInitialBufferingDuration** returns the total duration of buffering (in seconds) a viewer experiences at the beginning of playback.
- **getAbandonmentStatus** returns the status of video abandonment, *true* or *false.*

- **getAvgHttpLatency** returns average HTTP latency (in milliseconds) measured while a video is being downloaded. We monitor the HTTP request / response time when the ABR player requests a segment via HTTP.

- **getVideoChunks** returns the number of segments that the ABR player has downloaded during playback.

- **getVideoBytes** returns the total amount of video bytes that the ABR player has downloaded during playback.

- **getAdBlockStatus** returns the status of adBlock plug-in installation, *true* or *false*. To detect the adBlock extension on the Chrome web browser, we arbitrarily inject the ad-similar scripts into the web page and check whether the adBlock extension blocks the scripts or not.

7.3 YouTube measurements

We analyzed 1,471,958 YouTube views collected between February 2015 and July 2016 from more than 1,000 viewers in 117 countries. We note that the dataset only includes the video sessions where the viewers watched videos through YouTube web site using the Chrome browser on desktops or laptops. Table 7.1a shows the top ten countries along with the total number of reported views. We also compare and analyze the measurements for different U.S. ISPs (Table 7.1b).

7.3.0.1 Start-up delay

We measure the elapsed time from when a play button is clicked to when the main video starts. There are two factors that contribute to start-up delay: initial buffering and pre-roll ads. For initial buffering, an ABR player typically downloads a few segments (two or three) before it starts to begin playback. The required number of segments depends on ABR configuration. For example, the player may store a larger number of segments (five or ten) to avoid future bandwidth fluctuations at the beginning of playback [137]. Secondly, an ABR player does not play the selected video until viewers have watched the pre-roll

Table 7.1: YouSlow dataset

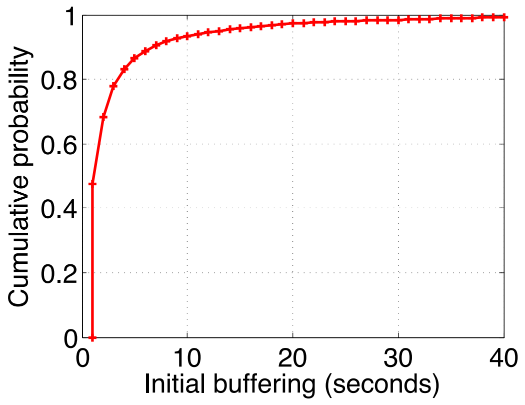
(a) Top 10 countries

Country	Total number of reported views
United States	461,557
South Korea	55,559
United Kingdom	100,748
Indonesia	46,218
India	96,801
Canada	43,864
Malaysia	72,477
Philippines	31,238
Germany	57,876
Italy	29,998

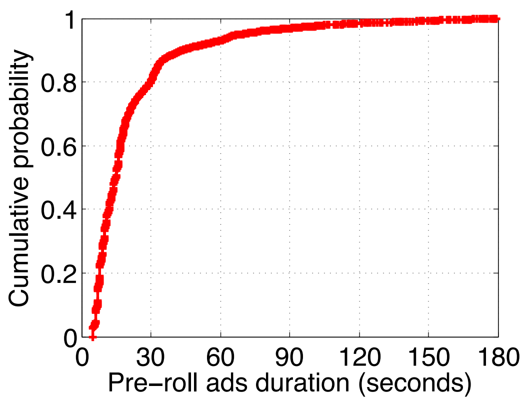
(b) Top 8 U.S. ISPs

U.S. ISP	Total number of reported views
Comcast	92,660
Time Warner Cable	41,873
AT&T	91,231
Qwest Communications	18,708
Verizon	61,141
Century Link	13,396
Charter Communications	49,401
Frontier Communications	13,145

video ad. YouTube's advertising policies [138] describes two types of video ads: *skippable* and *non-skippable*. *Skippable* video ads allow viewers to skip the ad after five seconds. *Non-skippable* video ads must be watched to view the main video and they are usually 15-20 seconds long [139]. Both types of ad can appear before, during or after the main video. YouSlow is not able to distinguish if the ads are *skippable* or *non-skippable*. The ad



(a) Initial buffering duration



(b) Pre-roll ad length

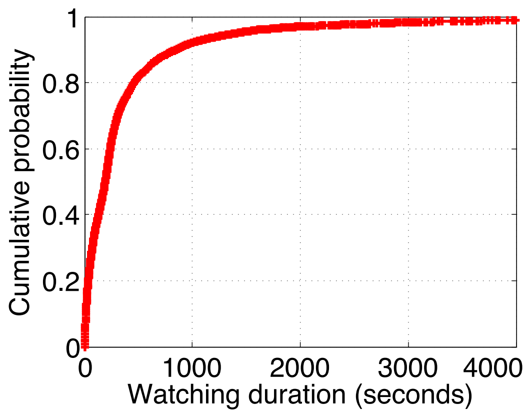
Figure 7.2: Start-up delay caused by initial buffering and pre-roll ad

length recommended by YouTube is less than 3 minutes. The post-roll ads are typically not effective because most viewers close videos once they have watched the main content. The viewers who use an ad-block extension [140] may be able to watch the entire video without ads. We observe that the player uses different URL parameters for downloading the video ads and the main video. To distinguish them, we use the Chrome webRequest API [141]. Currently, YouSlow focuses on the analysis of pre-roll ads in YouTube.

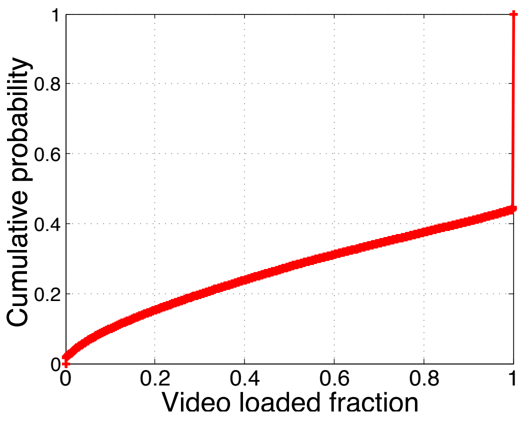
Figure 7.2a presents the cumulative probability of the initial buffering duration. Figure 7.2b shows the cumulative probability of how long the viewers watched the pre-roll ads before the main content. Compared to the pre-roll ads, the initial buffering has a relatively shorter duration.

7.3.0.2 Video watching duration

We measure how long a viewer stays in each video session. The watching duration also includes rebuffering and start-up latency. Based on the experimental results in Figure 7.3a, we observe that the average of watching duration is 6:36 minutes per video session and the median is 2:39 minutes.



(a) YouTube watching duration



(b) Video loaded fraction

Figure 7.3: Video watching duration and video loaded fraction

7.3.0.3 Video loaded fraction

We measure video engagement by monitoring the video loaded fraction described in Section 7.2. According to Figure 7.3b, more than 40% of viewers closed YouTube videos in the middle of the playback. They may have lost their interest in the videos or suffered from unexpected viewing interruptions such as video ads, rebufferings and bitrate changes.

7.3.0.4 Bitrate changes

We observe that most video sessions ($>$ 99%) experience fewer than four bitrate changes during playback. Figure 7.4 shows the probability mass function (PMF) of bitrate changes in the dataset. 83% of video sessions in YouTube did not change bitrates during the entire playback.

7.3.0.5 Played bitrates

According to YouTube's encoding policies [142], YouTube streams eight different bitrates: `highres`, `hd1440`, `hd1080`, `hd720`, `large`, `medium`, `small` and `tiny`. We describe each bitrate setting in Table 7.2, and measured the distribution of played bitrates in Table 7.3. These measurements indicate that most viewers on desktops or laptops watched YouTube videos with `large` (33.1%) or `medium` (23.7%) bitrates. We also observed a few `hd1440` and

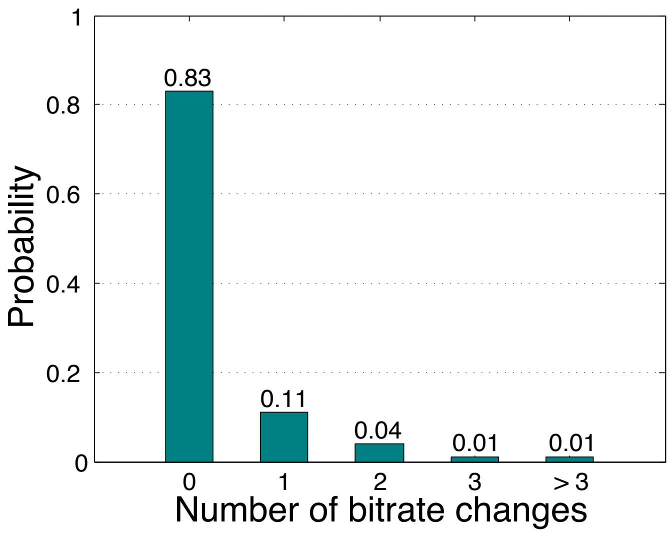


Figure 7.4: Probability of number of bitrate changes

Table 7.2: YouTube bitrate setting

Type	Video bitrate	Resolution
`highres`	35 - 45 Mb/s	3840 × 2160
`hd1440`	10 Mb/s	2560 × 1440
`hd1080`	8,000 kb/s	1920 × 1080
`hd720`	5,000 kb/s	1280 × 720
`large`	2,500 kb/s	854 × 480
`medium`	1,000 kb/s	640 × 360
`small`	400 kb/s	426 × 240
`tiny`	80 kb/s	256 × 144

Table 7.3: YouTube played bitrates (%)

`hd1080`	`hd720`	`large`	`medium`	`small`	`tiny`
6.8%	18.2%	33.1%	23.7%	13.3%	4.9%

`highres` videos, but the probability is much smaller (< 0.1%).

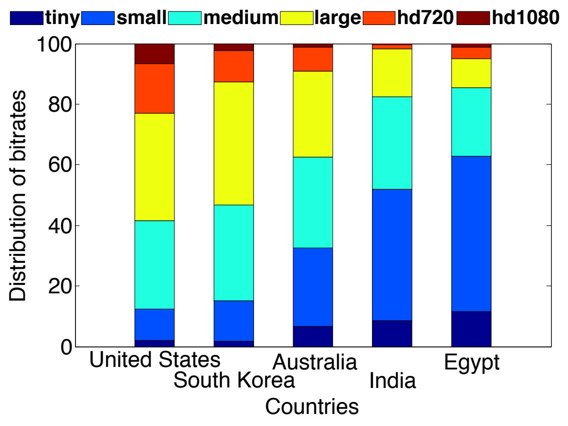
In Figure 7.5a, we compare the distributions of played bitrates across countries. For example, viewers in the United States and South Korea experienced higher bitrates in comparison to the ones in India and Egypt. Figure 7.5b shows the distributions of played bitrates for different ISPs in United States. For more details, we compare the distributions depending on different types of Internet connections such as fiber to the home (FTTH), hybrid fiber-coaxial (HFC) and digital subscriber line (DSL). We collected 7,074 samples in total for FTTH from Verizon's *FiOS* Internet service, and 6,618 samples for HFC from Time Warner Cable, Charter Communications, Cox Communications, Comcast and AT&T's *U-verse* (formerly *Project Lightspeed*). For DSL, we obtained 2,384 samples from Verizon (*non-FiOS*), AT&T (*non-U-verse*) and Qwest Communications. YouSlow can distinguish these by comparing the hostnames of the Internet service providers of the viewers using the IP geo-location database[3]. For example, Verizon hosts consistent domain names (e.g., `x.x.fios.verizon.net`) for their *FiOS* users. Through the measurements, we observe that the viewers using FTTH watched more videos at HD bitrates (36.8%) than the ones using HFC (25.3%) or DSL (14.4%).

7.3.0.6 Rebufferings

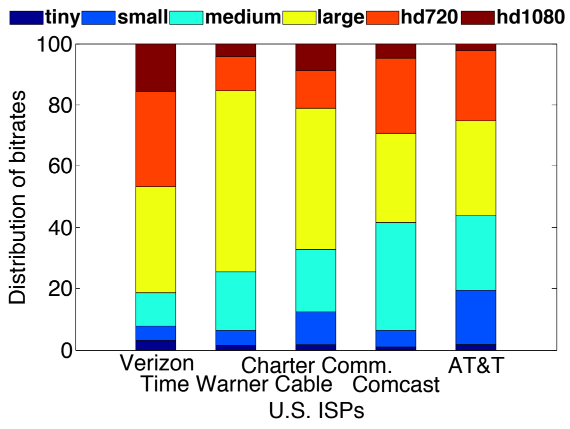
Figure 7.6 shows the PMF graph of total number of rebufferings. In the dataset, we find that more than 99% of video sessions suffered from fewer than six rebufferings and 67% of viewers experienced no rebufferings during the entire playback. Figure 7.7 shows the cumulative probability of total rebuffering duration per video session. Our experimental results show that only 10% of rebufferings exceeds 10 seconds in total.

7.3.0.7 AdBlock extension

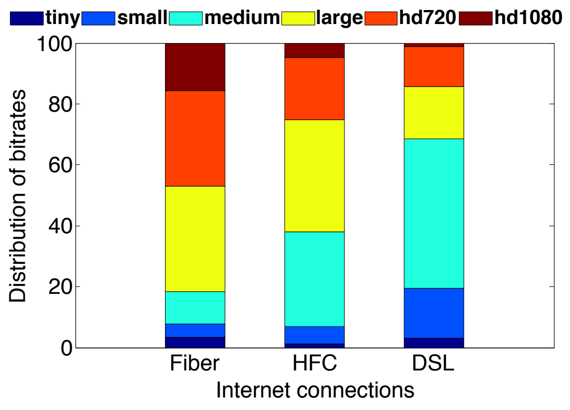
The YouSlow Chrome extension (version 1.2.8) is able to detect if a viewer uses an adBlock extension [140] on the Chrome web browser while a video is being played. Using the extension, a viewer may watch the YouTube videos without experiencing ads during entire playback. The extension investigates the URLs and HTML elements on the web page and blocks them if they are known as advertisement. We analyzed a total of 124,744 video sessions and found that 84,698 (67.9%) videos were watched on the Chrome web browsers using the adBlock extension.



(a) Countries



(b) U.S. ISPs



(c) Different types of Internet connections

Figure 7.5: Comparison of YouTube played bitrates

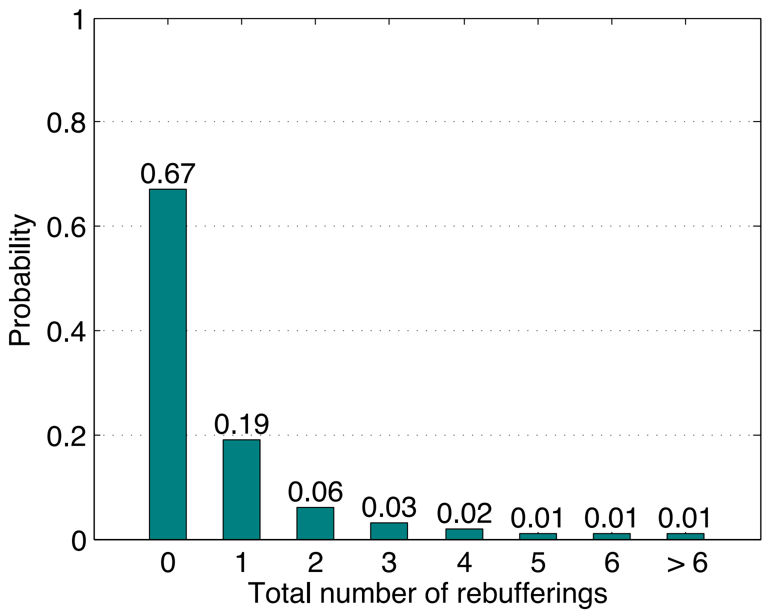


Figure 7.6: Probability of total number of rebufferings

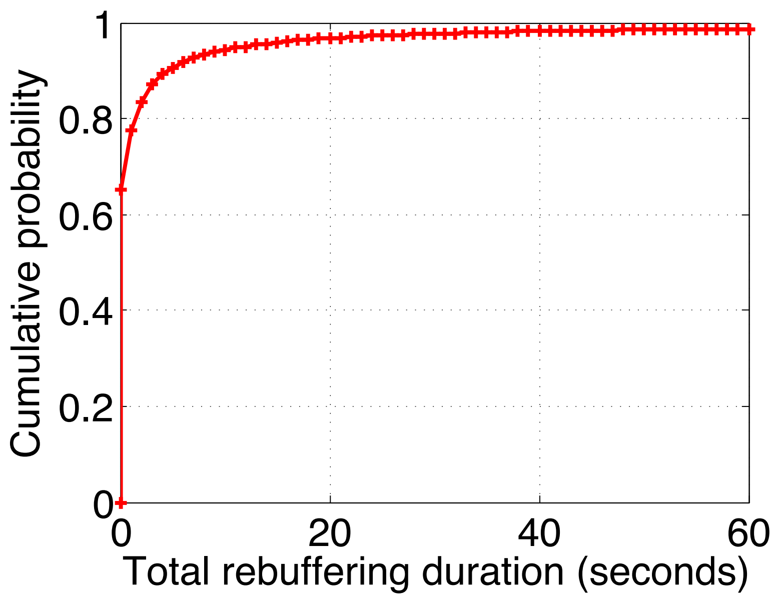


Figure 7.7: Cumulative probability of total rebuffering duration

7.3.0.8 Moving a scrollbar of YouTube player during playback

The YouSlow Chrome extension (version 1.2.8) is able to detect if a viewer is moving the scrollbar of YouTube player forwards or backwards during playback. We analyzed a total

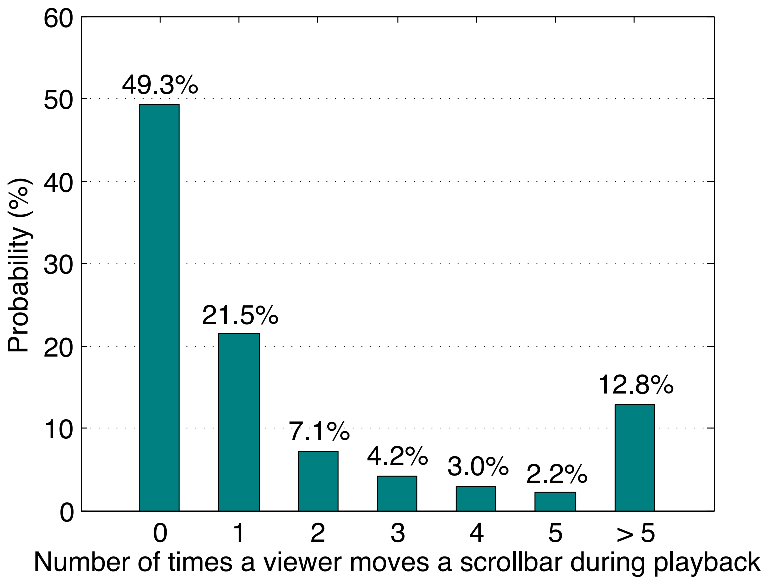


Figure 7.8: Probability of total number of times a viewer moves a scrollbar during playback

of 125,277 video sessions. Figure 7.8 shows our experimental results.

7.4 Video QoE analysis via YouSlow

In this section, we describe our analysis of video QoE based on YouSlow measurements. We are trying to answer the following questions:

- How do start-up delay, rebufferings and bitrate changes affect viewing interruption?
- What metrics can we use to analyze the impact of the above playback events on video QoE?

7.4.1 QoS and QoE methods for an analysis of video QoE

7.4.1.1 QoS methods

Several researchers [19; 20; 21] have used QoS-based metrics such as monitoring throughput, goodput, packet delay and jitter from intermediate nodes such as routers between viewers' devices and video servers, to analyze the performance of video streaming. This approach typically focuses on finding network impairments, but there are challenges to estimating

QoE for buffered video streaming. As an example, periods of low TCP throughput do not always interrupt a viewer's watching experience if an ABR player has downloaded enough data into the playout buffer. The QoS-based metrics are unable to detect the impact of low-throughput period since they cannot accurately track the playout buffer level from within the network routers.

Today's popular video streaming services such as YouTube and Netflix provide a video quality report to viewers [143; 144]. They simply measure the download speed from their own video content servers or video applications and rate the video streaming quality for different ISPs and geographical location of viewers. However, the output does not provide any QoE metrics to viewers, such as how often they experience bitrate changes and rebufferings.

7.4.1.2 QoE methods

In terms of the QoE definitions by ITU-T [145], it is the overall acceptability of an application or service, as perceived subjectively by the end-user. For video QoE, it is a perceptual measure that reflects viewers satisfaction with their video streaming experience. The common approach is a subjective method, hiring a group of people, having them watch short video clips, and scoring their viewing experiences under the laboratory or the crowd-sourcing environments [146; 147; 148]. However, such survey-based metrics are typically costly and time-consuming. In addition, it is difficult to automate and control the testing environments during the evaluation. To avoid high cost of subjective methods, objective methods are developed to estimate QoE of viewers [149; 150; 151]. This method focuses on building a statistical model based on various input QoS parameters measured from network or service layers such as packet loss, throughput, bitrate and video frame loss. However, it is hard to develop such QoE prediction models. For example, any modification made to current models such as adding or removing input parameters may require new tests to create new models. For evaluation, it also requires the survey-based methods. Due to the above limitations, today's many video researchers focus on the data-driven analysis [152; 153; 154; 155]. The common approach is from quality of "*experience*" such as happiness or satisfaction to quality of "*engagement*" such as video abandonment and failure [153]. The user-engagement metrics can be easily quantified and measured since it does not require direct user-involvement such as the mean opinion score (MOS) metrics that most subjective methods use.

7.4.1.3 Our analysis methodology and metrics

YouSlow is able to detect the video abandonment event directly from within the video player. Since the video rendering quality and level of interest are independent, we believe that our results are relatively insensitive to changes in how we define QoE-driven abandonment. We compute QoE abandonment ratio by dividing the number of sessions abandoned due to viewing interruptions such as rebufferings and bitrate changes by the total number of video sessions. For instance, we calculate an abandonment rate depending on how long viewers suffer from pre-roll ads and rebufferings and analyze the impact of bitrate changes on abandonment by comparing with constant bitrate videos. With a large number of samples, we believe that monitoring abandonment rates gives us practical and reliable outputs to analyze viewing interruptions in online video streaming.

7.4.2 QoE analysis report

Below, we summarize our QoE analysis based on YouSlow measurements in YouTube. The video samples are grouped and analyzed using the following notations:

- **Unimpaired videos:** The viewers watched the videos without any viewing interruptions such as video ads, long initial buffering (> 1 second), rebufferings and bitrate changes.
- **Ad-free videos:** The viewers watched the videos without experiencing pre-roll ads before the main content.
- **Rebuffered videos:** The viewers suffered from rebufferings int the middle of playback.
- **Initial buffered videos:** The viewers experienced long initial buffering (> 1 second) at the beginning of playback.

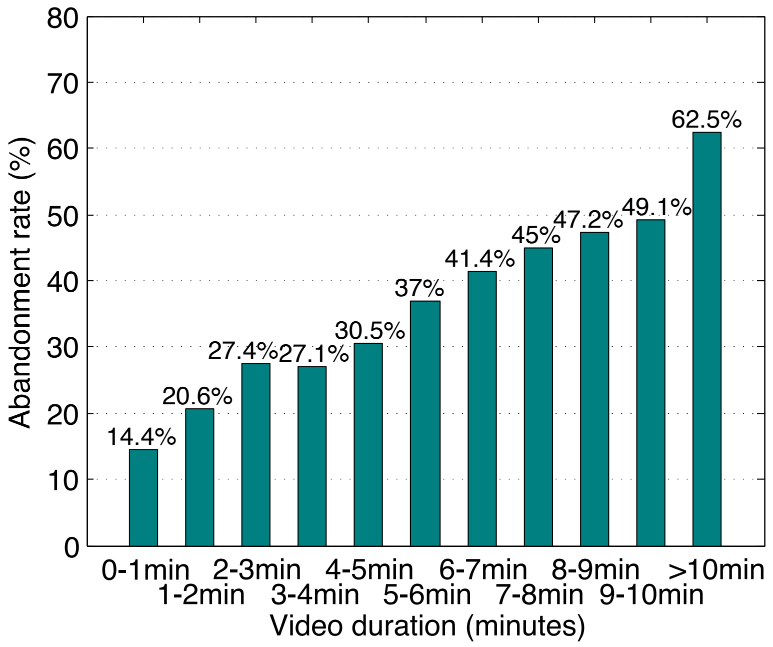


Figure 7.9: Abandonment rate (%) for unimpaired videos as video length increases

7.4.2.1 Video length

As a baseline analysis, we analyzed the abandonment rate depending on the video length. Concentrating on the impact of video length, we analyzed the unimpaired video sessions only. Figure 7.9 shows our experimental results. We find that an abandonment rate rises as video length increases. Most viewers decide whether or not they want to watch the video at the beginning of playback [156]. Figure 7.10 shows how many viewers stayed in the video sessions as the playback ratio increased for different video lengths. The playback ratio shows the ratio of content that has played in a video. For the videos that are longer than 10 minutes, for instance, we find that only 60% of viewers stayed in the video sessions when the playback ratio is 0.2. In addition, we analyzed the number of viewers as the playback duration (in seconds) increased for the videos that are longer than 5 minutes. As shown in Figure 7.11, about 44% of viewers abandoned the videos during the initial 60 seconds and then they started to abandon at slower rates.

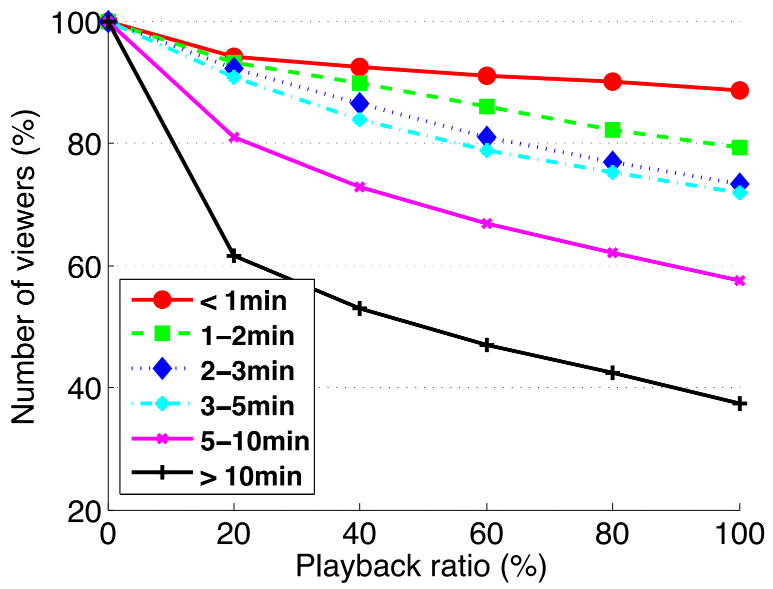


Figure 7.10: Number of viewers (%) for unimpaired videos as playback ratio increases

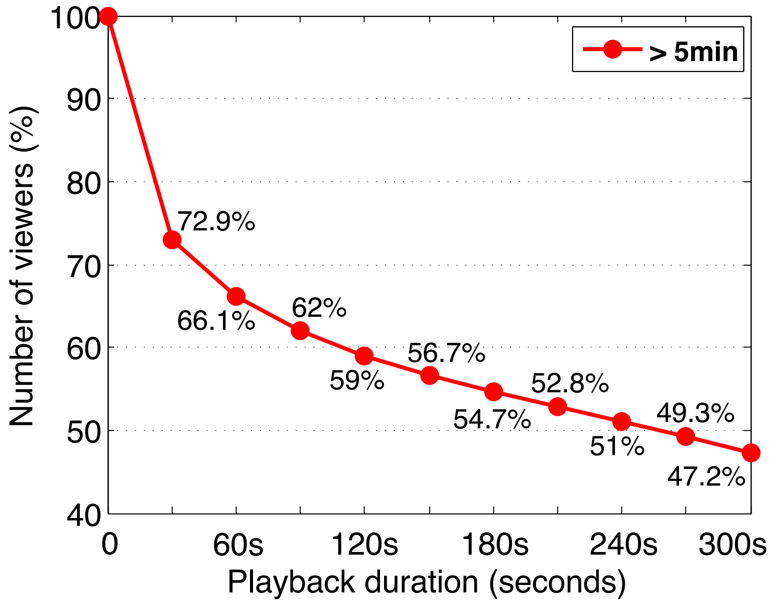


Figure 7.11: Number of viewers (%) for unimpaired videos (> 5 minutes) as playback length increases

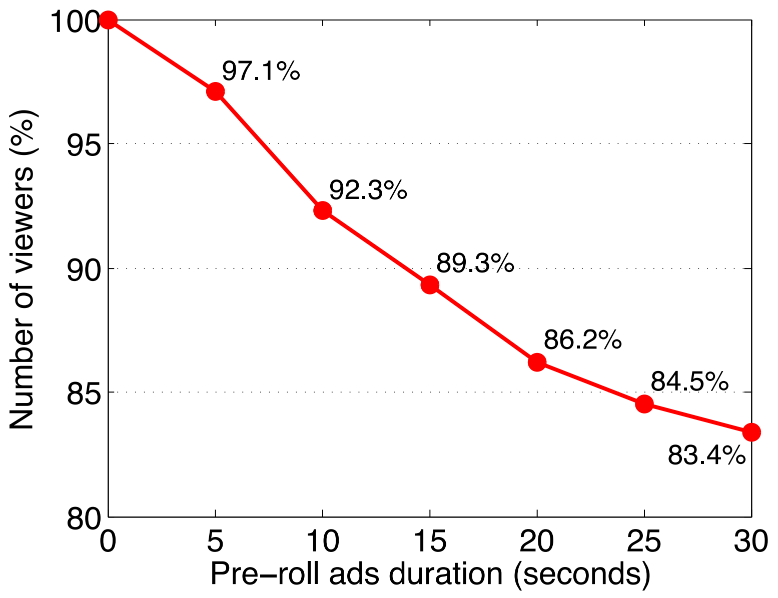


Figure 7.12: Number of viewers (%) during pre-roll ads as ad duration increases

7.4.2.2 Pre-roll ads

We have recently added a pre-roll ad analysis function to our Chrome extension (version 1.2.7). We analyzed a total of 11,038 video sessions where the viewers experienced the pre-roll ads. Among them, 2,635 videos were abandoned during the ads. Based on this analysis, the abandonment ratio for pre-roll ads is about 23.9%. Figure 7.12 shows how many the viewers stayed during the pre-roll ads as the ad length increased. We conjecture that most of the abandonment in Figure 7.12 took place during the *non-skippable* ads. Otherwise, the viewer may skip the ads instead of abandoning the videos. *Non-skippable* ads are usually 15-20 seconds long. Our experimental results show that about 10% of viewers abandoned the videos when the ads lasted for 15 seconds in YouTube.

7.4.2.3 Rebuffering

Most recent studies on video QoE [19; 126; 124; 157; 125] agree that rebufferings should be avoided if at all possible in order to enhance video QoE. In addition, they show that QoE of viewers can vary depending on the rebuffering pattern, i.e., how many or how often rebufferings appear during playback. We try to understand how viewers react to such different

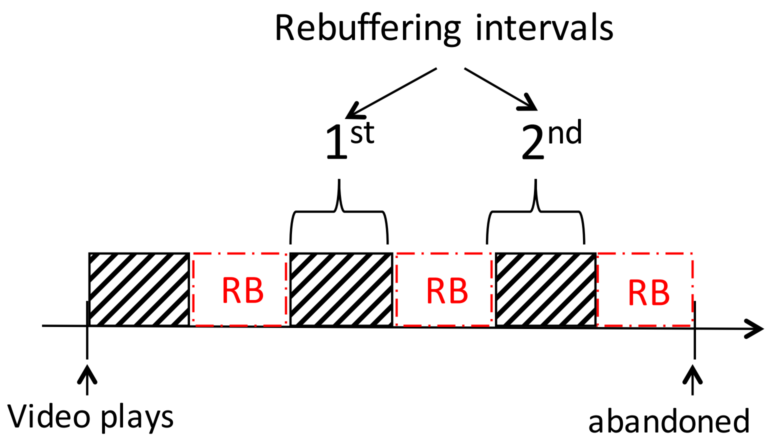


Figure 7.13: Two rebuffering (RB) intervals with three rebufferings

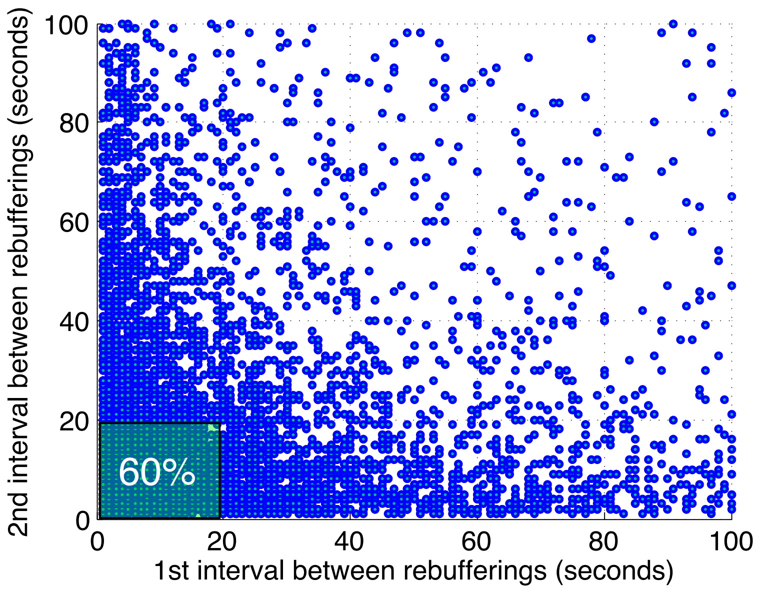


Figure 7.14: Plotting abandonments for videos with three rebufferings

rebuffering patterns in YouTube, along with abandonment rates. As a baseline analysis, we extract the video sessions from the dataset where the total number of rebufferings is three (Figure 7.13), and plot the abandonments based on the rebuffering intervals (Figure 7.14).

In Figure 7.14, we observe 60% of abandonments when the rebufferings intervals are less than 20 seconds. We frequently observe such short rebuffering intervals when an ABR player requests a higher bitrate than what a network can handle. In this case, the video play has to be paused until the player stores at least one segment in the buffer, which can cause a series of short-term rebufferings. Furthermore, we observe that an abandonment pattern varies depending on rebuffering intervals. For instance, let's suppose that we have a certain range of first rebuffering interval between 0 and 20 seconds in Figure 7.14. Depending on the second interval, we clearly see that the distribution of abandonments varies. The question is, how do we normalize the impact of rebuffering intervals and correlate the results with QoE assessments such as MOS? If we take into account a higher number of rebufferings or additional factors such as rebuffering duration and total playback length, QoE modeling will be much more complicated. To avoid such complexity, we consider a simpler metric below and analyze how will the metric predict the abandonment rate.

Rebuffering ratio: The total rebuffering duration is not sufficient for modeling QoE metrics since it does not take into account the total duration of playback. For example, viewers may experience watching interruptions differently depending on total playback duration, even if the video session has the same duration of total rebuffering. As an example, we calculate the abandonment rates for the video sessions where the total rebuffering duration is between 10 and 15 seconds but they have different total playback durations (20 through 100 seconds). Figure 7.15 shows our experimental results. We clearly see that the abandonment rate decreases when the playback duration increases.

$$\text{RB ratio} = \frac{\text{Rebuffering duration (second)}}{\text{Total playback duration (second)}} \tag{7.1}$$

To reflect this, we analyze the impact of rebufferings on abandonment rates using the rebuffering (RB) ratio in Equation 7.1. The ratio is defined as the fraction of time when a viewer experiences rebufferings while watching a video. As an example, rebufferings occur for ten seconds while a viewer stays in the video session for 100 seconds. In this case, the rebuffering ratio will be $10/100 = 0.1$. Depending on the rebuffering ratio, we calculate the abandonment rate. To avoid the video abandonment due to the lack of interest at the beginning of playback, we analyze the video sessions where the viewers watched the videos for at least 60 seconds. Figure 7.16 shows average abandonment rate as rebuffering ratio increases. We plot the results with the error bars based on the standard error of the mean (SEM), but the error values are less than 0.2% for each rebuffering ratio. We subtracted

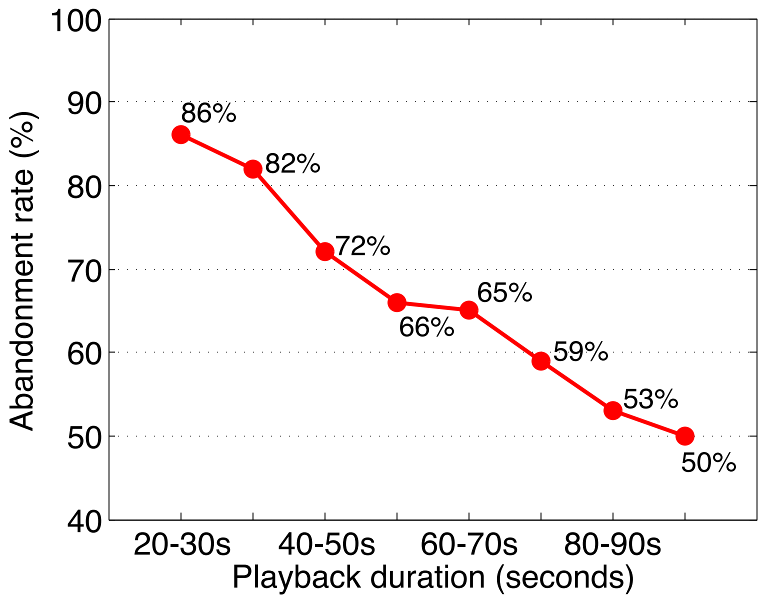


Figure 7.15: Abandonment rates (%) for ad-free, non-initial buffered and rebuffered videos - the same rebuffering duration but with different playback durations

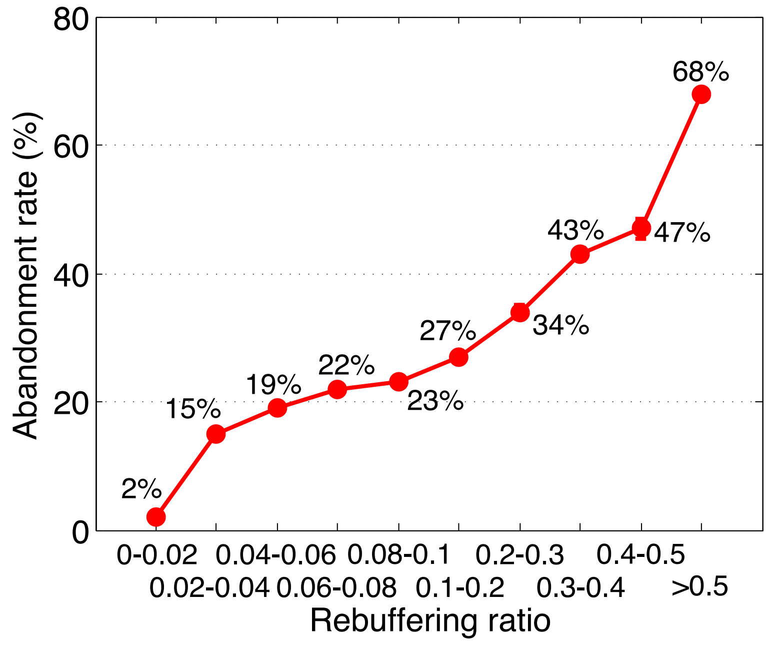


Figure 7.16: The impact of rebuffering on abandonment rates (%) for ad-free, non-initial buffered and rebuffered videos

Table 7.4: Number of samples for rebuffering ratio analysis

Rebuffering ratio	Number of samples
0 - 0.02	143,799
0.02 - 0.04	19,584
0.04 - 0.06	8,998
0.06 - 0.08	5,210
0.08 - 0.1	3,403
0.1 - 0.2	6,776
0.2 - 0.3	2,521
0.3 - 0.4	1,462
0.4 - 0.5	984
> 0.5	6,649

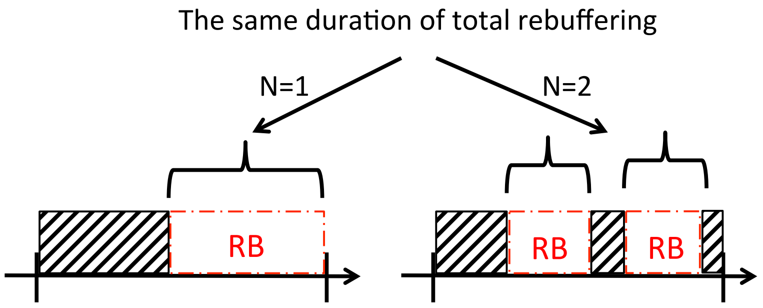


Figure 7.17: The same total rebuffering (RB) duration with different number of rebufferings

the unimpaired rate (RB ratio = 0) from the impaired rates. The abscissa indicates a range of rebuffering ratio ($x - y$ represents $x <$ ratio $\leq y$). Table 7.4 shows the number of samples for each rebuffering ratio. The results tell us that more viewers abandoned the videos as the rebuffering ratio increased.

We note that the rebuffering ratio does not take the number of rebufferings into account. As shown in Figure 7.17, for instance, it is possible that the number of rebufferings can vary although the total rebuffering duration is the same. This can affect video QoE differently. To prove it, we compare the impact of a single rebuffering event and multiple rebufferings by comparing the abandonment rates along with rebuffering ratio. Figure 7.18 shows our

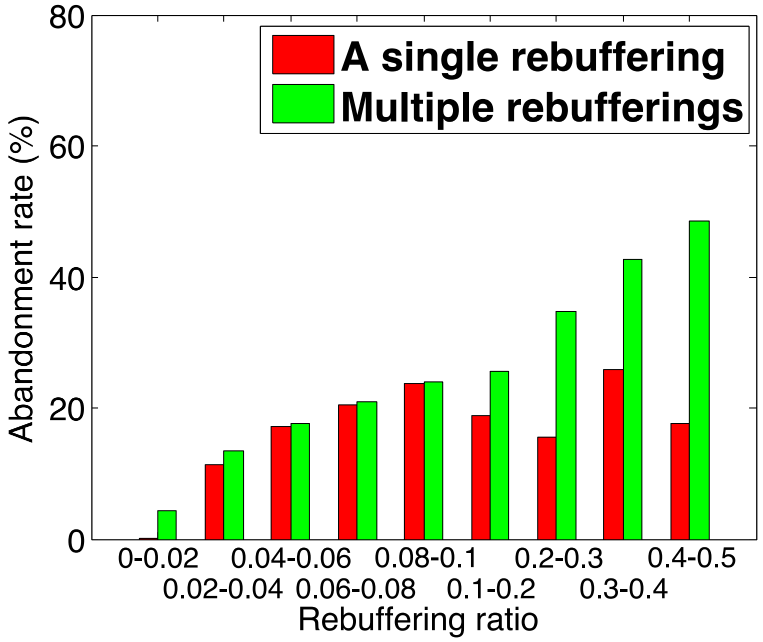


Figure 7.18: Comparison of abandonment rates between a single rebuffering event and multiple rebufferings for ad-free, non-initial buffered and rebuffered videos

experimental results. We clearly see that multiple rebufferings cause higher abandonment rates than a single rebuffering event.

Rebuffering early vs. later: We analyzed the impact of rebuffering start time on video abandonment. We collected the number of video sessions with abandonment due to rebufferings or initial buffering and counted the number of abandonment depending on the rebuffering start times. Figure 7.19 shows our experimental results. During the experiment, we analyze the video sessions with a single rebuffering event to avoid the impact of multiple rebufferings. We observe that the viewers were more likely to close the videos when they experience the initial buffering at the beginning of a playback. The initial buffering is related to the ABR heuristics of selecting bitrates and the network conditions when the video starts. For example, an ABR player may request high bitrate segments when it starts downloading a video. If the high bandwidth is available, it can play such high quality of video instantly, increasing the video QoE. However, it may experience a long initial buffering at the beginning of a playback if the network is congested.

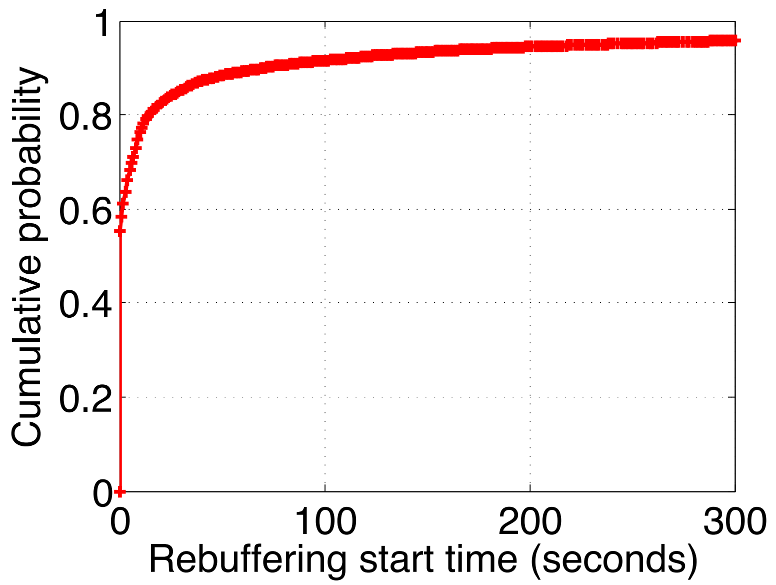


Figure 7.19: Cumulative probability of number of abandonments for ad-free, initial buffered and rebuffered videos with different rebuffering start times

7.4.2.4 Bitrate switching

Some papers [158; 128; 159; 127; 160] investigate the impact of bitrate changes on video QoE. They claim that providing a bitrate as high as possible does not necessarily lead to the highest QoE [128]. They agree that it is difficult to create a metric that takes into account of all the bitrate switching events, such as the number of bitrate changes, their amplitude (i.e., by how much bitrate increases or decreases) and the duration of each bitrate. Below, we try to find a simple metric that can properly reflect and quantify the impact of bitrate changes on abandonment rates in YouTube.

Bitrate change ratio: To find the impact of bitrate changes on abandonment rates, we use the following equation:

$$\text{BR change ratio} = \frac{\sum_{\text{i=1}}^{\text{Num. of BR changes}} |\log(\text{BR}_\text{i}/\text{BR}_{\text{i}-1})|}{\text{Num. of BR changes}} \tag{7.2}$$

BR_i and BR_{i-1} denote the newly selected bitrate and the previous bitrate (in kb/s), respectively. Using the above equation, we calculate the abandonment rates. To remove

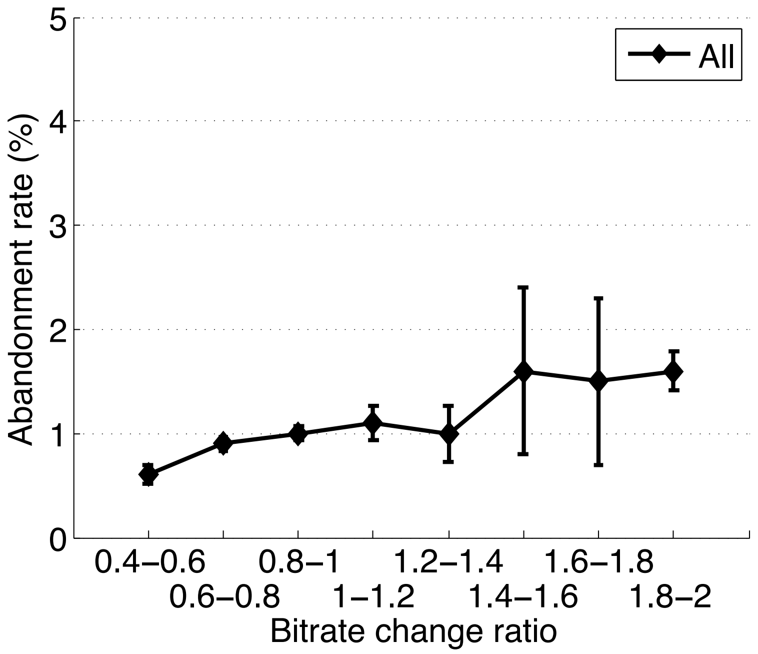


Figure 7.20: The impact of bitrate change on abandonment rates (%) for ad-free, non-initial buffered and non-rebuffered videos

the influence of other factors such as rebufferings and ads, we first collect the video sessions with bitrate changes only. To avoid counting the cases where a video is closed due to lack of interest, we only considered the videos as abandoned when they were watched at least 60 seconds and closed within five seconds after the bitrate was changed in the middle of a playback. Figure 7.20 shows our experimental results. We plot the results with the error bars based on the standard error of the mean (SEM). Table 7.5 shows the number of samples for each bitrate change ratio. The analysis indicates that the viewers were more likely to close the videos when the bitrate change ratio increased (Figure 7.20). However, the abandonment rate is not significantly different ($< 2\%$).

Table 7.5: Number of samples for bitrate change ratio analysis

	0.4 - 0.6	0.6 - 0.8	0.8 - 1	1 - 1.2	1.2 - 1.4	1.4 - 1.6	1.6 - 1.8	1.8 - 2
All bitrate changes	6,684	16,200	20,780	3,359	1,478	198	7,009	3,690
Positive bitrate changes	3,883	9,596	10,561	1,522	515	14	3,484	1,918
Negative bitrate changes	1,425	2,862	4,546	963	45	6	1,284	1,440

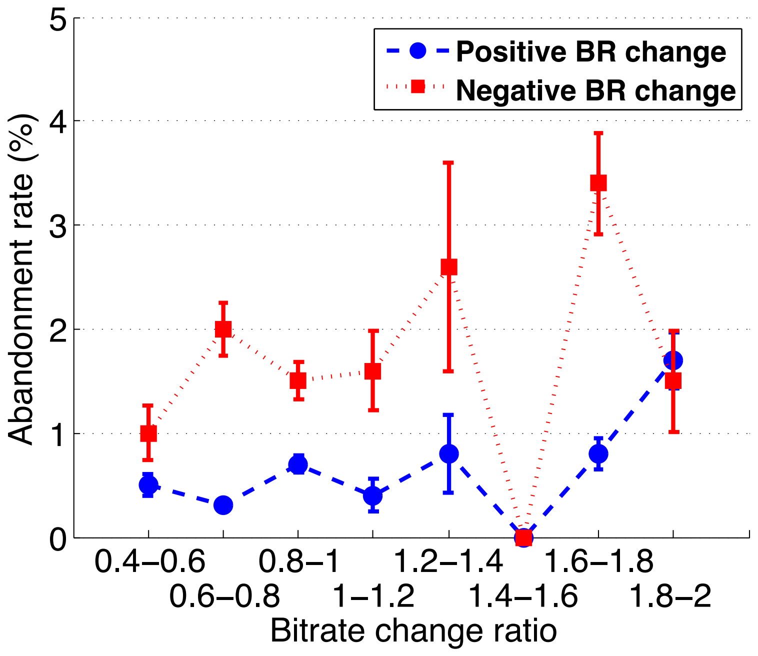


Figure 7.21: The impact of positive and negative bitrate changes on abandonment rates (%) for ad-free, non-initial buffered and non-rebuffered videos

The above result leads to the following question: does switching to a higher bitrate during playback also increase abandonment rate? To figure this out, we analyzed the video sessions with positive or negative bitrate changes separately. In Figure 7.21, positive bitrate changes present the views where there was only bitrate increase during video playback (e.g., BR_i - $BR_{i-1} > 0$) and negative bitrate changes present the views where there was only bitrate decrease during playback (e.g., BR_i - $BR_{i-1} < 0$). We clearly observe that decreasing bitrate causes higher abandonment rate than increasing bitrate during playback. Note that we have collected a small number of samples for the bitrate change ratio between 1.4 and 1.6 (Table 7.5) and all the viewers for the range completely watched the videos until the end.

The impact of bitrate changes on video loaded fraction: We compare the impact of positive and negative bitrate changes on the video loaded fraction. To remove the impact of multiple bitrate changes, we analyzed the video sessions that experienced a single bitrate change only. We split the dataset into three groups depending on different starting bitrates

Table 7.6: The impact of a single bitrate (BR) change on video loaded fraction for ad-free, non-initial buffered and non-rebuffered videos with different starting bitrates and total playback lengths

`tiny` or `small`	0 - 60 s	60 - 120 s	120 - 180 s	$>$ 180 s
No BR changes	0.41	0.78	0.9	0.94
Positive BR change	0.38	0.69	0.85	0.9
Negative BR change	0.24	0.63	0.83	0.89
`medium` or `large`	0 - 60 s	60 - 120 s	120 - 180 s	$>$ 180 s
No BR changes	0.44	0.83	0.92	0.95
Positive BR change	0.43	0.77	0.87	0.92
Negative BR change	0.33	0.7	0.83	0.9
`hd` or `highres`	0 - 60 s	60 - 120 s	120 - 180 s	$>$ 180 s
No BR changes	0.48	0.8	0.91	0.94
Positive BR change	0.43	0.74	0.86	0.92
Negative BR change	0.4	0.7	0.83	0.89

such as `tiny` / `small`, `medium` / `large` and `hd` / `highres`. Table 7.6 shows the average of video loaded fraction based on the total playback length. We observe low video loaded fraction for short playback duration (0 - 60 seconds). This typically happens when the videos are not what the viewers expected in YouTube. In this cases, the videos are easily abandoned at the beginning of playback, which results in the low loaded fraction. The viewers typically stayed longer in the video sessions when they watched high (`medium` or above) bitrates at the beginning of a playback. In addition, the viewers were likely to abandon videos early when the bitrates decreased, but the video loaded fraction is not significantly different. Interestingly, we also observe that more viewers abandoned the videos early even when the players increased the bitrates regardless of starting bitrate and playback length.

We analyze the impact of bitrate changes depending on the average played bitrate. For instance, let's suppose that two viewers watch the same video. The first one experiences frequent bitrate changes between 1 Mb/s and 3 Mb/s during playback and the average played

Table 7.7: The impact of bitrate (BR) change on video loaded fraction for ad-free, non-initial buffered and non-rebuffered videos with constant bitrate or multiple bitrate changes

Avg. played bitrate (kb/s)	Constant BR	Multiple BR changes
350 - 450 kb/s	0.9	0.81
900 - 1100 kb/s	0.92	0.83
2400 - 2600 kb/s	0.93	0.82

bitrate is 2 Mb/s. On the other hand, the second viewer watch the video with 2 Mb/s without any bitrate changes. The average played bitrate is the same. How does this difference affect the viewing experience? Based on the experimental results in Table 7.7, we confirm that viewers prefer high starting bitrates with no bitrate changes.

7.4.3 Regression analysis

Throughout the previous experimental results, we find that viewers experience interruptions differently depending on total playback time, rebuffering duration, number of rebufferings and bitrate changes during playback. Based on our YouSlow dataset, we conduct multiple linear regression analysis to investigate the relationship between the abandonment rate and the two viewing interruptions, rebufferings and bitrate changes. To concentrate on the impact of rebufferings and bitrate changes, we omit the video sessions that experienced pre-roll ads and long initial buffering (>1 second). To reduce the oscillation due to the viewers who abandoned videos during the beginning of playback, we analyze the sessions where the viewers watched the videos for at least 60 seconds (Figure 7.11). In the dataset, we found a small number of exceptional cases ($<1\%$) where the viewers watched the videos to the end even if they suffered from a very long initial buffering or rebufferings throughout the entire playback. We considered these samples as falsely reported and removed them from the dataset. Considering the outliers, Table 7.8 shows the statistics of video session for our regression analysis.

Table 7.8: Video playback statistics for regression analysis

Term	Min.	Max.
Playback length	60 s	1,000 s
Num. of rebufferings	0	6
Total rebuffering duration	0	25 s
Num. of bitrate changes	0	3

Table 7.9: Multiple linear regression analysis on abandonment rate using the number of rebufferings (RBs) and bitrate (BR) changes

Predictor variable	S	R-sq	R-sq (adj)
(1) only	0.047	59.6%	57.6%
(2) only	0.061	32.6%	29.2%
(1) and (2)	0.033	81.4%	79.4%

(1) Num. of RBs (2) Num. of BR changes

(a) Model summary

Source	DF	Adj SS	Adj MS	F-Value	P-Value
Regression	2	0.091	0.045	41.52	< 0.0001
Error	19	0.02	0.001		
Total	21	0.112			

(b) Analysis of variance

Term	Coef	SE Coef	T-value	P-value	VIF
Constant	0.1821	0.018	9.92	< 0.0001	
Num. of RBs	0.0246	0.003	7.06	< 0.0001	1.02
Num. of BR changes	0.0374	0.008	4.71	0.0002	1.02

(c) Coefficients

7.4.3.1 Using the number of rebufferings and bitrate changes

Can we find a strong linear relationship between the abandonment rate and the number of rebufferings and bitrate changes? We conduct the regression analysis between the abandonment rates and the two factors. Our experimental results are shown in Table 7.9. We note that the *Pearson* correlation coefficient of two factors is 0.141 (P-value = 0.531). This indicates that the two factors are not highly correlated. According to the ANOVA (analysis of variance) results in Table 7.9b, two factors predict the abandonment rate significantly, $F(2, 19) = 41.52$, $p < 0.0001$. The fitted regression model found from the analysis is $(\text{Abandonment rate}) = 0.1821 + 0.0246 * \text{Num. of RBs} + 0.0374 * \text{Num. of BR changes}$. The p-values in Table 7.9c tell us that the number of rebufferings has more impact on the abandonment rate than the number of bitrate changes. We can see the high value of R-squared, 0.814 of the explained variability in abandonment rate. In general, the higher the R-squared, the better the model fits the data.

7.4.3.2 Using the number of rebufferings and rebuffering / bitrate change ratios

We categorize the dataset into several groups depending on three predictors, the number of rebufferings and the rebuffering and bitrate change ratios using Equation 7.1 and 7.2. For the rebuffering ratio, we first counted the number of sessions with the same rebuffering ratio and tried to calculate the abandonment rate. However, the total playback time varies between 60 seconds and 1,000 seconds. So, it is difficult to gather a sufficiently large number of samples for each rebuffering ratio for the analysis of the abandonment rate. To address this, we split the dataset into multiple subsets depending on the normalized rebuffering and bitrate change ratios. We divided sessions into 0.1 intervals of rebuffering ratio. For instance, we consider the sessions with the same range of rebuffering ratio between 0.1 and 0.2 as the session that has 0.15 of rebuffering ratio. For the bitrate change ratio, we divided the sessions into 0.2 intervals. Table 7.10 shows our regression analysis. Figure 7.22 presents the fitted lines between the abandonment rate and each predictor. Among the three predictors, the rebuffering ratio has the most impact on the abandonment rate while the bitrate change ratio has the least impact (Table 7.10a). Using the rebuffering ratio and the number of the rebufferings, we can achieve the highest R-squared value ($R^2 = 0.94$).

Table 7.10: Multiple linear regression analysis on abandonment rate using the number of rebufferings (RBs) and rebuffering / bitrate (BR) change ratios

Predictor variable	S	R-sq	R-sq (adj)
(1) only	0.06	91.3%	90.9%
(3) only	0.122	64.3%	62.7%
(1) and (3)	0.049	94.6%	94.0%
(1) and (2)	0.0615	91.4%	90.5%
(1), (2) and (3)	0.05	94.6%	93.7%

(1) RB ratio (2) BR change ratio (3) Num. of RBs

(a) Model summary

Source	DF	Adj SS	Adj MS	F-Value	P-Value
Regression	3	0.872	0.29	115.92	< 0.0001
Error	20	0.05	0.002		
Total	23	0.922			

(b) Analysis of variance

Term	Coef	SE Coef	T-value	P-value	VIF
Constant	0.142	0.033	4.28	0.0004	
RB ratio	2.156	0.212	10.17	< 0.0001	2.13
BR change ratio	0.001	0.027	0.07	0.9483	1.32
Num. of RBs	0.031	0.009	3.43	0.0027	2.04

(c) Coefficients

7.4.4 Summary of key observations

These are the key findings from our QoE experimental results:

Finding 1: Our measurements show that about 10% of viewers abandoned the YouTube videos when the pre-roll ads lasted for 15 seconds (Figure 7.12).

Finding 2: We observe that viewers are more likely to abandon videos with multiple rebufferings compared to a single rebuffering event even if the rebuffering ratio is the same

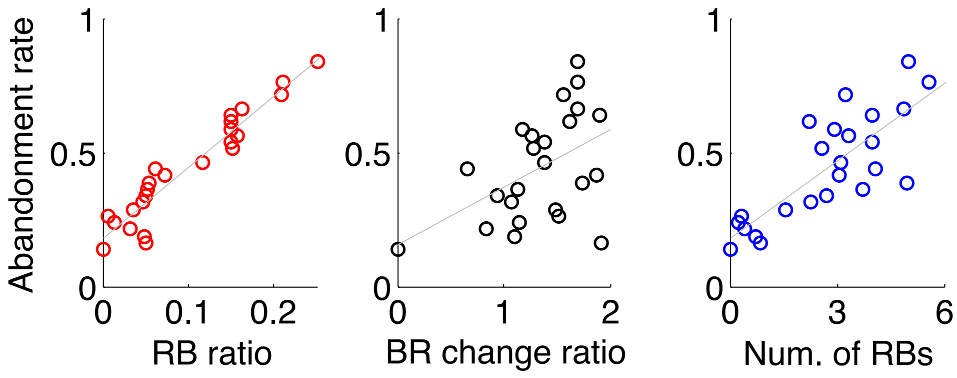


Figure 7.22: Fitted lines for multiple linear regression analysis

(Figure 7.18). We confirm that the initial buffering has more impact on the video abandonment than the rebufferings in the middle of a playback (Figure 7.19). In addition, we observe that viewers prefer constant bitrate to increasing bitrate during playback, but the abandonment rate is not significantly different (Figure 7.20 and 7.21).

Finding 3: We find that monitoring the number of rebufferings and the rebuffering ratio is a good metric to quantify video abandonment rates for short videos such as YouTube. Compared to the rebuffering impact, the bitrate change does not affect the video abandonment significantly. To estimate the abandonment rate in YouTube, we can create a strong linear regression model ($R^2 = 0.81$) using the number of rebufferings and bitrate changes only (Table 7.9). We can increase the R-squared value up to 0.94 with the combination of the rebuffering ratio and the number of rebufferings (Table 7.10).

7.5 YouSlow challenges

Below, we address YouSlow challenges:

Scalability: We currently operate a single database server that collects about 3,500 YouTube views every day. As the number of YouSlow users increases, we plan to deploy multiple servers on the cloud platform.

Video player APIs: YouSlow relies on the YouTube player API [135]. Thus, any modification made to current APIs can affect our measurements. To avoid this issue, we track its revision history and try to reflect every change to existing YouSlow platform.

Geolocation database update: YouSlow uses the MaxMind geolocation database that maps the public IP address to the ISP information such as hostname and location. The database needs to update periodically. Depending on ISPs, there is a possibility we may fail to find approximate ISP location. For example, when we watch YouTube videos in public places such as coffee shops or book stores, some local ISPs in New York such as AT&T and Time Warner Cable lead us to the wrong place (e.g., Butler, Kansas, US), instead of their real locations.

Development of iOS and Android mobile applications: We currently focus on the YouSlow Chrome extension. We released our mobile applications via YouSlow homepage. We plan to publish them in the Apple's App or Google Play store.

7.6 Future work

Adding other streaming services to YouSlow: As we noted, YouSlow platform can be easily implemented for other streaming services such as Netflix[4] and Hulu if they provide any player APIs. In this case, we can monitor and compare behaviors of video watching viewers, between short video clips such as music videos and sports highlights in YouTube and long videos such as movies and TV shows in Netflix [152]. In addition, YouSlow can be used to analyze video ads in social networks. For example, Facebook recently began allowing embedded videos to play automatically when users scroll to that page. Using YouSlow, we expect to investigate the impact of video ads on user behavior in Facebook and compare the results with YouTube.

Time and space consistency analysis for different ISPs: YouSlow analyzes video QoE depending on the location of viewers and the ISPs that the viewers are connected to. Based on the measurements, we can track the time and space consistency of various video playback events such as average played bitrate and rebuffering ratio in YouTube for different

[4]As of Nov. 14th, 2014, public API developers are no longer able to access Netflix content. Netflix is taking its API private [161].

ISPs. This approach is similar with other video QoE reports by Google [143], Netflix [144] and the Measuring Broadband America (MBA) program by the Federal Communications Commission (FCC)[5] [162]. We plan to develop the YouSlow APIs that allow end-users to retrieve and analyze the information over time and space.

Adding network troubleshooting functions to YouSlow: We plan to add various network testing and troubleshooting functions to the YouSlow Chrome extension. When rebufferings occur during playback, for example, YouSlow diagnoses common network connection issues by measuring ping and download speed and provide the measurements to the viewers via the extension.

Fixed vs. mobile devices: We can compare video abandonment when viewers watch videos on desktops / laptops or mobile devices. Due to lack of reports from mobile devices, we leave this for future work.

7.7 Related work

Video QoE analysis: Dobrian et al. [152] at Conviva monitored user-engagement based on various playback events measured from video players. The methodology used for data collection is similar to our approach. They focused on the analysis of initial buffering and rebuffering ratio, not bitrate switching. They found that the rebuffering ratio has the largest impact on video abandonment and the impact is quantitatively different depending on content types. They argued that initial buffering has significantly lower impact on video abandonment, diverging from our findings as shown in Figure 7.19. Unlike their approach, our platform allows viewers and video service providers to monitor various playback statistics in real time via our QoE monitoring system. In addition, we suggest simpler metrics (e.g., monitoring rebuffering ratio, number of rebufferings, bitrate change ratio over playback time) that can be implemented at video players to estimate abandonment rates. Shafiq et al. [154] monitored video abandonment by inspecting video packets from the ISPs' viewpoint, but the method is more complicated compared to our web browser plug-in that

[5]The Measuring Broadband America (MBA) program uses the FCC speed test application that measure fixed or mobile broadband performance such as upload and download speed, latency and packet loss.

can detect such abandonments directly from within video players. Hossefeld et al. [124] investigated the impact of rebuffering patterns (i.e., how many and often rebufferings appear during playback) on video QoE. They found that it is difficult to estimate video QoE by considering the total rebuffering duration only. Krishnan et al. [155] investigated the effectiveness of video ads by monitoring their completion and abandonment rates. They found that an ad for long-term videos such as TV shows and movies is about 4% more likely to complete than the same ad for the short-term videos such as YouTube's video clips. They also observed that the viewers abandoned more quickly in the beginning of the ad and abandoned at slower rates as the ad progressed.

Collecting measurement data from a web browser plug-in: For analyzing network performance issues such as page loading times, Dhawan et al. [163] introduce *Fathom*, a browser-based network measurement platform. As a proof of concept, they have built a Firefox plug-in that allows web sites or other parties to program network measurements using JavaScript. Barbara et al. [164] have built a YouTube monitoring tool (YoMo) that analyzes the amount of playtime buffered by the YouTube player. The Firefox plug-in focuses on the Flash-based streaming in YouTube and monitors TCP flows at the client in order to estimate the time when the YouTube player is stalling. They focused on the analysis of buffering status of YouTube player, but did not investigate QoE metrics in video streaming.

7.8 Conclusions

We introduced YouSlow as a new video QoE analysis tool for video QoE. Our experimental results show that monitoring the rebuffering ratio and counting the number of rebufferings during playback are proper QoE metrics to analyze abandonment rates for short videos such as YouTube. As key observations, we find that about 10% of viewers closed the videos during the pre-roll ads when the ads lasted for 15 seconds. Further, our analysis shows that viewers prefer constant bitrate to increasing bitrate during playback. Our regression analysis shows that the rebuffering ratio has the most significant impact on the abandonment rate compared to the bitrate change ratio and the number of rebufferings. We believe that our proposed QoE metrics and experimental results give us an insight to improving ABR heuristics in ABR players and enhancing viewing experiences.

Part V

Conclusions

This thesis investigated OTT video streaming services using ABR technologies. In ABR streaming, a video player running on a viewer's device adapts bitrates to match given network conditions. Even though such self-adjusting mechanism is designed to improve video QoE, there are still many challenges in providing a reliable streaming experience due to the lack of direct knowledge of access network channels, frequent user mobility and difficulty of monitoring the watching experiences of viewers. The primary contribution of this thesis lies in addressing those challenges.

First, this thesis introduces dynamic network condition-aware video server selection algorithms. In today's CDN-based video streaming, the video service providers typically assign video content servers that are geographically close to viewers. However, it is possible that the network conditions can be unstable even if the content server is located near the viewer. To resolve this issue, we suggest to discover a better video content server on wireless edge nodes (e.g., RNC in a 3G network and P-GW in a 4G network) based on measured RTTs when a viewer requests a video. Through an empirical analysis of YouTube, we prove that our solutions outperform the existing location-based algorithms by providing higher TCP performance.

Second, this thesis presents QoS-aware video streaming in wireless networks. In our proposed platform, wireless edge node supports an ABR player in selecting a proper bitrate by dynamically controlling the maximum allowable TCP throughput on the video streaming flow based on the changing network conditions over the air interface. By monitoring TCP/IP and HTTP packets in real time, it also discards unnecessary video packets in advance before they are being delivered to the viewer.

Third, this thesis presents QoE-aware video streaming using SDN. Our proposed SDN application monitors the status of streaming flows (e.g., downloading throughput and video QoE factors from viewer's device) in real time and dynamically changes routing paths in WANs using MPLS-TE. Instead of using simulation or emulation tools, we experimented with an off-the-shelf SDN platform (Juniper network's Junos Space [26]) to show the feasibility of our approach.

Fourth, this thesis investigates the impact of playout buffer size in ABR streaming on video QoE. To measure the viewer's watching experience, we conducted subjective video experiments, collecting data from more than 200 participants using an online crowdsourcing platform. Our survey results show that a small buffer can achieve higher QoE by yielding more high bitrate intervals than a large buffer especially under fast varying network conditions. Based on these findings, we suggest that an ABR player should change its maximum playout buffer size depending on the remaining buffer occupancy during playback. In our testbed, we demonstrated that an ABR player dynamically switching between small and large buffers outperforms the player with fixed buffers by providing fewer rebufferings and higher played bitrates during a download.

Finally, this thesis introduces YouSlow, a video QoE monitoring tool that can evaluate existing OTT video streaming services. The key idea is to monitor video abandonments and quantify the results for playback events such as starting bitrate, start-up latency, rebufferings and bitrate changes. The YouSlow tool has collected more than 1,400,000 YouTube views from more than 110 countries. We measured the impact of these playback events on video abandonment and found that about 10% of viewers abandoned the YouTube videos when the pre-roll ads lasted for 15 seconds. Even increasing the bitrate can annoy viewers; they prefer a high starting bitrate with no bitrate changes during playback. Our regression analysis shows that the rebuffering ratio has the most significant impact on the abandonment rate and the abandonment rate can be estimated accurately using the rebuffering ratio and the number of rebufferings ($R^2 = 0.94$).

This thesis proposes various solutions that can support network operators and video service providers to resolve the inefficiency of today's OTT video streaming. The solutions have been experimentally validated on real networking equipment such as Wi-Fi access points and SDN controllers. However, there is a difficulty in implementing and testing the ideas on wireless edge nodes in real mobile networks (e.g., 3G and 4G), which we leave for our future work. Despite the limited resource for evaluation, we believe that our proposed solutions and experimental results will be a stepping stone to how we improve today's OTT streaming in future networks, such as 5G networks.

Part VI

Glossary

720p, 1080p and 1440p. High-definition (HD) videos that have 720, 1080 and 1440 lines of vertical resolution, respectively. The *p* refers to progressive scan HD video.

ABR streaming. It represents adaptive bitrate streaming. The video streaming technology is described in Part I.

A-AMBR. APN Aggregate Maximum Bit Rate (A-AMBR) is the maximum allowed total non-GBR throughput to specific APN.

APN. An Access Point Name (APN) is the name of a gateway between a GSM, GPRS, 3G or 4G mobile network and another computer network, frequently the public Internet.

Buffer. In streaming applications, buffers store video or audio data until there is enough information for the stream to be composed.

Bitrate. The number of bits per second (bps) at which a video stream is delivered. For ABR streaming the bitrate will change based on a request from the video player on an end-user's device.

DASH. Dynamic Adaptive Streaming via HTTP, one of ABR streaming technologies developed by the Motion Picture Experts Group (MPEG).

DD-WRT. It is a Linux based alternative OpenSource firmware suitable for a variety of WLAN routers and embedded systems.

Encoding. Converting content from one form of video signal to another, either in real-time for live streaming or in non-real-time for further manipulation.

EPC. The Evolved Packet Core (EPC) defined in 3GPP Rel-8 [78].

EPS. 3GPP term referring to a complete end-to-end system, that is UE, E-UTRAN and EPC.

E-UTRAN. The radio access network that implements LTE radio interface technology.

EPS bearer. A bearer is a virtual pipe line connecting two or more points in the communication system in which data traffic follow through. An Evolved Packet System (EPS) bearer is a pipe line between an UE and a P-GW in an LTE network.

Frame rate. The number of frames per second (FPS) for video. Generally, $24\,p$ is used for transferring a video signal to film and $50/60\,p$ is used for High-definition television (HDTV).

Fast start. In video streaming, fast start is represents a technique that tries to fill-up the playout buffer of a streaming client at the beginning of a presentation and starts the presentation faster, hence improving user experience.

GBR. The minimum guaranteed bit rate per EPS bearer in an LTE network.

Goodput. The application level throughput. It counts the number of useful bits delivered through the network for a certain time period to determine the network efficiency.

HDS. It is Adobe's method for ABR streaming.

High definition. Often referred to as HD, resolutions of 1024×720 or 1920×1080.

HLS. HLS stands for HTTP Live Streaming, one of ABR streaming technologies developed by Apple.

I frame. Intra picture frame, or I-frame, compression is a shorthand way of referring to intraframe compression. It can also be used as the basis for interframe compression, if predictive (P frames) can reference a key frame (which is always an I frame) and then predict movement across multiple P frames.

Last mile technology. The technology that carries signals from the broad telecommunication backbone to and from the home or business.

MBR. The maximum guaranteed bit rate per EPS bearer in an LTE network.

Meta-data. Meta-data carries what a file contains such as file location, bitrates, time and date.

MOS. Mean opinion score is a metric that has been used for decades in telephony networks to obtain the human user's view of the quality of the network, typically on a scale of 1 to 5.

OTT. Over-the-top (OTT) is the delivery of film and TV content via the Internet, without requiring users to subscribe to traditional cable or satellite pay-TV services.

P-GW. A Packet Data Network Gateway (P-GW) is a gateway between the LTE network and other packet data networks, such as the Internet or SIP-based IMS networks.

Progressive download. A method for streaming non-live video to the user for immediate playback. Supported in the user's media player, progressive download employs HTTP, the protocol used to download everything from the web.

Player. A handheld device or an application running on PCs or mobile devices, that plays videos.

QCI. QoS Class Identifier (QCI) is a mechanism used in LTE networks to ensure bearer traffic is allocated appropriate QoS. Nine different QCI values are standardized to reference specific QoS characteristics regarding resource type (GBR or non-GBR), priority (1 9), packet delay budget (from 50 ms to 300 ms), packet error loss rate (from 10^{-2} to 10^{-6}).

QoE. Quality of Experience is a measure of the overall level of customer satisfaction with vendors, services or products. QoE expresses user-satisfaction both objectively and subjectively.

QoS. As per ITU-T Recommendation E.800 [146], quality of service is totality of characteristics of a telecommunications service that bear on its ability to satisfy stated and implied needs of the user of the service.

Rebuffering. Rebuffering occurs in the middle of a playback when the available bandwidth falls short of the presentation's required bandwidth. The play will be paused until there is enough data for the stream to be stored in the buffer.

SDN. Software-defined networking is an approach to networking in which control is decoupled from the physical infrastructure, allowing network administrators to manage network services through abstraction of lower-level functionality.

Smooth streaming. Microsoft's proprietary ABR streaming technology.

Standard definition. Often referred to as SD, resolutions of 720×576 or 720×480.

Transit network. Transit networks connect other networks, but do not serve end-users.

UE. In the UMTS and the LTE networks, user equipment (UE) is any device used directly by an end-user to communicate.

UE-AMBR. UE Aggregate Maximum Bit Rate (UE-AMBR) is the maximum allowed total non-GBR throughput among all APN to a specific UE.

UMTS. The Universal Mobile Telecommunications System (UMTS) is a third generation mobile cellular system.

Viewer. A person who watches videos, or a device that downloads and plays videos.

Part VII

Bibliography

Bibliography

[1] FCC Adopts 15th Report on the Status of Competition in the Market for the Delivery of Video Programming. Retrieved August 12, 2015 from `https://www.fcc.gov/document/fcc-adopts-15th-report-video-competition-0`.

[2] Maggie MacDonald. Comcast vs. Netflix: Why the FCC Should Redefine Multi-Channel Video Programming Distributors to Include over-the-Top Video Providers. *Colo. Tech. LJ*, 12:479, 2014.

[3] Nitin Narang, What is the Difference between OTT and IPTV? Retrieved August 12, 2015 from `http://www.mediaentertainmentinfo.com/2013/04/2-concept-series-what-is-the-difference-between-ott-and-iptv.html/`.

[4] Cisco, Visual Networking Index: Forecast and Methodology 20152020. Retrieved June 29, 2016 from `http://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/complete-white-paper-c11-481360.html`.

[5] Citrix, Bytemobile Mobile Analytics Report. Retrieved June 12, 2015 from `https://www.citrix.com/`.

[6] Adobe Digital Index, U.S. Digital Video Benchmark Q4 2015. Retrieved June 29, 2016 from `http://www.cmo.com/`.

[7] Ooyala, Global Video Index Q4 2015. Retrieved June 29, 2016 from `http://go.ooyala.com/wf-video-index-q4-2015.html`.

[8] Making Sense of Video Streaming Protocols. Retrieved June 21, 2016 from `https://www.linkedin.com/pulse/making-sense-video-streaming-protocols-dr-brijesh-kumar`.

[9] Henning Schulzrinne, Stephen L. Casner, Ron Frederick, and Van Jacobson. RTP: A Transport Protocol for Real-Time Applications. IETF Draft, July 2003.

[10] Henning Schulzrinne, Anup Rao, Rob Lanphier, Magnus Westerlund, and Martin Stiemerling. Real Time Streaming Protocol 2.0 (RTSP). IETF Draft, February 2014.

[11] Real-Time Messaging Protocol (RTMP) specification. Retrieved June 21, 2016 from `http://www.adobe.com/devnet/rtmp.html`.

[12] Microsoft Media Server Protocol. Retrieved June 21, 2016 from `https://msdn.microsoft.com/en-us/library/cc239490.aspx`.

[13] Raj Jain. Data Flies Standby with ABR Service. *Computing Research Repository (CoRR)*, cs.NI/9809100, 1998.

[14] Saamer Akhshabi, Ali C. Begen, and Constantine Dovrolis. An Experimental Evaluation of Rate-Adaptation Algorithms in Adaptive Streaming over HTTP. In *Proceedings of ACM MMSys*, San Jose, CA, USA, February 2011.

[15] Apple HTTP Live Streaming. Retrieved June 12, 2015 from `https://developer.apple.com/streaming/`.

[16] Microsoft IIS Smooth Streaming. Retrieved June 12, 2015 from `http://www.iis.net/downloads/microsoft/smooth-streaming`.

[17] Adobe HTTP Dynamic Streaming. Retrieved June 12, 2015 from `http://www.adobe.com/products/hds-dynamic-streaming.html`.

[18] Dynamic Adaptive Streaming over HTTP. Retrieved June 12, 2015 from `http://mpeg.chiariglione.org/standards/mpeg-dash`.

[19] Ricky K. P. Mok, Edmond W. W. Chan, and Kuang-Chiung Chang. Measuring the Quality of Experience of HTTP Video Streaming. In *Proceedings of IFIP/IEEE IM*, Dublin, Ireland, May 2011.

[20] Kuan-Ta Chen, Chi-Jui Chang, Chen-Chi Wu, Yu-Chun Chang, and Chin-Laung Lei. Quadrant of Euphoria: A Crowdsourcing Platform for QoE Assessment. *Journal of IEEE Network*, 24(2):28–35, March 2010.

[21] Pedro Casas, Raimund Schatz, and Tobias Hossfeld. Monitoring YouTube QoE: Is Your Mobile Network Delivering the Right Experience to your Customers? In *Proceedings of IEEE WCNC*, Sanghai, China, April 2013.

[22] Hyunwoo Nam, Bong Ho Kim, Doru Calin, and Henning Schulzrinne. A Mobile Video Traffic Analysis: Badly Designed Video Clients Can Waste Network Bandwidth. In *Proceedings of IEEE Globecom CTEMD Workshop*, Atlanta, USA, December 2013.

[23] Hyunwoo Nam, Kyung-Hwa Kim, Bong Ho Kim, Doru Calin, and Henning Schulzrinne. Towards Dynamic QoS-aware Over-The-Top Video Streaming. In *Proceedings of IEEE WoWMoM*, Sydney, Australia, June 2014.

[24] Hyunwoo Nam, Kyung Hwa Kim, Doru Calin, and Henning Schulzrinne. Towards Dynamic Network Condition-Aware Video Server Selection Algorithms over Wireless Networks. In *Proceedings of IEEE ISCC*, Madeira, Portugal, June 2014.

[25] Hyunwoo Nam, Kyung-Hwa Kim, Jong Yul Kim, and H. Schulzrinne. Towards QoE-aware Video Streaming Using SDN. In *Proceedings of IEEE Globecom*, December 2014.

[26] Juniper Network's Junos Space. Retrieved June 16, 2015 from `https://www.juniper.net/assets/us/en/local/pdf/datasheets/1000297-en.pdf`.

[27] Hyunwoo Nam, Kyung-Hwa Kim, Doru Calin, and Henning Schulzrinne. YouSlow: A Performance Analysis Tool for Adaptive Bitrate Video Streaming. In *Proceedings of ACM SIGCOMM*, Chicago, Illinois, USA, August 2014.

[28] Hyunwoo Nam, Kyung-Hwa Kim, and Henning Schulzrinne. QoE Matters More Than QoS: Why People Stop Watching Cat Videos. In *Proceedings of IEEE INFOCOM*, San Francisco, California, USA, April 2016.

[29] Martín Varela, Hyunwoo Nam, Henning Schulzrinne, and Toni Mäki. Generating Realistic Youtube-like Stall Patterns for Http Video Streaming Assessment. In *Proceedings of IEEE QOMEX*, Lisbon, Portugal, June 2016.

[30] Recording media streamed through PNM protocol. Retrieved July 12, 2016 from `http://all-streaming-media.com/streaming-media-faq/faq-pnm-protocol.htm`.

[31] Haakon Riiser. *Adaptive Bitrate Video Streaming over HTTP in Mobile Wireless Networks*. PhD thesis, University of Oslo, Oslo, Norway, June 2013.

[32] An Overview of Digital Rights Management. Retrieved July 12, 2016 from `http://www.encoding.com/digital-rights-management-drm/`.

[33] Michael C. Thornburgh. Adobe's Secure Real-Time Media Flow Protocol. IETF Draft, July 2013.

[34] Adobe Media Server family. Retrieved July 1, 2016 from `http://www.adobe.com/products/adobe-media-server-family.html`.

[35] Wowza media systems. Retrieved July 1, 2016 from `https://www.wowza.com/`.

[36] Dynamic RTMP Streaming. Retrieved July 1, 2016 from `https://support.jwplayer.com/customer/portal/articles/1430398-dynamic-rtmp-streaming`.

[37] Roger Pantos and Jr. William May. HTTP Live Streaming. IETF Draft, April 2015.

[38] Stefan Lederer, Christopher Müller, and Christian Timmerer. Dynamic Adaptive Streaming over HTTP Dataset. In *Proceedings of ACM MMSys*, Chapel Hill, North Carolina, USA, February 2012.

[39] W3C - Media Source Extensions. Retrieved September 2, 2015 from `http://w3c.github.io/media-source/`.

[40] Technical Note TN2224: Best Practices for Creating and Deploying HTTP Live Streaming Media for the iPhone and iPad. Retrieved June 22, 2016 from `https://developer.apple.com/library/ios/technotes/tn2224/_index.html`.

[41] IIS Smooth Streaming Technical Overview. Retrieved April 14, 2015 from `http://www.bogotobogo.com/VideoStreaming/Files/iis8/IIS_Smooth_Streaming_Technical_Overview.pdf`.

[42] Source code of Silverlight extensions. Retrieved August 31, 2015 from `http://slextensions.codeplex.com/SourceControl/latest#trunk/SLExtensions/AdaptiveStreaming/Heuristics/NetworkHeuristicsParams.cs`.

[43] jpcap – a network packet capture library. Retrieved September 2, 2015 from `http://jpcap.sourceforge.net/`.

[44] The Wireshark Network Analyzer. Retrieved September 2, 2015 from `https://www.wireshark.org/`.

[45] YouTube says HTML5 video ready for primetime, makes it default. Retrieved June 29, 2016 from `http://arstechnica.com/gadgets/2015/01/youtube-declares-html5-video-ready-for-primetime-makes-it-default/`.

[46] Ricky Yang and Harrison J. Son. YouTube's Live TV Streaming in Mobile Devices - HLS & Adaptive. Technical report, Netmanias, October 2013.

[47] Yao Liu, Fei Li, Lei Guo, Bo Shen, and Songqing Chen. A Comparative Study of Android and iOS for Accessing Internet Streaming Services. In *Proceedings of PAM*, PAM, Hong Kong, China, March 2013.

[48] Netem. Retrieved August 31, 2015 from `http://www.linuxfoundation.org/collaborate/workgroups/networking/netem`.

[49] Netflix Tech Blog. Retrieved June 22, 2016 from `http://techblog.netflix.com/`.

[50] David Ronca. A Brief History of Netflix Streaming. Technical report, Netflix, May 2013.

[51] Netflix Tchnical Center. Retrieved June 19, 2015 from `https://www.netflix.com/`.

[52] OMA Digital Rights Management. Retrieved August 31, 2015 from `http://technical.openmobilealliance.org/Technical/technical-information/release-program/current-releases/drm-v2-0`.

[53] UltraViolet. Retrieved August 31, 2015 from `https://www.myuv.com/en/us`.

[54] Internet video archive - Adaptive Bitrate Comparison Chart. Retrieved June 25, 2015 from `http://www.internetvideoarchive.com/documentation/video-api/progressive-download-vs-adaptive-bitrate/`.

[55] VP9 Video Codec. Retrieved July 5, 2016 from `http://www.webmproject.org/vp9/`.

[56] Cenc Initialization Data Format. Retrieved July 5, 2016 from `https://www.w3.org/TR/eme-initdata-cenc/#bib-CENC`.

[57] Widevine DRM. Retrieved July 5, 2016 from `http://www.widevine.com/wv_drm.html`.

[58] The State of MPEG-DASH 2016. Retrieved July 1, 2016 from `http://www.streamingmedia.com/Articles/Editorial/Featured-Articles/The-State-of-MPEG-DASH-2016-110099.aspx`.

[59] Encrypted Media Extensions. Retrieved July 1, 2016 from `https://w3c.github.io/encrypted-media/`.

[60] Akamai. Retrieved June 19, 2015 from `http://www.akamai.com/`.

[61] Limelight. Retrieved June 19, 2015 from `http://www.limelight.com/`.

[62] Sungsu Kim, Sin-Seok Seo, Joon-Myung Kang, Guy Pujolle, and James Won-Ki Hong. Autonomic Resource Allocation for Video Streaming Services in Content Delivery Networks. In *Proceedings IEEE of GIIS*, Choroni, Venezuela, December 2012.

[63] Ruben Torres, Alessandro Finamore, Jin Ryong Kim, Marco Mellia, and Sanjay Munafo, Maurizio M.and Rao. Dissecting Video Server Selection Strategies in the YouTube CDN. In *Proceedings of IEEE ICDCS*, Minneapolis, Minnesota, USA, June 2011.

[64] Vijay Kumar Adhikari, Sourabh Jain, Yingying Chen, and Zhi-Li Zhang. Reverse-Engineering the YouTube Video Delivery Cloud. In *Proceedings of IEEE ICME HotMD Workshop*, Barcelona, Spain, July 2011.

[65] Iperf. Retrieved June 19, 2015 from `https://iperf.sourceforge.net/`.

[66] MaxMind - GeoIP databases and Web services. Retrieved June 19, 2015 from `http://dev.maxmind.com/`.

[67] Vijay Kumar Adhikari, Sourabh Jain, and Zhi-Li Zhang. Where Do You "Tube"? Uncovering YouTube Server Selection Strategy. In *Proceedings of IEEE ICCCN*, Maui, Hawaii, July 2011.

[68] James F. Kurose and Keith W. Ross. *Computer Networking: A Top-Down Approach (6th Edition)*. Pearson, 6th edition, 2012.

[69] Krishna Balachandran, Doru Calin, Eunyoung Kim, and Kiran M. Rege. Clearmedia: A Proxy-based Architecture for Streaming Media Services over Wireless Networks. In *Proceedings IEEE of PIMRC*, Athens, Greece, September 2007.

[70] Ben Niven-Jenkins, Grant Watson, and Nabil Bitar. Use Cases for ALTO within CDNs. IETF Draft, April 2011.

[71] Jan Seedorf. Infrastructure-to-application information exposure from an ALTO-CDNI Perspective. IETF Draft, March 2012.

[72] Audrius Jurgelionis, Jukka-Pekka Laulajainen, Matti Hirvonen, and Alf Inge Wang. An Empirical Study of Netem Network Emulation Functionalities. In *Proceedings of IEEE ICCCN*, Maui, Hawaii, July 2011.

[73] Tcptrace. Retrieved June 19, 2015 from `http://www.tcptrace.org/`.

[74] Vijay Kumar Adhikari, Sourabh Jain, and Zhi-Li Zhang. YouTube Traffic Dynamics and Its Interplay with a Tier-1 ISP: An ISP Perspective. In *Proceedings of ACM SIGCOMM IMC*, New Delhi, India, September 2010.

[75] Vijay Kumar Adhikari, Sourabh Jain, Gyan Ranjan, and Zhi-Li Zhang. Understanding Data-center Driven Content Distribution. In *Proceedings of ACM CoNEXT Student Workshop*, Philadelphia, Pennsylvania, November 2010.

[76] Mohit Saxena, Umang Sharan, and Sonia Fahmy. Analyzing Video Services in Web 2.0: A Global Perspective. In *Proceedings of ACM MMSys NOSSDAV Workshop*, Braunschweig, Germany, May 2008.

[77] Li Erran Li, Z. Morley Mao, and Jennifer Rexford. Toward Software-Defined Cellular Networks. In *Proceedings of IEEE EWSDN*, Darmstadt, Germany, October 2012.

[78] 3GPP. 3GPP TS 23.203 v8.3.1, Technical Specication, Policy and charging control architecture (Release 8). Technical report, 3GPP, September 2008.

[79] MathWork - LTE System Toolbox. Retrieved April 14, 2015 from `http://www.mathworks.com/products/lte-system/`.

[80] OPNET - application and network performance. Retrieved April 14, 2015 from `http://www.opnet.com/`.

[81] Linksys WRT54GL Wireless-G Router. Retrieved June 26, 2015 from `http://www.linksys.com/us/p/P-WRT54GL/`.

[82] DD-WRT - a Linux based alternative OpenSource firmware. Retrieved June 26, 2015 from `http://www.dd-wrt.com/`.

[83] The netfilter.org project. Retrieved June 19, 2015 from `http://www.netfilter.org/`.

[84] Alessandro Finamore, Marco Mellia, Maurizio M. Munafò, Ruben Torres, and Sanjay G. Rao. YouTube Everywhere: Impact of Device and Infrastructure Synergies on User Experience. In *Proceedings of ACM SIGCOMM IMC*, Berlin, Germany, November 2011.

[85] Michael Zink, Kyoungwon Suh, Yu Gu, and Jim Kurose. Watch Global, Cache Local: YouTube Network Traffic at a Campus Network - Measurements and Implications. *Computer Science Department Faculty Publication Series.Paper 177*, 2008.

[86] Ashwin Rao, Arnaud Legout, Yeon-sup Lim, Don Towsley, Chadi Barakat, and Walid Dabbous. Network Characteristics of Video Streaming Traffic. In *Proceedings of ACM CoNext*, CoNEXT, Tokyo, Japan, December 2011.

[87] Mohammad Ashraful Hoque, Matti Siekkinen, Jukka K. Nurminen, and Mika Aalto. Investigating Streaming Techniques and Energy Efficiency of Mobile Video Services. *Computing Research Repository - arXiv:1209.2855*, 2012.

[88] Te-Yuan Huang, Nikhil Handigol, Brandon Heller, Nick McKeown, and Ramesh Johari. Confused, Timid, and Unstable: Picking a Video Streaming Rate is Hard. In *Proceedings of ACM SIGCOMM IMC*, Boston, Massachusetts, USA, November 2012.

[89] Te-Yuan Huang, Ramesh Johari, and Nick McKeown. Downton Abbey Without the Hiccups: Buffer-based Rate Adaptation for HTTP Video Streaming. In *Proceedings of ACM SIGCOMM FhMN Workshop*, Hong Kong, China, August 2013.

[90] Xing Xu, Yurong Jiang, Tobias Flach, Ethan Katz-Bassett, David Choffnes, and Ramesh Govindan. Investigating Transparent Web Proxies in Cellular Networks. Technical Report 14-944, University of Southern California, April 2014.

[91] Cisco ASR 5000 Multimedia Core Platform. Retrieved June 25, 2016 from `http://www.cisco.com/c/en/us/products/collateral/wireless/asr-5000-series/data_sheet_c78-606223.html`.

[92] Franck Le, Erich Nahum, Vasilis Pappas, Maroun Touma, and Dinesh Verma. Experiences Deploying a Transparent Split TCP Middlebox and the Implications for NFV. In *Proceedings of ACM SIGCOMM Workshop on HotMiddlebox*, London, United Kingdom, August 2015.

[93] Arash Molavi Kakhki, Fangfan Li, David Choffnes, Alan Mislove, and Ethan Katz-Bassett. BingeOn Under the Microscope: Understanding T-Mobile's Zero-Rating

Implementation. In *Proceedings of ACM SIGCOMM Workshop on Internet-QoE*, Florianópolis, Brazil, August 2016.

[94] Justine Sherry, Chang Lan, Raluca Ada Popa, and Sylvia Ratnasamy. BlindBox: Deep Packet Inspection over Encrypted Traffic. In *Proceedings of ACM SIGCOMM*, London, United Kingdom, August 2015.

[95] Visibility into Encrypted Traffic. Retrieved June 24, 2016 from `https://www.plixer.com/blog/network-security-forensics/visibility-encrypted-traffic/#more-28770`.

[96] OpenFlow Switch Specification Version 1.4.0. Retrieved June 19, 2015 from `https://www.opennetworking.org/`.

[97] Yubing Wang. Survey of Objective Video Quality Measurements. Technical report, EMC Corporation Hopkinton, 2006.

[98] Athula Balachandran, Vyas Sekar, Aditya Akella, Srinivasan Seshan, Ion Stoica, and Hui Zhang. A Quest for an Internet Video Quality-of-experience Metric. In *Proceedings of ACM HotNets Workshop*, Redmond, WA, USA, October 2012.

[99] René Serral-Gracià, Eduardo Cerqueira, Marília Curado, Marcelo Yannuzzi, Edmundo Monteiro, and Xavier Masip-Bruin. An Overview of Quality of Experience Measurement Challenges for Video Applications in IP Networks. In *Proceedings of WWIC*, Lulea, Sweden, June 2010.

[100] Ali Reza Sharafat, Saurav Das, Guru Parulkar, and Nick McKeown. MPLS-TE and MPLS VPNS with Openflow. In *Proceedings of ACM SIGCOMM*, Toronto, Ontario, Canada, August 2011.

[101] Saurav Das, Yiannis Yiakoumis, Guru Parulkar, Nick McKeown, Preeti Singh, Daniel Getachew, and Premal Dinesh Desai. Application-aware Aggregation and Traffic Engineering in a Converged Packet-Circuit Network. In *Proceedings of IEEE OFC/NFOEC*, Los Angeles, USA, March 2011.

[102] Hidetsugu Sugiyama. Programmable Network Systems: Through the Junos SDK and Junos Space SDK. In *Proceedings of IEEE WTC*, Miyazaki, Japan, March 2012.

[103] Citrix. Bytemobile Mobile Analytics Report. Technical report, Citrix Systems, May 2012.

[104] Shigang Chen and Klara Nahrstedt. An Overview of Quality-of-Service Routing for the Next Generation HighSpeed Networks: Problems and Solutions. *IEEE Network*, 12(6):64–79, November 1998.

[105] Victoria Fineberg. A Practical Architecture for Implementing End-to-End QoS in an IP Network. *IEEE Communications Magazine*, 40(1):122–130, Jan 2002.

[106] Zafar Ayyub Qazi, Jeongkeun Lee, Tao Jin, Gowtham Bellala, Manfred Arndt, and Guevara Noubir. Application-awareness in SDN. In *Proceedings of ACM SIGCOMM*, Hong Kong, China, August 2013.

[107] About NOX. Retrieved June 19, 2015 from `http://www.noxrepo.org/`.

[108] Michael Jarschel, Florian Wamser, Thomas Höhn, Thomas Zinner, and Phuoc Tran-Gia. SDN-based Application-Aware Networking on the Example of YouTube Video Streaming. In *Proceedings of IEEE EWSDN*, Berlin, Germany, October 2013.

[109] Application-aware Routing in Software-defined Networks. Retrieved June 19, 2015 from `http://www.aricent.com/pdf/Aricent_Whitepaper_-_Application_Aware_Routing_in_SDN.pdf`.

[110] Sushant Jain, Alok Kumar, Subhasree Mandal, Joon Ong, Leon Poutievski, Arjun Singh, Subbaiah Venkata, Jim Wanderer, Junlan Zhou, Min Zhu, Jon Zolla, Urs Hölzle, Stephen Stuart, and Amin Vahdat. B4: Experience with a Globally-deployed Software Defined Wan. In *Proceedings of ACM SIGCOMM*, August 2013.

[111] Shuhao Liu and Baochun Li. On scaling software-Defined Networking in wide-area networks. *Tsinghua Science and Technology*, 20(3):221–232, June 2015.

[112] Henrique Rodrigues, Inder Monga, Abhinava Sadasivarao, Sharfuddin Syed, Chin Guok, Eric Pouyoul, Chris Liou, and Tajana Rosing. Traffic Optimization in Multi-layered WANs Using SDN. In *Proceedings of IEEE HOTI*, California, USA, August 2014.

[113] Hesham Mekky, Fang Hao, Sarit Mukherjee, Zhi-Li Zhang, and T.V. Lakshman. Application-aware Data Plane Processing in SDN. In *Proceedings of ACM SIGCOMM Workshop on HotSDN*, Chicago, Illinois, USA, August 2014.

[114] Micorsoft Silverlight. Retrieved November 2, 2014 from `http://www.microsoft.com/silverlight/`.

[115] Video Streaming Analysis Report. Retrieved November 12, 2014 from `http://goo.gl/u0qL3T`.

[116] Te-Yuan Huang, Ramesh Johari, Nick McKeown, Matthew Trunnell, and Mark Watson. A Buffer-based Approach to Rate Adaptation: Evidence from a Large Video Streaming Service. In *Proceedings of ACM SIGCOMM*, Chicago, Illinois, USA, August 2014.

[117] IIS Smooth Streaming. Retrieved November 12, 2014 from `http://www.iis.net/downloads/microsoft/smooth-streaming/`.

[118] IIS smooth streaming HD sample content. Retrieved July 2, 2015 from `https://www.microsoft.com/en-us/download/details.aspx?id=18199`.

[119] Netlimiter. Retrieved November 2, 2014 from `http://www.netlimiter.com/`.

[120] Fiddler - Connection Simulator. Retrieved November 2, 2014 from `http://logic-worx.com/tools-and-apps/fiddler-connection-simulator/`.

[121] Ouldooz Baghban Karimi, Jiangchuan Liu, and Chonggang Wang. Seamless Wireless Connectivity for Multimedia Services in High Speed Trains. *IEEE Journal on Selected Areas in Communications*, 30(4):729–739, May 2012.

[122] Kyung-Hwa Kim, Hyunwoo Nam, and H. Schulzrinne. WiSlow: A Wi-Fi Network Performance Troubleshooting Tool for End Users. In *Proceedings of IEEE INFOCOM*, Toronto, Canada, April 2014.

[123] Amazon Mechanical Turk. Retrieved November 12, 2014 from `https://www.mturk.com/mturk/welcome`.

[124] Tobias Hossfeld, Dominik Strohmeier, Alexander Raake, and Raimund Schatz. Pippi Longstocking Calculus for Temporal Stimuli Pattern on YouTube QoE: 1+1=3 and $1\cdot4\neq4\cdot1$. In *Proceedings of ACM MMSys MoVid Workshop*, Oslo, Norway, February 2013.

[125] Toon De Pessemier, Katrien De Moor, Wout Joseph, Lieven De Marez, and Luc Martens. Quantifying the Influence of Rebuffering Interruptions on the User's Quality of Experience During Mobile Video Watching. *Broadcasting, IEEE Transactions on*, 59(1):47–61, March 2013.

[126] Alessandro Floris, Luigi Atzori, Giaime Ginesu, and Daniele D. Giusto. QoE Assessment of Multimedia Video Consumption on Tablet Devices. In *Proceedings of IEEE Globecom QoEMC Workshop*, Anaheim, California, December 2012.

[127] Pengpeng Ni, Ragnhild Eg, Alexander Eichhorn, Carsten Griwodz, and Pål Halvorsen. Flicker Effects in Adaptive Video Streaming to Handheld Devices. In *Proceedings of ACM Multimedia*, Scottsdale, Arizona, USA, November 2011.

[128] Ricky K. P. Mok, Xiapu Luo, Edmond W. W. Chan, and Rocky K. C. Chang. QDASH: A QoE-aware DASH System. In *Proceedings of ACM MMSys*, Chapel Hill, North Carolina, February 2012.

[129] Marie-Neige Garcia, Francesca De Simone, Samira Tavakoli, Nicolas Staelens, Sebastian Egger, Kjell Brunnström, and Alexander Raake. Quality of Experience and HTTP Adaptive Streaming: A Review of Subjective Studies. In *Proceedings of IEEE QoMEX*, Singapore, Sep 2014.

[130] Jong-Min Jeong and Jong-Deok Kim. Effective bandwidth measurement for Dynamic Adaptive Streaming over HTTP. In *Proceedings of IEEE ICOIN*, Siem Reap, Cambodia, January 2015.

[131] Guibin Tian and Yong Liu. Towards Agile and Smooth Video Adaptation in Dynamic HTTP Streaming. In *Proceedings of ACM CoNEXT*, Nice, France, December 2012.

[132] Luca De Cicco, Vito Caldaralo, Vittorio Palmisano, and Saverio Mascolo. ELASTIC: A Client-Side Controller for Dynamic Adaptive Streaming over HTTP (DASH). In *Proceedings of IEEE PV*, San Jose, California, USA, December 2013.

[133] Vimeo Player API. Retrieved July 1, 2016 from `https://developer.vimeo.com/player`.

[134] Chrome extensions. Retrieved April 20, 2015 from `https://developer.chrome.com/extensions`.

[135] YouTube IFrame Player APIs. Retrieved April 21, 2015 from `https://developers.google.com/youtube/iframe_api_reference`.

[136] YouTube JavaScript Player API Reference. Retrieved August 31, 2016 from `https://developers.google.com/youtube/js_api_reference`.

[137] K. M. Chan and Jack Y. B. Lee. Improving adaptive http streaming performance with predictive transmission and cross-layer client buffer estimation. *Multimedia Tools and Applications*, 75(10):5917–5937, 2016.

[138] YouTube video ads. Retrieved April 14, 2015 from `https://support.google.com/displayspecs/answer/6244541?hl=en&rd=2&ref_topic=6244532`.

[139] Non-skippable in-stream ads in YouTube. Retrieved June 19, 2015 from `https://support.google.com/youtube/answer/188038?hl=en`.

[140] Chrome and Safari AdBlock. Retrieved April 16, 2015 from `https://getadblock.com/`.

[141] Chrome extensions - webRequest API. Retrieved April 20, 2015 from `https://developer.chrome.com/extensions/webRequest`.

[142] YouTube live encoder settings, bitrates and resolutions. Retrieved April 14, 2015 from `https://support.google.com/youtube/answer/2853702?hl=en`.

[143] Google Video Quality Report. Retrieved July 24, 2015 from `http://www.google.com/get/videoqualityreport/`.

[144] Netflix ISP speed index. Retrieved July 12, 2016 from `https://ispspeedindex.netflix.com/`.

[145] ITU-T P.10/G.100 Amendment 1: New Appendix I Definition of Quality of Experience (QoE). Retrieved June 29, 2016 from `https://www.itu.int/rec/T-REC-P.10-200701-S!Amd1`.

[146] ITU-T Recommendation E.800 Definitions of terms related to quality of service. Retrieved June 29, 2016 from `http://www.itu.int/rec/T-REC-E.800-200809-I`.

[147] ITU-T Recommendation BT.500-13 Methodology for the subjective assessment of the quality of television pictures. Retrieved June 29, 2016 from `https://www.itu.int/rec/R-REC-BT.500-13-201201-I/en`.

[148] ITU-T P.910 Subjective video quality assessment methods for multimedia applications. Retrieved June 29, 2016 from `https://www.itu.int/rec/T-REC-P.910-200804-I/en`.

[149] ITU-T G.107 The E-model: a computational model for use in transmission planning. Retrieved June 29, 2016 from `https://www.itu.int/rec/T-REC-G.107`.

[150] ITU-T P.862 Perceptual evaluation of speech quality (PESQ): An objective method for end-to-end speech quality assessment of narrow-band telephone networks and speech codecs. Retrieved June 29, 2016 from `http://www.itu.int/rec/T-REC-P.862`.

[151] Kuan-Ta Chen, Chun-Ying Huang, Polly Huang, and Chin-Laung Lei. Quantifying Skype User Satisfaction. In *Proceedings of ACM SIGCOMM*, Pisa, Italy, September 2006.

[152] Florin Dobrian, Vyas Sekar, Asad Awan, Ion Stoica, Dilip Joseph, Aditya Ganjam, Jibin Zhan, and Hui Zhang. Understanding the Impact of Video Quality on User Engagement. In *Proceedings of ACM SIGCOMM*, Toronto, Ontario, Canada, August 2011.

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

[154] Muhammad Zubair Shafiq, Jeffrey Erman, Lusheng Ji, Alex X. Liu, Jeffrey Pang, and Jia Wang. Understanding the Impact of Network Dynamics on Mobile Video User Engagement. In *Proceedings of ACM SIGMETRICS*, Austin, Texas, USA, June 2014.

[155] S. Shunmuga Krishnan and Ramesh K. Sitaraman. Understanding the Effectiveness of Video Ads: A Measurement Study. In *Proceedings of ACM IMC*, Barcelona, Spain, October 2013.

[156] 3 Things You Need to Know About Making a Marketing Video Convert. Retrieved June 19, 2016 from `http://www.sailthru.com/marketing-blog/3-things-need-know-making-marketing-video-convert/`.

[157] Tobias Hossfeld, Sebastian Egger, Raimund Schatz, Markus Fiedler, Kathrin Masuch, and Charlott Lorentzen. Initial Delay vs. Interruptions: Between the Devil and the Deep Blue Sea. In *Proceedings of IEEE QoMEX*, Melbourne, Australia, July 2012.

[158] Liu Yitong, Shen Yun, Mao Yinian, Liu Jing, Lin Qi, and Yang Dacheng. A Study on Quality of Experience for Adaptive Streaming Service. In *Proceedings of IEEE ICC*, Budapest, Hungary, June 2013.

[159] Abdul Rehman and Zhou Wang. Perceptual Experience of Time-varying Video Quality. In *Proceedings of IEEE QoMEX*, Klagenfurt, Austria, July 2013.

[160] David C. Robinson, Yves Jutras, and Viorel Craciun. Subjective Video Quality Assessment of HTTP Adaptive Streaming Technologies. *Bell Labs Technical Journal*, 16(4):5–23, 2012.

[161] Netflix is shutting down its public APIs. Retrieved April 26, 2015 from `https://gigaom.com/2014/11/14/netflix-is-shutting-down-its-public-api-today/`.

[162] FCC Measuring Broadband America. Retrieved April 21, 2015 from `http://www.fcc.gov/measuring-broadband-america`.

[163] Mohan Dhawan, Justin Samuel, Renata Teixeira, Christian Kreibich, Mark Allman, Nicholas Weaver, and Vern Paxson. Fathom: A Browser-based Network Measurement Platform. In *Proceedings of ACM IMC*, Boston, Massachusetts, USA, November 2012.

[164] Barbara Staehle, Matthias Hirth, Rastin Pries, Dirk Staehle, Barbara Staehle, Matthias Hirth, Rastin Pries, and Dirk Staehle. YoMo: A YouTube Application Comfort Monitoring Tool. Retrieved July 1, 2016 from `http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.163.3983`.