March 2018

On the Performance of Adaptive Bitrate Streaming and Parallel Cloud Applications

Cong Wang
University of Massachusetts Amherst

ON THE PERFORMANCE OF ADAPTIVE BITRATE STREAMING AND PARALLEL CLOUD APPLICATIONS

A Dissertation Presented

by

CONG WANG

Submitted to the Graduate School of the
University of Massachusetts Amherst in partial fulfillment
of the requirements for the degree of

DOCTOR OF PHILOSOPHY

February 2018

Electrical and Computer Engineering

ON THE PERFORMANCE OF ADAPTIVE BITRATE STREAMING AND PARALLEL CLOUD APPLICATIONS

A Dissertation Presented

by

CONG WANG

Approved as to style and content by:

Michael Zink, Chair

Lixin Gao, Member

David Irwin, Member

Ramesh Sitaraman, Member

C. V. Hollot, Department Head
Electrical and Computer Engineering

To my family.

ACKNOWLEDGMENTS

I would like to express the deepest appreciation to my PhD advisor, Professor Michael Zink, who always encourages me and inspires me with full of valuable and insightful ideas. Without his guidance and persistent help this dissertation would not have been possible.

I am heartily thankful to my committee members Professor Lixin Gao, Professor David Irwin, and Professor Ramesh Sitaraman for presiding over my dissertation. I would like to thank my Master advisor, Professor Tilman Wolf, and Professor Weibo Gong for their constructive advice and invaluable help on both of my research and future career.

I also would like to acknowledge my labmates for the knowledge and experience they have shared with me. In particular, I want to thank Divyashri Bhat, Dilip Kumar Krishnappa, Eric Adams, Amr Rizk from Technische Universität Darmstadt and Fraida Fund from NYU Polytechnic School of Engineering, who worked together with me on so many great projects.

Finally, I appreciate all of the sincere support from my family and friends. Special thanks to Jieqi Kang, Shan Lu, Xiaolin Xu, Shuo Li, Yang Lei and Kan Fu, this dissertation pales in comparison to what I gained from them.

ABSTRACT

ON THE PERFORMANCE OF ADAPTIVE BITRATE STREAMING AND PARALLEL CLOUD APPLICATIONS

FEBRUARY 2018

CONG WANG

B.Sc., YANSHAN UNIVERSITY

M.Sc., UNIVERSITY OF MASSACHUSETTS AMHERST

Ph.D., UNIVERSITY OF MASSACHUSETTS AMHERST

Directed by: Professor Michael Zink

As shown in recent studies, video streaming has become the biggest category of backbone Internet traffic in US. With the boost of Internet and video-on-demand (VoD) technologies, millions of people can watch videos on a variety of client devices, such as smart phones, laptops, tablets, over different access networks, such as Wifi, Ethernet and cellular networks. Each of these client devices and access networks has different service requirements for video streaming, such as bitrates, resolutions, network bandwidth and CPU capabilities. To meet these diversified client demands and high service quality expectations, it becomes critical to build efficient, flexible and scalable video delivering platforms in the future video services.

Traditional stateful video streaming protocols, such as Real Time Messaging Protocol (RTMP) or Real Time Streaming Protocol (RTSP), are no longer sufficient for today's VoD service requirements. This is primarily because: *i)* traditional video streaming protocols

require special media servers to host the video contents, such as Windows Media Server, Flash media Server, Wowza, etc. *ii)* Protocols like RTSP usually require specific firewall configurations and plug-ins for video playback. *iii)* These protocols can only support fixed resolutions or bitrates throughout the video playback.

To ensure high quality video delivery to different end user devices, most of the content providers have switched to Adaptive Bitrate (ABR) streaming approach, which is specifically designed to enable a smooth user playback experience. ABR streaming is capable of monitoring the available network bandwidth, and therefore choosing the best suitable quality according to the network capacity. An HTTP based ABR streaming contents can be easily hosted on most of the regular web HTTP servers as well as web caches. Moreover, since HTTP-based video requests are client oriented, ABR mechanism can easily get access to video contents without special configurations on firewalls or proxies.

On the other hand, the rapid growth of video services requires large amounts of physical resources to host the videos and to serve the large population of clients. As with many other services, resources are usually provisioned such that peak requests can be handled without impact on the quality of the provided VoD service; on the contrary, analyses of VoD services have shown that the overall number of requests can vary significantly over time. Therefore, as an alternative solution, cloud services, such as Amazon Web Services (AWS), Google Cloud and Microsoft Azure Cloud, provide resources that can be quickly deployed and released according to the service load. This approach has been widely used for web and VoD services, such as Netflix, GitHub and Dropbox, since the content providers do not have to deploy and maintain expensive physical infrastructures. Instead, they only need to pay for the usage of the allocated cloud resources.

Hosting the cloud services requires tremendous energy consumption. This has been widely considered as the primary design constraint for scaling up today's cloud platforms. This is mainly due to the high cost and carbon footprint of powering and cooling these systems. Recent studies have shown that, if historical trends continue, the power requirements

of an exascale platform in 2020 would be 200MW with an annual electricity bill greater than $2.5B primarily composed of "dirty" energy sources. These costs are unsustainable for even the highest-value applications. Therefore, improving the energy efficiency has become critical in today's cloud world.

Taking into account these confounding facts and service requirements, in this dissertation, we seek to evaluate and improve the performance for both ABR video streaming and cloud applications. First, we present a set of measurement studies for ABR streaming applications. Using the data from the application, network, and physical layers in different network environments, we identify the key factors that can impact the quality of video delivering services. Then we develop and evaluate a new ABR streaming quality adaptation algorithm to improve the user playback experience. Our results show that the proposed adaptation algorithm can improve Quality of Experience (QoE) by up to 96% in terms of QoE metric *spectrum*.

In the second part of this dissertation, we explore the options for better application efficiency for ABR video transcoding services from the renewable energy perspective. We define a set of dynamic and static energy management policies that apply to distributed ABR video transcoding tasks. In addition, we extend the power management mechanisms to parallel cloud applications and show the general applicability of our approach. We show that, by utilizing the renewable energy sources, the transcoding grid energy usage can be reduced by 73-83%, and the corresponding energy cost can be reduced by 14-28% with satisfying viewer experience. When coming to the parallel cloud applications, with the effective use of power management policies, the total cost can be reduced by up to 67% comparing to when using fixed prices. This can be achieved by only increasing up to 17% of the application runtime and 9% of the total energy consumption.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Figure **Page**

CHAPTER 1

INTRODUCTION

1.1 Motivation

Coupling with today's high capacity Internet, video on demand (VoD) systems allow users to select and watch videos with their own preferences. With this high flexibility, VoD has become the most popular way of entertainment in today's Internet. According to a recent Sandvine report [14], 71% of the downstream Internet traffic at peak hours (8 PM to 1 AM EST) in the US is real time entertainment such as live streaming and video on demand. Also, over 50% of the peak period downstream traffic is composed of by Netflix and YouTube video streams.

With the rapid growth of VoD traffic and user population, it becomes critical to maintain high user satisfaction while delivering a large amount of video contents. Krishnan et al. [59] show that rebuffering events significantly reduce the user experience. Users are less likely to watch the complete video when experiencing long buffering time, and are less likely to return to the content provider's site if they experience a long start up wait time. On the other hand, with the support of a variety of content/network providers, users are able to watch video from various client platforms, such as smart phones, laptops, tablets, over a variety of access networks, such as Wifi, Ethernet and cellular networks. Each of these client devices and access networks has unique requirements for video streaming, such as bitrates, resolutions, network bandwidth and CPU capabilities. To meet these client demands and high service quality expectations, building more efficient, flexible and scalable video delivering platforms becomes increasingly important for future video services.

1.2 Adaptive Bitrate (ABR) Streaming in the Cloud

In order to fulfill the complex service requirements of ABR video delivering, virtually all the video content providers have switched to the adaptive bitrate (ABR) streaming approach. Unlike the traditional video streaming techniques, such as Real Time Messaging Protocol (RTMP) or Real Time Streaming Protocol (RTSP), ABR streaming mechanism encodes the videos into short segments with typical length varying from 2 seconds to 10 seconds, depending on the implementation. Each segment is encoded into multiple quality layers with different bitrates. During a streaming session, the client monitors the available network bandwidth and adjusts the requested video quality, according to the server-client link bandwidth.

The high demand of VoD services requires tremendous amount of storage space, network bandwidth and computation resources, especially for ABR contents which are encoded with Advanced Video Codec (AVC) [69]. This is because in case of AVC, videos are required to be encoded into multiple copies with different qualities in order to adapt to different service requirements. According to [38], there are about 500 hours of videos uploaded to YouTube every minute. As for another major video content provider, Netflix needs to deliver over 3×10^8 TB data to end users [37]. With traditional approaches, content providers have to supply a large amount of physical resources in order to meet the high demand of video services. As with many other services, resources are provided such that peak requests can be handled without impact on the quality of the VoD service; on the contrary, analyses of VoD services have shown that the overall number of requests can vary significantly over time, which can lead to significant over-provisioning of resources.

As an alternative solution, cloud services, such as Amazon Web Services (AWS), Google Cloud and Microsoft Azure Cloud, provide resources that can be quickly deployed and released according to customer needs. This approach has been widely used for web and VoD services, such as Netflix, GitHub and Dropbox, since the content providers do not have to

maintain expensive physical infrastructures. Instead, they only need to pay for the usage of the allocated cloud resources.

Cloud platforms provide high flexibility to customers by allowing them to pay only for the time period they actually use the hardware and for the amount of data that is actually transmitted. However, the deployment and maintainance of cloud services results in tremendous energy consumption, which has been widely regarded as the primary design constraint for scaling up the capacity of cloud platforms. This is mainly due to the cost and carbon footprint for powering and cooling these systems [18]. To illustrate the magnitude of this constraint, recent estimates project that, if historical trends continue, the power requirements of an exascale platform in 2020 would be 200MW [18] with an annual electricity bill greater than $2.5B [17], primarily composed of "dirty" energy sources. Such high costs are unsustainable for even the highest-value applications. Therefore, improving the energy efficiency has become critical in today's cloud world.

1.3 Dissertation Contributions and Outlines

As we outline in Section 3, while researchers have recognized the importance of improving performance and efficiency for ABR streaming services and cloud based applications, there has been little prior research on systematic characterization, design, implementation and analysis of cloud-based video eco-system. Thus, in this dissertation, we analyze and improve the performance of ABR streaming in the cloud environments. In the first part of this dissertation, we present measurements collected from ABR streaming applications. Using data from the application, network, and physical layers, in different network environments, we identify the key factors that directly impact the quality of video delivering services. Then we develop and evaluate a new quality adaptation algorithm aiming to improve the user experience, especially for the conditions with varying network capacities.

In the second part of this dissertation, we explore the options for better application efficiency for ABR video transcoding services from the renewable energy perspective. We

define a set of dynamic and static energy management policies that apply to distributed ABR video transcoding tasks. In addition, we extend the power management mechanisms to parallel cloud applications and show the general applicability of our approach. We show that, by utilizing the renewable energy sources, the transcoding grid energy usage can be reduced by 73-83%, and the corresponding energy cost can be reduced by 14-28% with satisfying viewer experience. When coming to the parallel cloud applications, with the effective use of power management policies, the total cost can be reduced by up to 67% comparing to when using fixed prices. This can be achieved by only increasing up to 17% of the application runtime time and 9% of the total energy consumption.

This dissertation includes eight chapters. In Chapters 1-3, we provide general information of this dissertation. Chapter 1 gives an overall introduction and motivation of this dissertation. Chapter 2 introduces the background information and provides explanations of key concepts that will be commonly used in later parts of this dissertation. In Chapter 3, we discuss the research works that are closely related to this dissertation topic, and introduce how our approach differs from the previous research works.

In Chapter 4, we provide a measurement study on two campus testbeds to show how ABR streaming applications perform in a mobile, long distance wireless environment. Using measurements from the application, network, and wireless physical layers, we identify characteristics of the cellular data network that directly impact the quality of the video service, and therefore present our approach for further improvement and optimization.

In Chapter 5, we perform a detailed analysis of the interplay between the ABR streaming and its underlying networking protocols. And then we develop a new ABR streaming rate adaption algorithm with accurate rate estimation and retransmission mechanisms to improve the user quality of experience by addressing many of our critical observations. In addition, we present a comprehensive evaluation of our proposed ABR streaming algorithm along with a set of other state-of-the-art algorithms.

In Chapter 6, we introduce the challenges for utilizing renewable energy sources for cloud based ABR video transcoding applications, and then propose a set of power management policies to enable the renewable energy usage and to improve the energy efficiency for these applications. We provide experimental evaluation of the policies in a small scale prototype cluster, as well as trace based simulation using a user behavior trace collected from a large scale video content delivery network.

In Chapter 7, we further extend the power management policies proposed in Chapter 6, to scenarios with parallel cloud applications. We provide further evaluation on the policies and show the general applicability of these policies for such parallel cloud applications.

In the last chapter, we conclude this dissertation, and provide possible directions for future research works.

CHAPTER 2

BACKGROUND

In this chapter, we introduce background information in this dissertation. We first introduce the ABR streaming and its QoE measurement metrics. Then, we present the cloud services that we use in this dissertation and their characteristics. Finally, we introduce the energy monitoring and controlling mechanisms for cloud server nodes.

2.1 ABR Streaming and DASH

In order to ensure a high user experience, many content providers have switched to ABR streaming, which has been widely implemented by modern video content providers, such as Microsoft's Smooth Streaming [105], Apple's HTTP Live Streaming [24] and Adobe's HTTP Dynamic Streaming [1]. Comparing to the traditional UDP based, single layer, real time media transfer protocols, e.g., RTP or RTSP, today's ABR streaming technologies are built on top of the Hyper Text Transfer Protocol (HTTP) and Transmission Control Protocol (TCP), aiming to provide reliable transmission.

TCP based applications perform steadily over stable network conditions. However, for AVC based ABR streaming, in case of low network capacity or high bandwidth fluctuations, clients can be exposed to frequent packet losses which leads to interrupted playback experience. Therefore, the ABR streaming approach transcodes single layer, high quality videos into multiple copies with different bitrates, then for each bitrate layer, the video is encoded into small segments with short duration, typically between 2 and 10 seconds, depending on the actual implementation. With ABR streaming, different client devices, varying from cell phones to PCs or large screen HD TVs, are all able to request their most

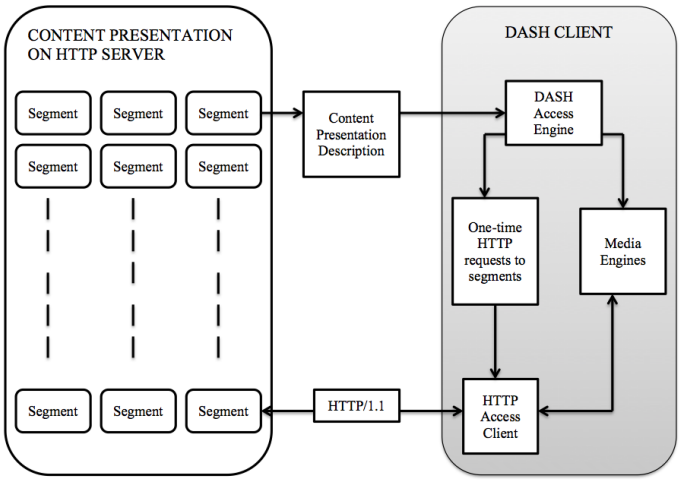


Figure 2.1: DASH Flow Diagram

suitable bitrates from the same video server according to their resolutions, processor capabilities and network conditions.

Prior to the downloading of video contents, clients devices are able to retrieve the Media Presentation Description (MPD) file containing a list of all available video and audio bitrates and segments. The actual video segments are requested by sending HTTP GET requests to the appropriate video server or cache, and, upon arrival at the client, stored in the client buffer. Typically the client playback will not start until achieving a certain buffer filling level. To quickly accumulate the client buffer, ABR algorithms usually select the lowest available bitrate at the beginning of playback. While playing, the ABR streaming algorithm continuously monitors the available network bandwidth, and make decision on the next bitrate request accordingly. In the case of network congestion, instead of pausing the playback, the ABR streaming mechanism reduces the requested quality bitrate, and therefore can maintain a smooth playback experience for users.

One of the most popular implementations of ABR streaming is the MPEG's Dynamic Streaming over HTTP (DASH) standard [89]. Its popularity can mostly be attributed to the facts that *i)* DASH-format videos can be streamed from any kind of HTTP server that are hosting the contents; *ii)* adaptation logic resides in the client, which makes DASH highly

scalable; and *iii)* it is an open standard. In this case, the quality of a video, along with resulting streaming bandwidth, can be adapted based on bottleneck bandwidth, server capacity, and client resources. Figure 2.1 depicts the DASH streaming flow process. The DASH server hosts two components: A Media Presentation Description (MPD) manifest file which contains the index of all the available bitrate (qualities), segment lengths, format, URL, etc; and video segments which contain the actual multimedia contents in the form of single or multiple files based on the encoder [63]. The client periodically (2 – 10 seconds) probes the available bandwidth and local buffer condition, and then selects a suitable quality level from the MPD file. Finally it downloads the desired content from the HTTP server, and plays the video. Therefore, with the benefits of lightweight HTTP server, and the flexibility of bitrate adaptation, DASH is able to provide high resilience to network variations and smooth playback experience.

2.2 Quality of Service (QoS) and Quality of Experience (QoE)

With the increasing popularity of video streaming services, one of the critical goals for content providers is to ensure a high-quality experience for end users. As stated in Chapter 1, users are highly likely to leave the content provider's site if they experience frequent playback pausing and rebuffering.

In order to quantitatively measure the performance of video streaming services, we look into two metrics. The first one is Quality-of-Service (QoS), which was proposed by the International Telecommunication Union (ITU) in 1994 [51]. QoS considers network and video playback aspects in order to decide the video delivery performance, such as, bitrate, bandwidth, playback pausing, etc.

On the other hand, Quality-of-Experience (QoE) is a set of performance metrics that focuses on the perspective of user satisfaction. The current measurement studies typically classify QoE into two categories: subjective QoE and objective QoE. Subjective QoE is based on users' opinions. One of the most commonly used subjective QoE metric is *Mean*

Opinion Score (MOS) [102]. In this case, the QoE is evaluated by user's ratings on a scale from 1 to 5. The higher the MOS, the higher the QoE.

In this dissertation, we focus on the second QoE category with objective metrics, i.e., metrics that can be quantified without subjective assessment. This is because objective QoE evaluations can be automated, therefore, they are easier to be integrated into service improvements. Here we list the metrics we consider in this dissertation.

- **Video initiation time.** In order to provide users with a smooth playback experience, media players usually buffer the initial part of the video before they actually start displaying the content. Most players set a certain buffer level threshold. As soon as the buffer level accumulates beyond the threshold, the video playback starts. The time period between when the first segment is buffered and when playback starts is described as video initiation time.

- **Video rebuffering.** A video segment has to be downloaded before it is rendered and presented onto the user's screen. If the segment download speed is slower than the video playback, the buffer level drops. To deal with buffer depletion, video players pause the playback and wait for the next frame to arrive when buffer level is critically low. This is named as rebuffering. Based on the video download rate, rebuffering can happen multiple times during the playback of a video. Rebuffering events can significantly reduce the user experience since the video playback "freezes" [59].

- **Average bitrate.** The average bitrate measures the number of bits per unit playback time, and is a typical metric for objective video playback quality. Usually higher average bitrates ensure a better view from the users. However, higher bitrates require better network bandwidth and higher CPU capacity on the client devices. Choosing inappropriate bitrates can either result in low video playback quality or frequent video rebuffering.

- **Quality switching.** In the case of ABR streaming, the video quality (bitrate) is chosen by the client algorithm, which is a reflection of network capacity. Therefore, under unstable network conditions, the video quality can vary in order to avoid video rebuffering. The number of quality changes is identified as detrimental to QoE by viewers by Zink et al. [107].

- **Spectrum.** Another video quality metric we considered is the presentation quality of video playback. Here we employ the metric, named *spectrum*, proposed by Zink et al. [107]. Spectrum provides a simple yet efficient way to determine the video quality from the viewer's perspective. The authors believe that two factors reduce the user experience: the video quality degradation and the frequency of quality switches. As shown in a comparison study, the spectrum reflects perceived quality better than PSNR. The overall spectrum of quality adaptive video v can be represented as:

$$s(v) = \sum_{t=1}^{T} z_t (h_t - \frac{1}{\sum_{i=1}^{T} z_i} (\sum_{j=1}^{T} z_j h_j))^2 \tag{2.1}$$

where the two terms h_t and z_t are defined as:

h_t: the number of layers in time slot $t, t = 1, ..., T$;

z_t: indication of a step in time slot t, $z_t \in \{0, 1\}, t = 1, ..., T$.

Usually, a lower spectrum means less quality fluctuations, resulting in a higher QoE. The benefit of this metric is that it reflects the subjective quality observed by viewers, and can be easily calculated.

2.3 Cloud Services

When serving millions of users, hosting a large scale of video repository becomes very expensive, especially due to the large upfront investments in hardware and time spend on the heavy lifting of managing the hardware. Cloud platforms have become one of the pri-

mary enablers to reduce the huge cost of video hosting. With the support of cloud services, content providers can rent the hardware and pay for the actual usage time and amount of data transfer, instead of purchasing and maintaining dedicated servers. In addition, a variety of hardware configurations are available varying from basic, low performance VMs to high-end bare metal servers depending on the actual needs of users.

In this dissertation, we look into many of the popular cloud service providers including both commercial clouds and research clouds. Commercial clouds are designed on pay-as-you-use model, with the goal of providing reliable, flexible and low cost IT resources. Users can get access to servers, storage, databases and a broad set of application services over the Internet. In this dissertation, we make use of two commercial cloud services: AWS Elastic Compute Cloud (EC2) [2] and Rackspace Cloud [13]. Both EC2 and Rackspace provide resizable cloud computing capacity to execute applications on demand. The cost of commercial cloud services depends on the type of resources used and the duration of the usage, as well as additional factors such as the amount of I/O performed and the amount of storage used. In addition to regular cloud server hosting, the commercial clouds also offers services including cloud storage, Content Delivery Networks (CDN) and cloud-based website hosting.

In contrast, research clouds provide free resources for the research community, on a reservation basis. In this dissertation, we make use of two research cloud platforms: ExoGENI [3] and CloudLab [80]. ExoGENI is an open-source research cloud platform, which allows users to programmatically manage a controllable, shared substrate on a reservation basis. The ExoGENI cloud uses a slice-based architecture which gives experimenters more flexibility than commercial clouds, since it allows them to create customized network topologies for a compute cluster.

CloudLab is a scientific instrument, open-source cloud platform. CloudLab allows users to instantiate a complete private cloud stack and to get full visibility to each aspect

of the cloud facility. In addition, CloudLab allows the users to get access to its IPMI based power monitoring and controlling modules for real time power consumption measurements.

2.4 Cloud Power Management Mechanisms

Energy consumption is one of the key aspects that constrains the scaling up of large cloud platforms. In order to optimize the energy usage and the performance of cloud applications, we make use of various techniques to monitor and control power consumption for either individual servers or at the scale of server racks.

Power capping mechanisms. Power capping enables the system administrators to cap the power consumption of a server or a group of servers. The corresponding power consumption can therefore be adjusted with the power capping configurations. In this dissertation, we cap the active power in a prototype by controlling CPU power states, e.g., via Dynamic Voltage and Frequency scaling and changing C-states, and capping utilization.

IPMI. In this dissertation, we make use of the Intelligent Platform Management Interface (IPMI) to control and monitor the power usage of the nodes in real time. IPMI is an autonomous computer subsystem that is separate from the operating system (OS). It allows the users to remotely boot or shut down severs or maintain the system in case of OS failures. IPMI is also able to monitor the system status, such as system temperature, power usage, and fan speed. Each CloudLab server is equipped with an out-of-band server management card that supports the IPMI protocol, which supports a fine-grained power monitoring at a 1Hz resolution at 1W granularity.

CHAPTER 3

RELATED WORK

There has been extensive research works on improving the performance and energy efficiency for both ABR streaming and cloud applications. In this chapter, we discuss the related work in these areas.

3.1 ABR Streaming Services

Due to its popularity, ABR streaming has been extensively studied in recent publications, especially for DASH. The related work for ABR streaming can be coarsely divided into two categories, i.e., (i) measurement studies showing real-world streaming behavior of ABR streaming, and (ii) studies that propose new quality adaptation mechanisms to improve the DASH streaming performance.

3.1.1 Measurements on ABR Streaming

Large-scale studies of DASH performance have been conducted in commercial video streaming platforms such as Hulu, Netflix and Vudu [21,49,50]. Similar measurement also studied aspects of the transmission behavior of video clients [20], DASH network traffic characteristics [78], and DASH QoE [41,74]. These measurement studies point out some of the real-world difficulties that face ABR streaming such as the high variability of the end-to-end throughput, the inaccuracy of client based rate estimation, as well as rebuffering risks due to non-preemptive segment download. These factors make a purely rate-based quality adaptation approach highly volatile.

3.1.2 Rate Selection Policies

As the DASH standard does not specify how to perform rate adaptation, many research groups studied the impact of different rate adaptation algorithms on DASH performance. The existing body of work on DASH quality adaptation has two sources of information, i.e., the buffer fill and the available bandwidth estimate. Proposed algorithms usually use one of these two sources as a main information source and the other one to cover corner cases. The work in [50] uses measurements from a commercial video streaming platform to assert that it is sufficient to mainly use the client buffer fill level to determine the quality of the next segment to download. Here, the authors find that it is important to obtain available bandwidth estimates only during the startup phase.

Villa et al. [94] propose a traffic shaping mechanism to alter packet interarrival times to improve the accuracy of available bandwidth estimates by reducing traffic burstiness. Tian et al. [90] proposed a PID controller for rate adaptation which takes the buffer filling into account to refine the available bandwidth estimates. Similarly, the authors of [36] propose a control theoretic approach to stabilize the buffer filling at certain level. In addition, a recent work [26] proposed a light weight rate adaptation algorithm based on a buffer map that avoids heavy online computation and provides playback with less rebufferring events.

Studies that investigate the impact of different segments sizes and lengths include [31, 54, 96]. In Sect. 5.5, we provided a brief sketch of the SARA algorithm from [54], which takes into account that segment sizes may differ widely even within the same quality level.

Another research direction is on studying and optimizing DASH performance under various network architectures. E.g., DASH over CDN [19, 68], DASH over CCN [30, 62], centralized control plane [42], etc. In addition, there are many DASH algorithms that take QoE into account when performing rate adaption. The first approach, that is denoted PANDA (for probe-and-adapt) [65, 66], presents a buffer filling based adaptation algorithm that solves the quality selection optimization problem with respect to an α-fairness objec-

tive using a dynamic programming approach. The authors use peak signal-to-noise ratio (PSNR) to capture QoE.

The second approach is given in [104], where the authors propose a QoE metric that is a weighted combination of the average video quality, the average quality variation, the rebuffering time, and the startup delay. The authors formulate the rate selection problem as a stochastic optimal control problem. Assuming stability of network conditions on *known* finite time horizons, the proposed algorithm uses a model predictive control approach to optimize the QoE metrics. This algorithm is based on an offline section, which does offline optimization (using CPLEX) for different scenarios, and an online section, which comprises table lookups of precalculated solutions. A similar approach that is solely based on the buffer filling is given in [88] which formulates this rate selection problem as a slot based deterministic optimization problem that is solved using a Lyapunov technique. Here, BOLA-U maximizes a weighted combination of the quality bitrate and the smoothness measured in terms of average rebuffering time. A second variant of this algorithm, BOLA-O, addresses the trade off between maximizing the quality bitrate and reducing oscillations during playback. Unlike BOLA-U, BOLA-O measures the download rate of previous streamed segments to choose a more sustainable quality bitrate for the client player.

A similar approach has been leveraged in [101] which formulates the rate selection problem as a MDP and uses dynamic programming to find an optimal solution. For these methods to find the optimal rate selection policy, strong assumptions have to be made on the statistics and predictability of the network conditions.

3.2 Energy-Aware Cloud-based Transcoding

Energy-aware cloud applications have become a widely discussed topic in recent publications. This is largely due to the increasing energy usage of IT infrastructure and increasing weight of video streaming services. However, little prior research has been conducted on transcoding with renewable energy powered data centers.

Initial research focuses on transcoding job scheduling aiming to shift transcoding jobs from the centralized media server to the local offloading server [34], or using DVFS management scheme to reduce power consumption and thread management and scheduling schemes [87]. These works look into cloud based transcoding scenarios, however, they are not taking renewable energy resources into account.

Another research direction is on optimizing transcoding job scheduling to maximize the Quality of Experience (QoS). For example, Gao et al. [43] look into predictive resource provisioning that optimizes QoS, while Wei et al. [98] investigate a task scheduling algorithm to dynamically adjust the resource reservation and schedule the tasks to improve the QoS. These works differ from ours because they do not consider energy consumption in cloud environments.

In this dissertation, our work takes into account both transcoding performance and renewable energy utilization. To the best of our knowledge, this is the first attempt to conduct large scale transcoding on clusters powered by both variable renewable energy and non-renewable energy prices.

3.3 Energy-Aware Parallel Cloud Application

Recently, researchers have recognized the importance of optimizing the usage of variable power. This is due, in part, to a combination of rising energy prices and falling prices for solar panels and wind turbines. In particular, data centers are beginning to make use of substantial renewable deployments, as evidenced by the 40MW co-located solar farm that powers Apple's new iCloud data center in Maiden, North Carolina [24].

Initial research on optimization for power variations has focused on either using energy storage to offset power shortages [46], or enabling isolated system components to adapt their power usage. Research on using energy storage focuses on the best combination of energy storage devices, e.g., batteries, flywheels, etc., to minimize costs, and evaluates the potential savings based on realistic workloads, power prices, and battery costs. Our work

differs from this work by focusing on optimizing parallel applications given a variable power source and a small fixed amount of energy storage capacity.

Another research direction is on adapting different system components to run on variable power. These system components include web servers [58], distributed caches [83], file systems [84], virtual machine migrations [61], and batch schedulers [45]. Our work focuses on parallel applications, rather than individual components. Prior work on batch schedulers is most closely related to our work. However, this work differs from ours in its focus on short batch tasks, which a scheduler may simply defer until enough power is available to run them. Unfortunately, for long-running, parallel tasks there may never be enough power to run them to completion, which requires them to, instead, dynamically adapt their execution in real time as power varies.

Another interesting approach is to run parallel tasks in virtual machines (VMs) and migrate them before deactivating nodes to consolidate workload. However, VMs introduce additional virtualization overheads that degrade performance, and the migration overhead is often large, especially for HPC applications that have large memory footprints. In addition, consolidating applications on a small subset of nodes may overload nodes and degrade performance, i.e., by causing memory thrashing.

The slack-based energy gear optimization leverages inter-node bottlenecks in MPI programs to improve energy-efficiency [55]. Our work differs from this approach by considering power variations from green energy sources, as well as both active and inactive power capping techniques.

Our approach in this dissertation aims to maximize energy efficiency by deciding the optimal number of nodes and capacity of each node based on the availability of renewable energy resources. The energy cost can be significantly reduced by using minimal amount of "dirty" energy resources and battery for energy storage. By sacrificing the least amount of performance, the maximum efficiency of renewable energy can be achieved.

CHAPTER 4

USER EXPERIENCE CHARACTERIZATION FOR WIRELESS ABR STREAMING

4.1 Introduction

Over the past few years, mobile video has become an essential Internet service. The increasing ubiquity of smartphones, tablets, and other mobile devices, together with the growth of media-intensive Internet applications that drive video usage, ensures that this trend will continue. According to [35], More than half a billion mobile devices and connections were added in 2015. If historical trends continue, 75% of the world's mobile data traffic will be video by 2020.

In reaction to this trend, various industry bodies have been working to standardize the delivery of mobile video. In particular, DASH has been specifically designed to meet this demand and enable a high-quality experience for end users of video on demand and real time communication services. It is designed to adapt to dynamic link characteristics, aiming to deliver the best possible video quality in any scenario.

Meanwhile, as demand for instant access to high-quality multimedia content grows, wireless carriers are racing to deploy upgraded networks that are better equipped to meet this demand. Serving wireless video is a significant challenge for already-stressed cellular data networks [39]. In addition to the high bandwidth requirements, video traffic imposes additional latency and packet loss constraints for acceptable service.

Yet despite widespread acknowledgement of the challenges associated with mobile video delivery and attempts to take these into account when developing new Internet video standards, surprisingly little is known about how these challenges affect consumers of mo-

bile video. Although a great deal of effort has been concentrated on quantifying the aggregate effect of mobile video on core and access networks, there has been no study of how mobile video services are experienced by individual users in a pedestrian mobile setting.

In this chapter, we collect mobile measurements from DASH by moving at walking speeds through an 802.16e WiMAX [23] network. Using measurements from the application, network, and wireless physical layers, we identify characteristics of the cellular data network that directly impact the quality of video service, and suggest areas for further improvements and optimizations.

Our contributions for this part of dissertation are, therefore, the following:

- Empirical measurements quantifying key characteristics of network and video performance experienced by a pedestrian user in a realistic mobile setting. We correlate measurements of wireless link quality to video bitrate, video frame rate, video frame size, packet loss, and other relevant statistics, for DASH services.

- Identification of behaviors in DASH when interacting with the wireless environment contributes to poor user experience. By noting these behaviors, we hope to distinguish areas for further improvement in these evolving Internet video standards.

4.2 Measurement Platform

This work was conducted using dedicated experimental WiMAX 802.16e networks installed at the campuses of the Polytechnic Institute of NYU (NYU-Poly) and the University of Massachusetts Amherst (UMass Amherst). Each installation includes a commercial WiMAX base station (BS) operating in a licensed frequency band, as well as other components required to route traffic from WiMAX clients to the Internet or other networks. For highly controlled experimentation, this platform has a distinct advantage over commercial cellular networks because it allows us to isolate the effects of the wireless channel quality from other variables such as competing traffic, carrier routing and shaping policies, and ra-

dio configuration. This increases consistency and repeatability, while still being more true to life than a simulation or emulation environment. In this section, we introduce the setup of WiMAX testbeds, and the tools for conducting the measurements.

4.2.1 UMass WiMAX Testbed

In order to systematically evaluate the user experience of DASH, we conduct part of our measurements on the UMass Amherst 802.16e mobile WiMAX research testbed in comparison to the WiMAX testbed at New York University with a similar configuration. The experiments are done using software tools from the WiMAX project [40] as well as the GIMI [9] instrumentation and measurement infrastructure.

The WiMAX technology offers wide-area broadband wireless communication similar to other 4G systems, with advanced mobility and flexibility, high data rates, and support for diversified QoS service classes. The bandwidth and range of WiMAX make it suitable for a number of potential applications, including providing mobile broadband coverage to a variety of devices as a Metropolitan Area Network (MAN), providing an alternative to cable for last mile broadband access (especially to rural areas where it is not practical to install a wired connection), and serving as a wireless backhaul for cellular networks.

The UMass WiMAX testbed is part of the deployment of the mesoscale research testbed of Global Environment for Network Innovations (GENI) [27,28]. This testbed provides network researchers with 4G cellular services and wide-area coverage with fully programmable infrastructure. Both the WiMAX base station and clients are virtualized such that users can run multiple experiments concurrently. The testbeds are managed by the cOntrol and Management Framework (OMF) [77], which gives local and remote researchers a robust set of scripting, experiment control, management, and measurement tools to run their experiments. The testbed nodes may be "fixed" nodes, which remain connected to a wired control network at all times, or "mobile" nodes, which may be temporarily disconnected from the control network for mobile experiments.

Access Mode	SOFDMA/TDD
Center Frequency	2.595 GHz
Channel Bandwidth	10 MHz
Frame Duration	5 ms
Antenna Beamwidth	120°
Transmit Power	38 dBm
TDD DL:UL ratio	35:12
CRC	Enabled
Packing	Enabled
Fragmentation	Enabled
Compressed MAP	Disabled
Service Class	Best Effort (BE)

Table 4.1: Selected parameters of WiMAX BS configuration.

The GENI WiMAX BSs used in this study are commercial WiMAX 802.16e radios from NEC, operated via customized control software developed at WINLAB [29] that is installed on a separate BS controller. Table 4.1 summarizes key BS configuration parameters for the experiments presented in this chapter.

4.2.2 Testbed Configuration

The network configuration for the WiMAX testbeds is shown in Fig. 4.1. Each testbed node is equipped with an Intel Centrino Advanced-N+WiMAX 6250 wireless network adapter which is configured to connect to a GENI-operated WiMAX network. When a testbed node requests entry to the WiMAX network, Generic Routing Encapsulation (GRE) [47] tunnels are set up on the BS and the BS controller to route traffic to and from the WiMAX client. Meanwhile, a Click software router [56] running on the BS controller is configured to forward traffic between the WiMAX network and a user-defined datapath. In this way, WiMAX clients can communicate with other hosts on a campus network, the public Internet, or a GENI backbone network.

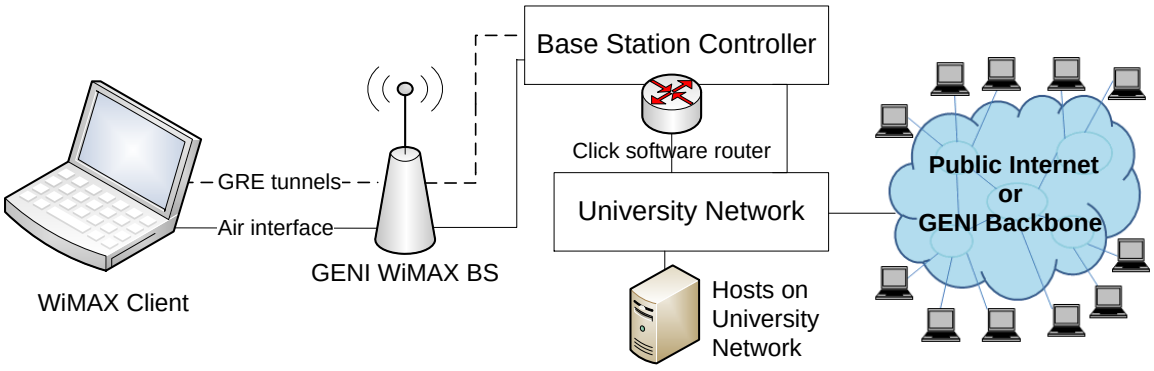


Figure 4.1: The WiMAX testbed network configuration. A software router configured by the BS controller forwards traffic from each client to its predefined datapath on the university network, the public Internet, or a GENI backbone link.

4.2.3 Measurement Infrastructure

This work was made possible by the extended measurement capabilities offered by several open-source tools. The GIMI [9] instrumentation and measurement toolset provided storage and presentation of measurement data, which was collected by specially instrumented versions of popular Internet applications.

The GIMI project aims to provide instrumentation and measurement services for experimenters on selected types of GENI aggregates. GIMI builds on OMF and its associated measurement library, OML [70], which are already used on GENI WiMAX testbeds. The GIMI toolset adds utilities for pushing measurements from a WiMAX testbed to a user's personal storage on an iRODS [7] data grid, and for visualization and presentation of measurement results.

The WiMAX clients used in this work were equipped with GIMI tools, which we used to push measurements collected by an OML server on the clients to the GIMI measurement data archive (i.e., iRODS).

In a previous study [40], we quantified the performance of this platform, particularly with regard to achievable data rates. We found that the data rates achieved on this platform are similar to typical data rates measured by others for users of commercial HSPA+ networks and users of commercial LTE networks [48]. This suggests that with regard

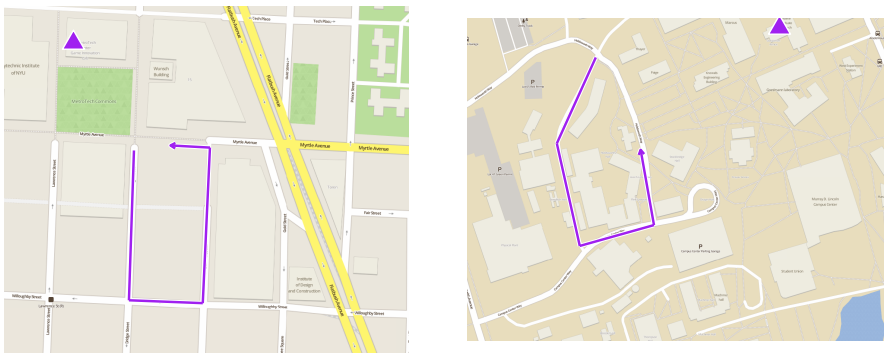
to overall performance, this platform can be considered broadly representative of current wireless broadband networks.

The clients used in this study were standard laptops equipped with a commercial WiMAX network adapter, a USB GPS dongle, and a webcam. On these, we installed modified version of 2.1.0 of the popular VLC media player, which includes a DASH plugin [72]. This software was instrumented using the OML measurement framework [70] to include "hooks" that collect key network and video metrics from the application, and stream measurements to a local database at regular intervals while the application is running. We also used logger applications to continuously collect GPS coordinates from the *gpsd* daemon in Linux and wireless link characteristics from the WiMAX device. We collect the following data from these tools:

- **VLC with DASH.** We collect DASH performance data including DASH download rate, DASH chosen bitrate and buffer status.
- **WiMAX Logger.** We collect wireless signal data includingReceived signal strength indicator (RSSI), carrier to interference plus noise ratio (CINR) and frequency.
- **GPS Logger.** We collect location data including latitude, longitude, altitude and moving speed.

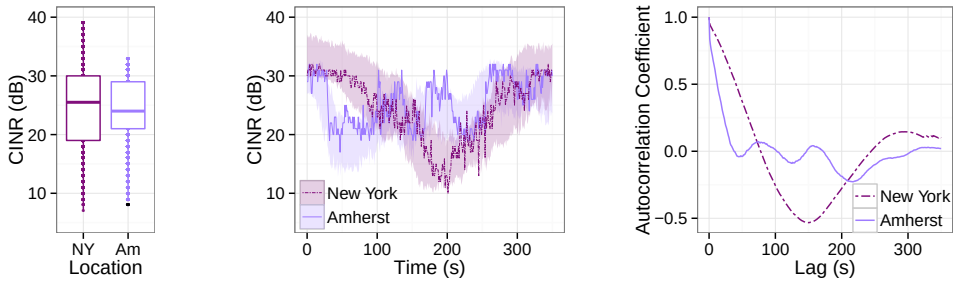
4.3 Wireless Environment

Because the characteristics of a wireless channel are heavily dependent on the topography of an area, we collected measurements at two campuses in very different wireless settings. NYU-Poly is in an urban area composed mainly of high-rise commercial, civic, and residential buildings. It is a highly dynamic environment, with heavy moving vehicles and pedestrian traffic in the radio path. In contrast, the UMass Amherst campus is a suburban/semi-rural environment, with few tall buildings and little traffic.



(a) (b)

Figure 4.2: The mobility patterns followed in New York and Amherst, respectively, and the location of each WiMAX BS. *Map data ©OpenStreetMap contributors, tiles from skobbler.*



(a) (b) (c)

Figure 4.3: WiMAX link characteristics. Figure 4.3a shows the distribution of CINR (signal quality) measurements at each location, Figure 4.3b shows CINR along each experiment path as a function of time for a single representative trial, with the shaded area indicating a range of one standard deviation above and below the mean CINR calculated across all trials. Figure 4.3c gives the autocorrelation of CINR over a long timescale.

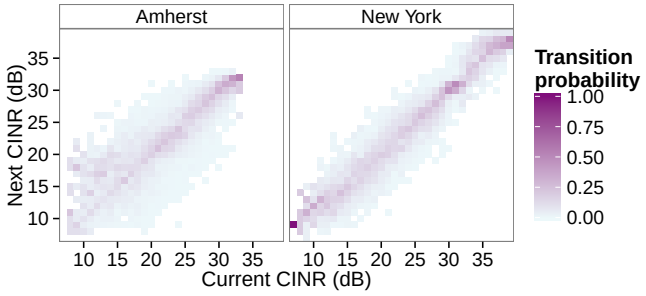


Figure 4.4: Empirical CINR transitions over a timescale of one second. All measurements are collected on a mobile client moving at walking speeds.

All of the measurements described in this chapter were collected along a 400 m measurement path at each location. The geographical layout of this path is depicted in Figure 4.2, and its wireless link characteristics are given in Figure 4.3.

Figure 4.3a shows the distribution of CINR values measured over all experiments, while Figure 4.3b shows CINR as a function of time along the experiment path. At both locations, a similar mean carrier to interference plus noise ratio (CINR) of approximately 25 dB is observed along the measurement path, although a slightly greater range of CINR values is observed in New York.

However, the channel behavior in time is quite different for the two wireless settings. Because the wireless signal propagates through "street canyons" in the urban area, the wireless signal tends to be very consistent when moving within a single "canyon." In the suburban environment, where variations in signal strength are attributed to shadowing from individual buildings and obstructions in the signal path, more dramatic variations are observed over a short timescale.

This property is supported by Figure 4.3c, which shows the autocorrelation of each channel for lags up to 350 seconds. For the Amherst channel, the autocorrelation function closely resembles the widely reported exponential model. This intuitively represents the idea that locations that are close together are highly correlated, with the correlation decreasing gradually with distance. In New York, the autocorrelation function is shaped

more like an exponential decaying sinusoid [106]. For the urban wireless channel, there is a strong correlation between measurements on a single block in the street grid, even though these may be separated by some tens of seconds, and virtually no correlation across intersections. This is reflected in the shape of the autocorrelation function, which shows a strong correlation between samples collected on the same block, and a sharp change in correlation coefficient at each intersection.

The behavior of the wireless channel over a fine timescale, meanwhile, shows that over a short interval, the wireless channel is more consistent in the urban environment. Figure 4.4 shows the empirical probability of transitioning from one CINR value to another over a timescale of one second. In New York, two CINR values observed one second apart almost always differ by less than 5 dB, while in Amherst, transitions greater than 10 dB are seen quite often.

Put simply, we may state based on Figure 4.3b, Figure 4.3c, and Figure 4.4, that a period of especially poor or especially good signal quality is likely to be of short duration in Amherst, and of long duration in New York.

We note these differences because the video applications under consideration use short-term estimates of network metrics to adjust bandwidth usage; the time variance of the channel is therefore highly relevant to performance. Although these application-level decision are generally designed to apply only for a short timescale, in practice we will see that the behavior of the channel over a longer timescale is also highly relevant.

4.4 Experimental Evaluation

Our procedure for measuring the user experience of DASH is as follows. The DASH-enabled VLC player [72] is installed on a laptop, which connects to the campus network through a WiMAX access network. We use an Apache HTTP server to hosts all the video segments and the media presentation description (MPD) file. The link between this video server and the WiMAX network is a wired connection under low load, so that any network-

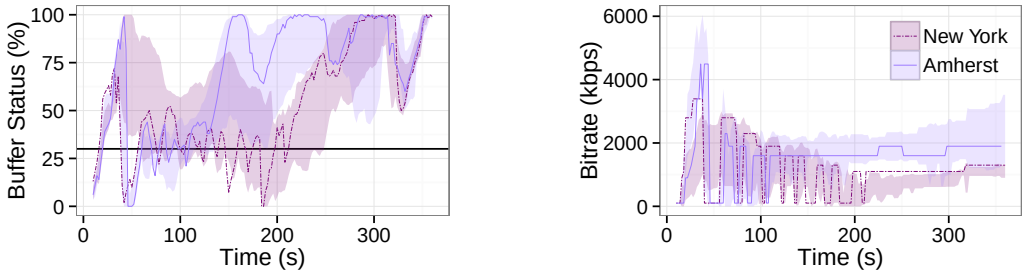
related behaviors we observe may be attributed to the wireless link. We stream DASH video to the client while moving at walking speeds along the experiment path described in Section 4.3. The experiment is repeated ten times at each location, with consistent results observed across the ten trials.

For the video, we used the Big Buck Bunny animated video from the DASH dataset [63], with 2-second segments encoded at bitrates of 100, 200, 350, 500, 700, 900, 1100, 1300, 1600, 1900, 2300, 2800, 3400, 4500, and 6000 kbps. Because the VLC media player cannot currently play back H264/MP4 videos with dynamic resolution, we used video encoded at a constant resolution of 480p and a 24 fps frame rate. Our choice of 2-second segments was intended to provide high flexibility for adapting to bandwidth fluctuations in the mobile setting. Similarly, we used a maximum buffer size of 30 seconds (the default in VLC), to help the client smooth over temporary disruptions in signal quality.

The adaptation policy used by a DASH client is not standardized. The DASH client in VLC follows a simple rate-based adaptation scheme which uses buffer state and bandwidth history to decide what bitrate to request for the next video segment. We call this a *maximum bitrate-low buffer avoidance* policy; a similar policy has been used e.g. in [64].

- If the video buffer is full, the client does nothing.
- If the video buffer is less than 30% full, the client requests the next video segment at the lowest bitrate level, to avoid buffer depletion and a freeze in video playout.
- Otherwise, the client requests the next segment at the highest bitrate that is less than the empirical bitrate measured when downloading the previous segment.

This simple policy is suitable for this study because our intent is not to evaluate a particular adaptation policy, but rather to gain a broad understanding of DASH performance in a particular setting.



(a) (b)

Figure 4.5: Download rate for video segments over 375 seconds of playback time, and buffer status over the same period. The line gives values for a representative trial, while the shaded region shows one standard deviation above and below the mean for all trials. In Figure 4.5a, the 30% point buffer state is marked by a horizontal line.

4.4.1 Buffer Status

Figure 4.5a shows the buffer status of the DASH client over 375 seconds of video playback. A maximum of 30 seconds of video may be buffered at one time. Thus, at 100% buffer status, 30 seconds of video are in the buffer; at 0%, freezes in video playback will occur. The 30% point, at which the client drops to the lowest available bitrate to avoid buffer starvation, is marked by a horizontal line in Figure 4.5a.

The shape of the buffer status curve is closely related to the CINR curve in Figure 4.3b. When the wireless channel quality is poor, there tends to be a corresponding dip in the buffer status. At Amherst, after an initial period of poor channel state, the signal recovers and the DASH client builds up a sufficiently large buffer to survive subsequent periods of poor channel quality without buffer starvation. In New York, where periods of poor channel quality tend to last for an extended period of time, the DASH client cannot survive the low CINR periods without dropping below the 30% state.

4.4.2 Video Bitrate

The buffer status directly impacts the video segment bitrate selected by the DASH client, shown in Figure 4.5b. The mean bitrate over 375 seconds of video playback was

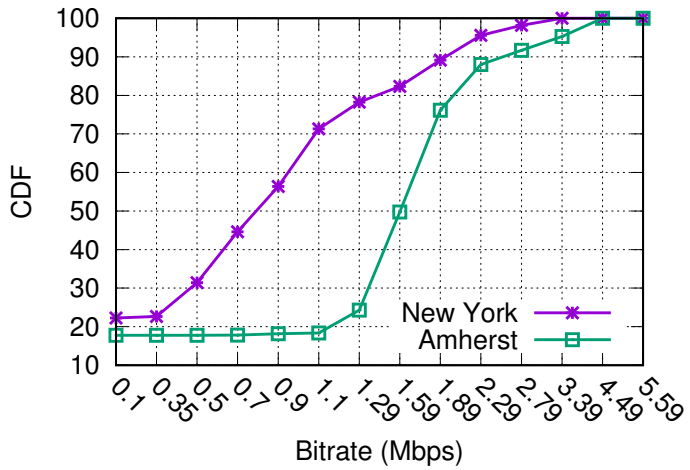


Figure 4.6: Distribution of bitrates for video segments downloaded by the DASH client at New York and Amherst.

1003 kbps in New York and 1744 kbps in Amherst, with standard deviations of 416.1 and 715.0 kbps, respectively.

Figure 4.6 gives the relative frequency of each bitrate selected by the DASH client. At both locations, we observe a high proportion of instances where the lowest bitrate is used, due to the adaptation strategy of dropping to the lowest bitrate whenever the buffer is approaching starvation. This behavior has the unfortunate effect of introducing sudden changes in video quality, rather than gradual changes which may be preferable to most users [71]. This effect is especially pronounced in New York, where we see many instances of fallback to the lowest bitrate and then recovery to a higher bitrate (Figure 4.5b). These coincide with the areas in Figure 4.5a where buffer status drops below 30%.

We also see evidence of this behavior in Figure 4.7, which shows the empirical probability that the next segment is downloaded at a particular bitrate, given the current segment's bitrate. The transitions are generally smooth (i.e., the bitrate of the next segment is similar to the current segment's bitrate). However, there is always a substantial probability of dropping to the lowest bitrate to avoid buffer depletion (and subsequently, abruptly recovering to a high bitrate).

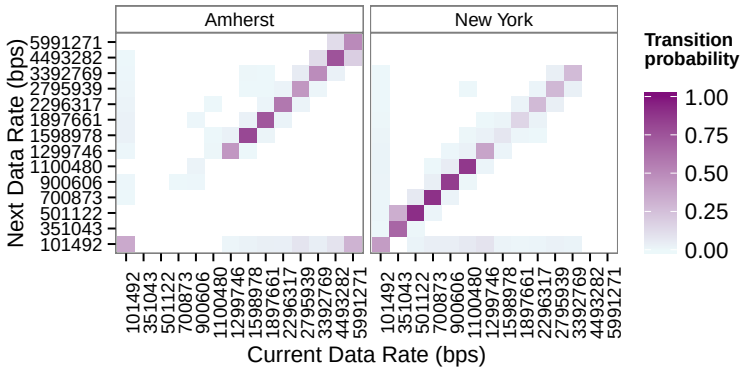


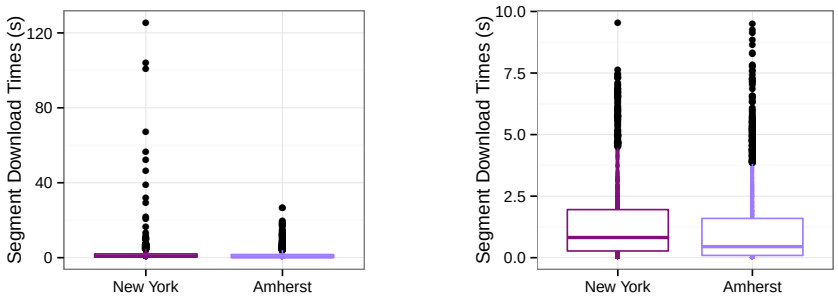
Figure 4.7: State transition probabilities for video download bitrate.

4.4.3 Segment Download Time

For a DASH video client to avoid freezes, the average download time per segment should be less than the segment playing time. Figure 4.8 shows the distribution of segment download times for our measurements with 2-second segments, with the mean download time given as the horizontal line in the boxplots, the upper and lower hinges corresponding to the first and third quartiles, and outliers beyond 1.5 interquartile range (IQR) of the hinges plotted as points. Overall, we measured a mean download time of 1.56 seconds per segment, with 75% of segments downloaded in less than 1.87 seconds. However, the outliers - though rare - include segment download times as high as 125 seconds, especially in the urban setting. Because the client uses sequential HTTP downloading, long download times block the downloading of future segments, which often causes playback to freeze (depending on buffer status).

4.4.4 Buffer Status

It is also noteworthy that as the segment download time increases, the bandwidth measured over the duration of the download is less predictive of the bandwidth for the next download period. The extent of this relationship is governed by the autocorrelation of the wireless channel, shown in Figure 4.3c. When long segment download times exist (as observed in the urban wireless network), the simple rate-based adaptation logic is less ef-



(a) (b)

Figure 4.8: The distribution of segment download times is shown in Figure 4.8a. Figure 4.8b is a zoomed-in version of the same data, showing download times that are shorter than ten seconds.

fective. Long download times (those above the third quartile) tend to appear in clusters because of this effect.

4.4.5 Video Playback Freezes

A key metric for video streaming performance is the frequency and duration of freezes in video playback. Here, the behavior of the DASH client is very different depending on the wireless setting. In Amherst, on average, playback was frozen for 7% of the 375 second playback period, during which time 8.1 short interruptions (lasting less than four seconds) and 0.5 long interruptions (with duration 4-6 seconds) were observed. In New York, the video was stalled for 15% of the same playback period on average. During this interval, an average of 8.1 interruptions less than four seconds and 2.3 interruptions lasting four seconds or more were observed, with some interruptions lasting up to 68 seconds.

The extended freezes may be attributed to the occasional long segment download times measured in New York (Figure 4.8). These results are also corroborated by Figure 4.5a, which shows the video buffer status while walking along the experiment path. In New York, we see an extended period of time during which the buffer is almost empty, while at Amherst, the buffer is depleted only for short intervals before rebounding.

4.5 Summary

In this chapter, we evaluate the DASH performance over a wireless scenario. Results show that the performance of an adaptive video delivery strategy in a mobile setting depends on the characteristics of the wireless network in time, in both the short- and long-term. For example, the maximum bitrate-low buffer avoidance policy used by the VLC DASH client has good performance in the suburban location, but not in the urban setting, where signal quality is generally constant for the length of at least one "block" in the street grid. For this setting, a strategy that adjusts download rate to maintain a constant buffer status, such as the policy suggested in [90], might be more appropriate. (This would also offer a smoother playback experience.) We expect that in a very rural area where distance-based path loss is most significant in determining wireless signal quality, or a crowded network where load on the BS dictates data rates, entirely different policies might be optimal.

We also note that the DASH client sometimes experiences freezes in video playback as a result of extremely long segment download times in areas with poor signal quality. Segment abandonment mechanisms, e.g., the implementation in BOLA adaptation algorithm [88], and parallel segment downloading mechanisms, could alleviate this issue.

CHAPTER 5

QOE AWARE QUALITY ADAPTATION FOR ABR STREAMING SERVICES

5.1 Introduction

In Chapter 4, we show that the DASH user playback experience can be significantly compromised under unstable network conditions. In this chapter, we design and evaluate a set of DASH quality adaptation algorithms to improve the QoE and to ensure a smooth DASH playback experience.

Today's DASH implementations are mostly based on HTTP and the benefits of the underlying TCP protocol, which include standardized transport, firewall penetration, and adaptation to bandwidth changes. Unfortunately, the use of TCP also brings a set of difficulties which we identify and investigate in this chapter. Examples of such challenges are, the dual control-loop (one for DASH and one for TCP as already identified by Huang et al. [49]), the impact of video segment size, and the impact of dead times on congestion window size.

An additional challenge for DASH video streaming is to provide a high QoE to the viewer. For example, results in Chapter 4 have shown significant quality fluctuation when DASH experiences unstable network condition. From the onset, DASH has been designed with the goal to prevent re-buffering events, which have the most serious impact on QoE, as shown by Krishnan and Sitaraman [59]. In addition to re-buffering events, frequent changes in quality have also been identified as detrimental to QoE by viewers, e.g., in Zink et al. [107].

These insights motivated us to design a new DASH adaptation algorithm that aims at achieving the highest possible quality (in terms of bitrate) while minimizing the number of

quality changes, since we believe that most existing approaches only focus on the former and, thus, do not always result in a satisfying QoE. In this chapter, we study the effectiveness of transporting the data from the content providers to the customers and propose bitrate selection algorithms to improve the user Quality of Experience (QoE).

We therefore make the following contributions in this part of dissertation:

- **Critical Observations.** We perform a detailed analysis of the interplay between the DASH adaptation mechanism and underlying TCP. Through this analysis we identify several issues that contribute to the problem of optimizing the DASH streaming performance. Most notably, we identify *i)* the substantial impact of segment size on the download rate; *ii)* the impact of dead time on the congestion window; and *iii)* the inaccuracy of segment-based available bandwidth estimation.

- **SQUAD & SQUAD-BR/RR.** We develop a new DASH rate adaption algorithm denoted SQUAD, that has the goal to maximize the quality of experience of users watching video by addressing many of our critical observations. To achieve this goal, we consider two quantitative metrics that describe QoE and combine these in an online optimization algorithm. With SQUAD-BR and -RR, we present an extension of this algorithm that performs retransmissions for low-quality buffered but not played-out segments, to further improve QoE.

- **Evaluation.** We present results from an extensive evaluation of the SQUAD algorithm. We perform a series of experiments in a controlled environment (GENI testbed) and in the wild (public Internet) and compare SQUAD, SQUAD-BR, and SQUAD-RR with a large set of other existing DASH algorithms.

5.2 DASH Player Architecture

In this section, we highlight the main components of a general DASH player architecture as depicted in Fig. 5.1. We divide the client block into the following logical components: *(i) playout buffer*, *(ii) rate estimation / prediction* and *(iii) quality adaptation*.

In the following, we provide some initial definitions of DASH parameters to lay the ground for subsequent modeling. We denote the size of a DASH video segment (in bits) of a certain quality as $s_{i,q}$, with the segment number $i \in \{1, \ldots, N\}$ and the quality level $q \in \{1, \ldots, Q\}$. Here, $q = 1$ ($q = Q$) denotes the lowest (highest) quality with respect to the video quality bitrate r_q, i.e., $r_j < r_{j+1}$ for $j \in \{1, Q-1\}$. We denote the quality of a fetched segment i by q_i and its quality bitrate by r_{q_i} and drop the subscript when it is obvious. We consider the case where all video segments have equal length in time, i.e., every segment carries X seconds. The fetch time of segment number i in quality level q is given as $t_{i,q} = s_{i,q}/R_{i,q}$, where $R_{i,q}$ is the segment download rate in bit/sec. In Sect. 5.3 we describe how the segment download rate $R_{i,q}$ depends on a combination of different factors such as the network conditions, the TCP state and even the size of the segment $s_{i,q}$.

5.2.1 The Playback Buffer

The purpose of the playback buffer $B(i)$ is to smooth short-term variations of the network conditions. Specifically, *temporary* fluctuations in the segment download rates R should be absorbed by the buffer, i.e., producing fluctuations in the buffer filling $B(i)$ while keeping the steady state playback quality q unchanged.

In this work, we define $B(i)$ as the cumulative number of video seconds contained in the playout buffer after fetching segment i. Note that $B(i)$ is defined over $i \in \{0, \ldots, N\}$ and that by convention $B(0) = 0$. We define the event $\{B(i) = 0\}$ as the *rebuffering* event. For any practical application, the playout buffer is set to a finite size $B_{\max}$. For $B(i-1) < B_{\max}$ we write down the buffer recurrence as

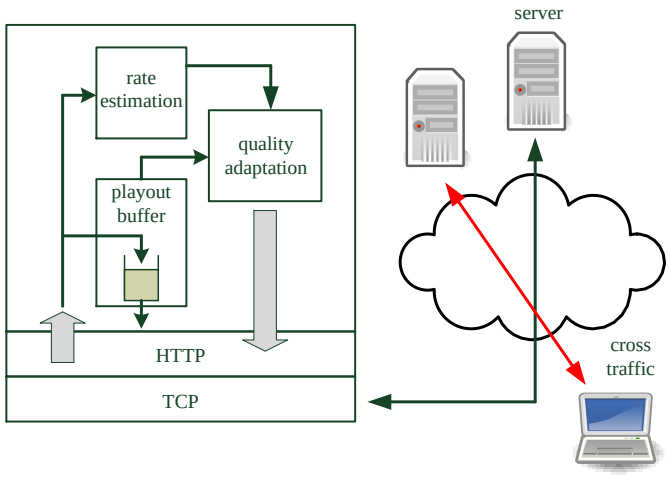


Figure 5.1: Coarse architecture of a DASH client. Buffer filling and download rate estimates are fed to the quality adaptation logic which decides the quality of the next segment. The stream encounters varying network conditions, e.g., due to contending cross traffic.

$$B(i) = \max\left\{0, B(i-1) + X - t_i\right\}, \tag{5.1}$$

where t_i is the fetch time of segment i. For $B(i) = B_{\max}$ we have $B(i+1) = B_{\max} - t_{i+1}$ since the player idles for X seconds when the buffer is full.

5.2.2 Rate Estimation

One basic client-side download rate estimation logic in DASH simply divides the segment size $s_{i,q}$ over the segment fetch time t_i. Here, the fetch time is given by the time difference between the timestamps of the HTTP `GET` request and the segment being delivered to the playout buffer, i.e.,

$$d_{i,q} = \frac{s_{i,q}}{t_i^{\texttt{delivered}} - t_i^{\texttt{GET}}} \tag{5.2}$$

Note that the rate estimate $d_{i,q}$ is smeared by the one-way delay of the `GET` request. This error diminishes with increasing segment size $s_{i,q}$. Rate estimation methods that calculate the download rate over multiple concatenated segments extend (5.2) to

$$d_{i,j,q} = \frac{\sum_{k=i}^{j} s_{k,q}}{t_j^{\texttt{delivered}} - t_i^{\texttt{GET}}} \tag{5.3}$$

for segment indexes $j \geq i$. In Sect. 5.3, we provide a critical evaluation of the foundations of the DASH rate estimates.

5.2.3 Quality Adaptation

Next, we review some basic concepts for the segment based quality adaptation logic shown in Fig. 5.1. DASH clients first fetch media presentation description (MPD) files that contain information of the contents to be streamed, e.g., server IPs, bitrates of different quality levels and the URIs to the segments of different qualities [86]. Basically, quality adaptation algorithms have two sources of information, i.e., the status of the playout buffer filling and the download rate of previous segments provided by the rate estimation logic. Buffer based quality adaptation takes, in general, the current buffer filling $B(i)$ and, in some cases, the change of the buffer filling $(B(j) - B(i)) \,/\, \left(t_j^{\texttt{delivered}} - t_i^{\texttt{delivered}}\right)$ for segments $j \geq i$, into account when deciding on the quality of the next segment to be fetched. The change of the buffer filling is an indicator for a mismatch of the segment download rate and the chosen rate. In DASH, there exists a subtle relationship between the change of buffer filling metric and the average download rate, since DASH introduces inter-request time gaps. We will provide a detailed analysis of this issue in Sect. 5.3.

Quality adaptation mechanisms also take the download rate estimation into account, aiming to match the segment download rate to the playout bitrate. This is done by choosing segment qualities with bitrates lower than the estimated available bandwidth. The estimation of the available bandwidth for the next segment(s) given the download rate history can rely, for example, on network active probing techniques [52] or simply on time series analysis. Furthermore, the rate based adaptation mechanism bears the risk of quality oscillation in accordance to the fluctuation of available bandwidth. From a streaming point of view, it is well known that frequent video quality oscillations are detrimental to the QoE perceived by the users.

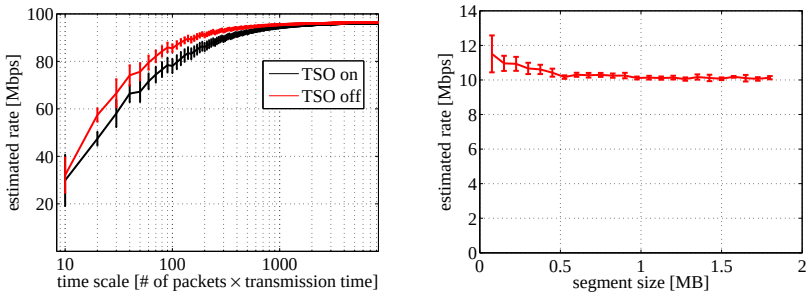
In the following sections, we will refer to the basic DASH player architecture and describe the details of the modifications introduced by our adaptation algorithm SQUAD. We will provide some critical observations on DASH that inspire some concepts of SQUAD.

5.3 Critical Observations for DASH

In this section, we make some fundamental observations in DASH that increases the difficulties of DASH performance optimization.

5.3.1 User Space Rate Estimation

DASH clients *usually* estimate the segment download rate from segment timestamps in user space. Depending on the hardware environment and its configuration, these estimations may vary significantly. Trivially, estimates made in virtualized environments may be highly varying, because of VM scheduling [100]. Therefore, in this part, we look at a non-virtualized bare-metal topology and show the impact of different settings of the network interface card (NIC), at the example of `segmentation offloading` (SO), on the DASH segment rate estimates. To this end, we use the Emulab testbed [99] to build a butterfly topology as depicted in Fig. 5.11 with $M = 1$ node on each side, where we use bare metal machines connected via 100 Mbps links. In this case we do not generate any cross traffic. We emulate a persistent HTTP DASH flow through a long running greedy TCP flow, and then compare the segment download rate estimates obtained for different segment lengths. Figure 5.2a shows the rate estimates over increasing estimation time scales. We normalize the x-axis, i.e., the time scale, to the number of packets that fit into one time slot at line rate. From Fig. 5.2a, we deduce that the user space rate estimate is indeed affected by the segmentation offloading for small DASH segments, i.e., in the order of tens to hundreds of packets. On the other hand, larger time scales, i.e., equivalent to larger DASH segments, allow some averaging such that the impact of SO can be negligible.



(a) Segmentation offloading. (b) Rate versus segment size.

Figure 5.2: User space rate estimations in DASH: (a) Impact of NIC segmentation offloading on rate estimation in user space. (b) User space rate estimation in DASH is more accurate for longer segments.

To get a better understanding of the DASH user space segment download rate estimation, we rerun the experiment with the Python DASH player from [54]. We modified the player to continuously measure the download rate *during* segment download process by taking one measurement point every 50 packets. A one-minute video is streamed in the topology in Fig. 5.11 with link capacities of 10Mbps, one DASH flow and no cross traffic. The video segment length is 2 sec. We measure the download rate for segments of different sizes as shown in Fig. 5.2b, which depicts averages and 0.95 confidence intervals. Here, too, we observe that the inaccuracy of rate estimates is highest when the segments are small.

5.3.2 Interaction with Underlying Protocols

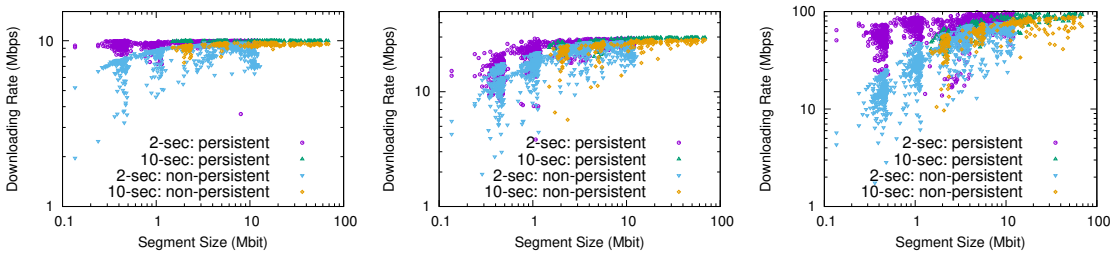
In the following, we discuss specific characteristics of DASH that arise due to TCP.

Dual control loop

The design choice of DASH to utilize HTTP for adaptive bitrate streaming brings numerous advantages, such as standardized transportation, firewall penetration, adaptation to the bandwidth changes and all the advantages of TCP. However, since the DASH player needs to specify the quality bitrate of the segments to be fetched, it may be regarded as

an outer control loop, while TCP running as an inner control loop that prevents congestion. While TCP aims for the fair share on a packet level time scale, DASH aims for the sustainable quality bitrate, i.e., essentially the fair share, on a segment level. The problem exacerbates when the control loop of DASH runs on a per segment basis. Since the segments are of different sizes, the time scale on which DASH tries to find the fair share is continuously changing. Figure 5.3 shows empirical segment download rates for different segment sizes from testbed measurements, with one DASH flow using the topology shown in Fig. 5.11 without cross traffic. We vary the segment lengths (in seconds), the link capacities, and the persistency of DASH HTTP connection. Figure 5.3 clearly shows the impact of the DASH segment size on the download rate for both HTTP connection types. The empirical download rate may be much lower for small sized segments.

Our algorithm (SQUAD), which we present in Sect. 5.4, takes into account this discrepancy of the rate estimation time scales and provides DASH with (available) download rate information on the appropriate time scale.



(a) 10 Mbps. (b) 30 Mbps. (c) 100 Mbps.

Figure 5.3: DASH segment download rates for links of different capacity. The segment size has a substantial impact.

DASH download rate vs. available bandwidth

In general, the segment download rate of the DASH client is derived from the HTTP activity, i.e., from (5.2). This download rate estimation does not necessarily coincide with the available bandwidth on the client-server path [67]. Basically, TCP aims for the fair

share to which a long-lived TCP flow would converge to. In contrast, as shown in Fig. 5.3, DASH cannot even utilize the full capacity of the pipe for small segment sizes.

DASH is TCP submissive

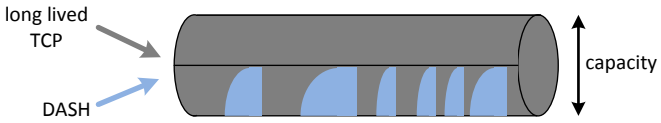
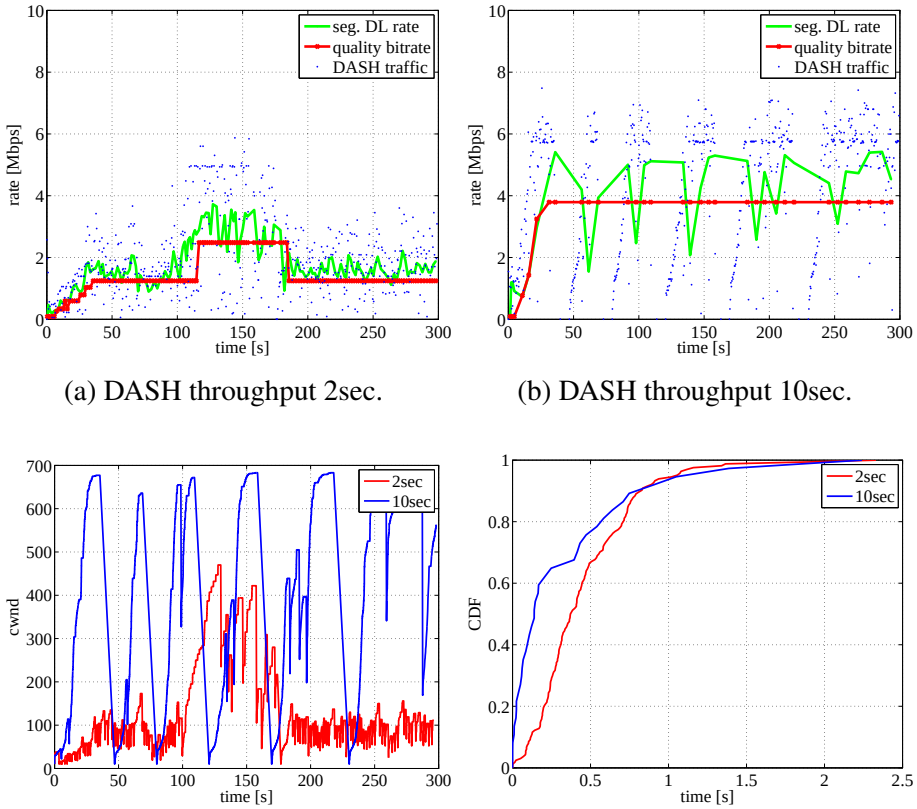


Figure 5.4: DASH abstraction as source of TCP mice flows.

Although DASH utilizes HTTP over TCP/IP to retrieve segments, we argue that it does not necessarily receive its fair bandwidth share when competing with other long-lived TCP cross traffic. The reason for this behavior is that a DASH video stream does not constitute a *long-lived* TCP flow from the server to the client. Figure 5.4 shows a sketch of this behavior for one hypothetical case of one DASH flow competing with one long-lived TCP flow.

Reasons for this discontinuous traffic behavior lie in the nature of DASH streaming. We assume a persistent HTTP connection, since in case of DASH over non-persistent HTTP, it is simple to show that the continuous TCP cross traffic receives more than the fair bandwidth share. In general, there exist dead times of no DASH transmission that result from the DASH adaptation algorithm itself, i.e., depending on how often the DASH adaptation logic fetches a new segment and the corresponding buffer filling. Figure 5.5d shows the empirical Cumulative Distribution Function (eCDF) of the dead times between receiving the last packet of one DASH segment and sending out the HTTP `GET` request for the next one. Clearly, there is silence time of multiple hundred milliseconds in the median case. This is sketched in Fig. 5.4 as gaps between the DASH segments. In addition, due to the discontinuous behavior of DASH, the congestion window of the corresponding TCP session has a higher risk of being reset due to idling more than long lived greedy TCP sessions.



(a) DASH throughput 2sec.

(b) DASH throughput 10sec.

(c) DASH-TCP cwnd.

(d) Silent times between segment reception and next segment request.

Figure 5.5: DASH is TCP submissive: impact of the segment length for one DASH flow competing with one TCP Reno flow.

In Fig. 5.5, we show the results for the experiment of competing DASH and a long-lived TCP flow as sketched in Fig. 5.4. We run this experiment with two different segment sizes, i.e., 2 sec and 10 sec, a link capacity of 10 Mbps and a DASH repository of the `Big Buck Bunny` dataset with quality bitrates ranging from 89 Kbps to 4.2 Mbps. We emphasize that we do not aim to investigate ABR segment size optimizations in this work, but merely show that different settings of DASH parameters can lead to entirely different performance. Our goal is to show the fundamental roots for this varying behavior. Figure 5.5a shows the segment throughput as measured by the client as well as a finer grained throughput measurement that is done on the wire. For a segment size of 2 sec, the quality bitrate of the DASH stream is remarkably low with respect to the fair share. The reason for that can

be inferred from Fig. 5.5c, where we observe that the corresponding congestions window (measured by number of TCP packets) does not ramp up as the segments are "small" in size. In contrast, for 10 sec segments the congestion window in Fig. 5.5c is nearly an order of magnitude higher and the quality bitrate shown in Fig. 5.5b stays at a high quality level with bitrate of 3.8 Mbps.

The interrupted TCP stream generated by DASH can be modeled as the output of an ON-OFF source that generates TCP mice flows (although technically imprecise because of the persistent HTTP connection). Hence, it is known that competing long-lived (elephant) and mice flows suffer from fairness disparities [44].

5.4 Spectrum-based Quality Adaptation

In this section, we describe a DASH quality adaptation algorithm that addresses many of the critical points raised in Sect. 5.3. We list all symbols and their definitions in 5.1. The aim of this algorithm is to maximize the quality of experience associated with DASH streaming sessions in a quantitative manner. We consider two metrics that describe QoE, and connect these metrics to an online optimization algorithm in a novel manner. Given a DASH streaming session of N segments each of length X seconds, the first metric we consider is the average video bitrate, i.e., the average video quality, which we express as

$$\bar{r} = \frac{1}{N} \sum_{i=1}^{N} r_{q_i}. \tag{5.4}$$

Recall that r_{q_i} is the quality bitrate of segment i at quality level $q \in \{1, \ldots, Q\}$. Equation (5.4) does not consider the detrimental impact of rebuffering on the average video quality. One method to capture rebuffering in (5.4) is to substitute N by N', i.e., the number of segments N in addition to the number of segments that can be fitted in the rebuffering time as $\lceil t_{rebuf}/X \rceil$. Trivially, the corresponding rates r_{q_i} are set to zero.

The second metric that we consider is a centralized measure for the variation of the video quality around the average quality which is denoted as "spectrum" in [107]. In this

Symbol	Description
q	Video segment quality level
r_q	Video quality bitrate of quality level q
$R_{i,q}$	Segment download rate in bit/sec for segment i, quality level q
$t_{i,q}$	Video segment fetch time of segment number i in quality level q
$B(i)$	The playout buffer size in terms of number of segments
$d_{i,q}$	The download rate for segment number i, quality q
$s_{i,q}$	Segment size of segment number i, quality q
$t_i^{c_j,\texttt{delivered}}$	Time required to download c_j bits of segment i in quality q
$\hat{d}^{\varepsilon}_{s_{i,q}}$	Empirical lower bound of download rate of segment number i, quality q
$\mathcal{L}^{\alpha}_{n+1}(v)$	Weighted quality list
$H(N)$	Spectrum given window with N segments
q_i^C	Chosen quality of segment number i
q_i^s	Sustainable quality of segment number i
c_l and c_h	Lower and upper buffer filling thresholds
$\hat{t}_{A_{r,q}}$	Estimated available download time for segment r in quality q

Table 5.1: Symbols of notations.

chapter, we slightly adapt the spectrum definition to DASH streaming to express the video bitrate variation around the average bitrate given N segments as

$$H(N) = \sum_{i=1}^{N} z_i \left(r_{q_i} - \frac{\sum_{j=1}^{N} z_j r_{q_j}}{\sum_{j=1}^{N} z_j} \right)^2 \tag{5.5}$$

where $z_i = \mathbb{1}_{\{r_{q_i} \neq r_{q_{i-1}}\}}$. In the following, we describe our DASH quality adaptation algorithm that has the objective of maximizing (5.4) while minimizing the spectrum (5.5). From a classification point of view, our quality adaptation algorithm can be regarded as a rate *and* buffer based algorithm, since it takes the information on the buffer filling as well as the historical download rates to decide on the quality of the next segment.

5.4.1 Smooth and Reliable Rate Estimation

Given our observation in Sect. 5.3 that the measured segment download rate varies with the segment size and the current state of the server TCP state machine, we introduce two key ideas to provide smooth and reliable estimates of the download rate of the next segment to be fetched.

Smooth sub-segment rate estimates: Looking at the download rate estimation methods from Sect. 5.3, we observe that the estimates are usually calculated upon the arrival of a video segment. Since segments vary in size, the different estimates correspond to different time scales. Hence, given the knowledge of the time scale dependence of the behavior of network protocols (such as TCP fairness) and the available bandwidth, we decide to take running estimates of the download rate vs. the downloaded data amount. Starting from the request time $t_n^{\texttt{GET}}$ for segment n we calculate the running download rate estimate as

$$\hat{d}_s = \frac{s}{t_n^{c_j,\texttt{delivered}} - t_n^{\texttt{GET}}} \tag{5.6}$$

where s represents the segment size, $t_n^{c_j,\texttt{delivered}}$ is the time required to download c_j bits of segment n in quality q, where $c_j \leq s_{n,q}$. Figure 5.6a depicts this procedure. In the following, we calculate $\hat{d}_s$ from (5.14) using the granularity of $c_{j+1} - c_j = 50$ IP packets (roughly 75kB). The granularity of the smooth sub-segment rate estimates technique determines the smallest segment size for which an acceptable estimate accuracy can be obtained. It also determines the computational complexity of the estimation through the number of given samples. This parameter can be set at the start of a streaming session as soon as the segment sizes of a video are known from its MPD file.

Reliable rate estimates: One important contributor to the efficiency of any quality adaptation algorithm for DASH is the accuracy of the download rate estimation. However, in Sect. 5.3, we showed that the segment download rate in case of DASH depends on the segments size as well as the TCP state. Practical approaches to predict the TCP transmission rate that are based, e.g., on machine learning techniques have been considered in [90]. In contrast, in this work we assume a thin client that keeps track of the segment download history. However, we do not aim to predict the TCP transmission rate in such manner. Instead, we collect the download rates $\hat{d}_s$ similar to Fig. 5.3 and deduce an empirical lower bound on the download rate for the next segment $n+1$ of size $s_{n+1,q}$ from the percentile of the empirical distribution as

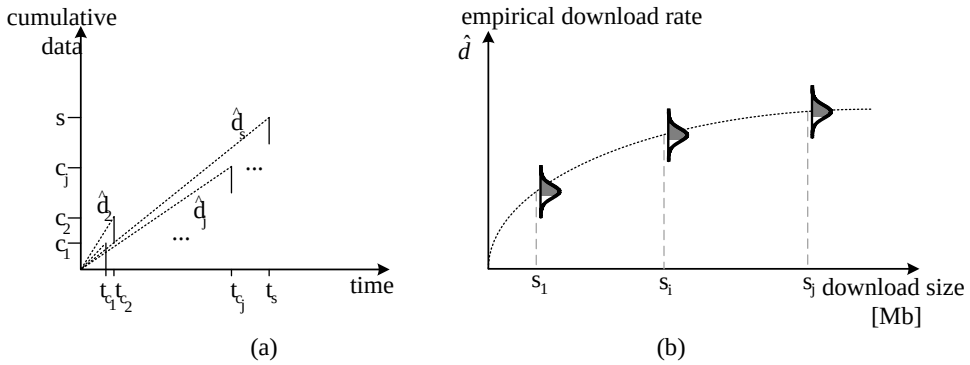


Figure 5.6: (a) Sub-segment rate estimation. (b) Empirical downloading rate vs. segment download size.

$$P\left[r_{n+1,q} \leq \hat{d}^{\varepsilon}_{s_{n+1,q}}\right] \leq \varepsilon, \tag{5.7}$$

where ε is a conservatism parameter. The smaller ε, the more conservative is the lower bound $\hat{d}^{\varepsilon}_{s_{n+1,q}}$ for the download rate of the next segment, as sketched in Fig. 5.6b.

5.4.2 Init: A Slow Start of Segment Quality

Initially, we start the streaming session by retrieving the corresponding MPD file which we modified to include the segments sizes (in kB). Modifying the MPD to include segment size was first proposed by [54]. Note that our implementation can be easily adapted to other techniques used to obtain the actual segment sizes, e.g., through HTTP range requests. Since we do not assume any prior information on the available bandwidth along the path between the client and the streaming server, we stream the first segment with the lowest possible quality. Instead of streaming only the first segment in the lowest quality, we may stream a consecutive train of the first W_1 segments in the lowest quality. This conservative yet tunable parameter allows the player to quickly accumulate the buffer and therefore reduce the risk of rebuffering at the beginning of playback. However, a larger W_1 value keeps the player longer at the lowest quality, which increases the risk of video abandonment. In the sequel, we set $W_1 = 5$ if not stated otherwise.

The segment quality *slow start* behavior starts from segment $W_1 + 1$. Here, with every segment we double the quality level requested until we reach the highest possible quality.

We conclude the initial phase and go to the next phase, where we denote the steady state, either when the quality slow start is finished or when the download time of the last fetched segment is longer than doubling the segment length X. We choose this empirical break condition to minimize the risk of rebuffering in the initial phase.

5.4.3 Steady State

In this section, we describe how the player decides on the quality of the next requested segment based on our spectrum based quality adaptation.

5.4.3.1 Spectrum-based Adaptation

After the initial phase, i.e., $W_1 + \lfloor \log(Q) \rfloor$ segments, the player decides on the rest of the video qualities using the spectrum H given in (5.5). Given the trace of quality levels downloaded so far, the player aims at minimizing (5.5) for all possible quality levels for the next segment. However, this strategy leads to the preference of segment qualities that are close to the average quality bitrate and may not necessarily have a drift towards higher qualities. Since the goal of our adaptation algorithm is to maximize the average bitrate (5.4) subject to minimizing the quality variations around the mean (5.5), we modify the adaptation algorithm as follows.

We sort the qualities according to their calculated spectrum H, where we mark the qualities either as "sustainable" or "unsustainable", according to their estimated fetch time $\hat{t}$. We denote a quality sustainable if its fetch time $\hat{t}$ is less or equal to the X seconds of video contained in that segment. Considering the playout buffer evolution (5.1), this is equivalent to imposing a buffer constraint such that we do not fetch qualities that lead to buffer drain. In the sequel we will relax this constraint. This ordering operation is depicted in Fig. 5.7 and it provides the player with a first reference for choosing the quality of the next segment. We conjecture that, in general, consumers are less likely to recall the movie quality after a few minutes. Therefore, the *current* QoE is a function of the segment qualities over a window v of past segments. Hence, as shown in Fig. 5.7, given n downloaded segments

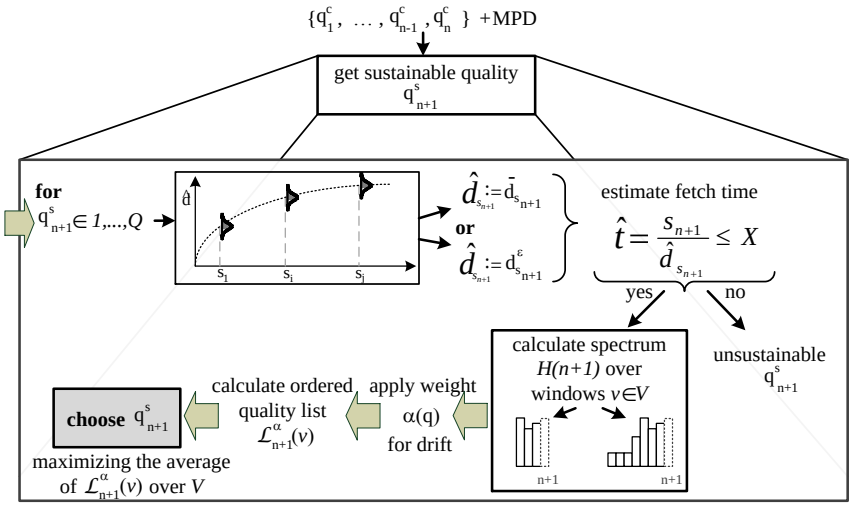


Figure 5.7: Spectrum based sustainable quality identification.

with qualities q_i^C with $i \in \{1, \ldots, n\}$, we calculate the ordered quality lists $\mathcal{L}_{n+1}(v)$ for segment $n+1$ for multiple backward window lengths $v \in V$, where V denotes the set of window lengths used to calculate the spectrum. To include the drift to higher qualities, we multiply each of the values $\mathcal{L}_{n+1}(v)$ with a corresponding quality weight

$$\alpha(q) = \left(\frac{r_1}{r_Q}\right)^{\frac{1}{Q-q+1}}. \tag{5.8}$$

Note, the bitrate ratio $r_1/r_Q < 1$ and the weighting function $\alpha(q)$ is concave in $Q - q$.

We combine the weighted quality lists $\mathcal{L}^{\alpha}_{n+1}(v)$, i.e., the element wise product of the two vectors α and $\mathcal{L}_{n+1}(v)$, and calculate the average of $\mathcal{L}^{\alpha}_{n+1}(v)$ over multiple v. The basic adaptation algorithm sets the so-called *chosen* quality for the next segment q^C_{n+1} to the *sustainable* quality q^s_{n+1} which minimizes the average

$$\sum_{v \in V} \frac{1}{|V|} \mathcal{L}^{\alpha}_{n+1}(v), \tag{5.9}$$

where $|V|$ denotes the cardinality of V. Next, we use $V = \{4, 8, 16\}$ segments.

5.4.3.2 Buffer Guidance - Latent Fallback

Now we turn our attention to relax the restrictive buffer constraint from above that we do not fetch qualities that lead to buffer drain. In general, our aim is to stream a movie in highest sustainable quality while minimizing the quality variations. Conceptually, a quality adaptation strategy that tries to keep the buffer filling fixed will eventually *follow* the variations of the available bandwidth. Hence, we use a buffer guidance approach to complement the spectrum based adaptation from Sect. 5.4.3.1. Here, we allow the playout buffer B to drain at most by a certain amount whenever the available bandwidth decreases and the current quality level becomes unsustainable. This latent fallback allows the player to sacrifice buffer filling to maintain unsustainable but spectrum minimizing quality levels for short periods of time in order to be able to overcome temporary available bandwidth fluctuations. This, however, is only possible when the buffer filling is high enough to minimize the risk of rebuffering. In the following we describe the details of this algorithm.

As depicted in Fig. 5.8, we divide the playout buffer into three areas, "low", "medium" and "high". Similar divisions have been introduced in [54] and `VLC` [72]. In our case, we only allow latent fallback when the buffer is in the "high" region. We mark the buffer division by cutoff percentages c_l and c_h, where the subscripts stand for the lowest and highest area. When the buffer filling is above c_h we calculate the ordered quality lists $\mathcal{L}^{\alpha}_{n+1}(v)$ as in Sect. 5.4.3.1 but we *only* mark qualities as unsustainable, that lead to a buffer drain below c_l. In other words, we mark qualities q as unsustainable only when

$$\hat{B}(n+1) := B(n) - \hat{\delta}(n+1, q) < c_l, \tag{5.10}$$

where $\hat{\delta}(n+1, q) := \hat{t}_{n+1,q} - X$ is the estimated buffer drain due to fetching segment $n+1$ in quality level q. We keep applying the latent fallback algorithm as long as the buffer filling is above c_h. As soon as the buffer filling is equal or below c_h, we return to the buffer constraint from Sect. 5.4.3.1.

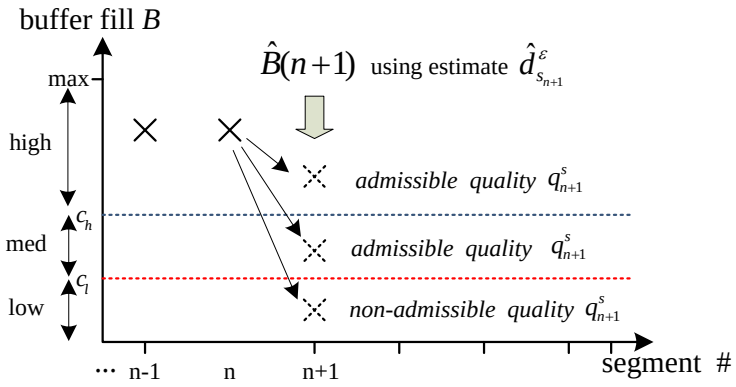


Figure 5.8: Estimated buffer drain $\hat{\delta}(n+1,q) = B(n) - \hat{B}(n+1)$ determines the admissibility of a proposed quality q^s_{n+1}. Latent fallback is viable only if $B(n)$ is above c_h.

5.4.3.3 Player States

In steady state we differentiate between three states in which the player can be, which we denote as **(i) decreasing**, **(ii) steady** and **(iii) increasing**. These states simply describe the relation between the current *chosen* segment quality q^C_n and the *sustainable* quality level that is calculated for the next segment to be fetched q^s_{n+1}. Based on its state, the player decides on the *chosen* segment quality for the next segment q^C_{n+1}. Next, we describe the steps that are common to all states before delving into the particular details of every state. Fig. 5.9 shows an overview of the entire algorithm.

Before fetching a new segment, we always calculate the corresponding *sustainable* quality level q^s_{n+1} as described in Sect. 5.4.3.2. In particular, q^s_{n+1} is calculated as the quality which minimizes the spectrum, technically (5.9), while meeting the stringent buffer constraint $\hat{t}_{n+1} \leq X$ from Sect. 5.4.3.1. To this end, we first calculate the estimated fetch time for segment $n+1$ in quality q for $q \in \{1,\ldots,Q\}$ as $\hat{t}_{n+1,q} = \frac{s_{n+1,q}}{\bar{d}_{s_{n+1,q}}}$, where $\bar{d}_{s_{n+1,q}}$ denotes the average download rate estimated empirically from Fig. 5.6b (similar to Fig. 5.3) given the size of the next segment $s_{n+1,q}$. Hence, as stated above, we find q^s_{n+1} that minimizes (5.9) and compare it to q^C_n to detect the current player state. Next, we describe the steps carried out when the different player states are detected:

Decreasing: In the decreasing state we detect that the sustainable quality level is less than the current chosen quality, $q^s_{n+1} < q^C_n$. Here we first invoke the latent fallback technique that is described in Sect. 5.4.3.2 to avoid rapid quality changes in exchange for buffer filling. To this end, we recalculate the sustainable quality q^s_{n+1} that minimizes (5.9), however, using the ε percentile $\hat{d}^{\varepsilon}_{s_{n+1,q}}$ from (5.7). In the following, we set the percentile $\varepsilon = 0.2$ to strike a balance between smoothness and responsiveness. Next, we consider the buffer filling as in Sect. 5.4.3.2. If the buffer filling is higher than c_h, i.e., we have enough segments buffered, we may sacrifice buffer filling in exchange for holding the quality level q_n. Hence, as long as the estimated buffer underflow probability after fetching the next segment is less than ε, i.e.,

$$P\left[\hat{B}(n+1) < c_l\right] \leq \varepsilon, \tag{5.11}$$

we set the *chosen* quality for the next segment as the average $q^C_{n+1} := (q^C_n + q^s_{n+1})/2$ with q^s_{n+1} calculated using the percentile $\hat{d}^{\varepsilon}_{s_{n+1,q}}$. Note that we do not perform this averaging procedure more than once during one decreasing period, since the idea here is to hold a moderate quality level as long as the buffer filling permits. As described in Sect. 5.4.3.2, as soon as the current buffer filling q^C_n falls below c_h, we set the chosen quality level as $q^C_{n+1} := q^s_{n+1}$. This procedure is depicted in Fig. 5.9.

Increasing: In the increasing state we observe that $q^s_{n+1} > q^C_n$ through the use of the average download rate $\bar{d}_{s_{n+1,q}}$. This state denotes that the player has room for increasing the segment quality. In this case we set the chosen quality as $q^C_{n+1} := q^s_{n+1}$.

Steady: In the steady case we detect no change in the calculated sustainable quality level with respect to the previously fetched segment, i.e., $q^s_{n+1} = q^C_n$. In this case, we decide to be cautious and recalculate q^s_{n+1} using the percentile $\hat{d}^{\varepsilon}_{s_{n+1,q}}$. If q^s_{n+1} stays unchanged, then we set the quality as $q^C_{n+1} := q^s_{n+1}$; if we detect $q^s_{n+1} < q^C_n$ then we undergo the same procedure as in the decreasing state.

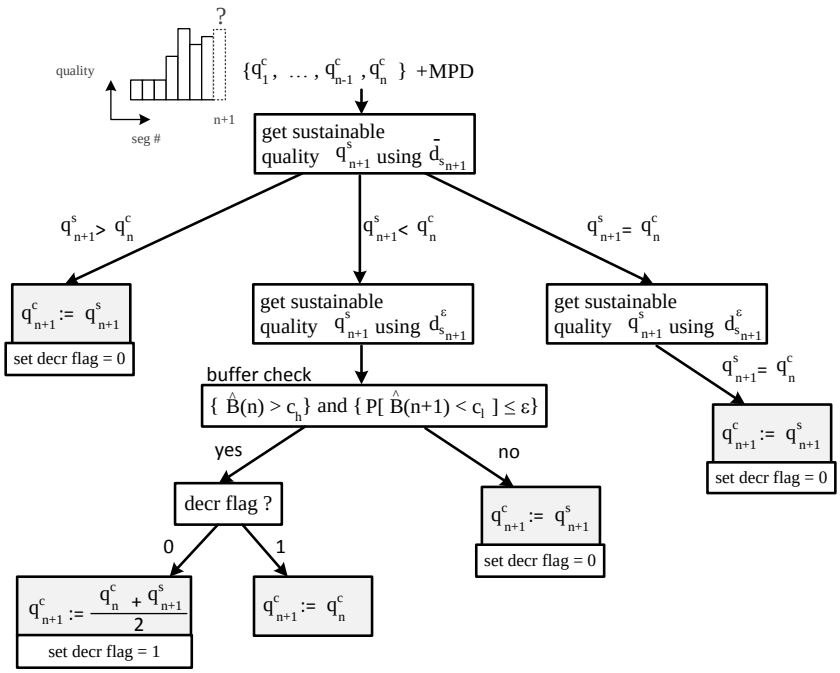
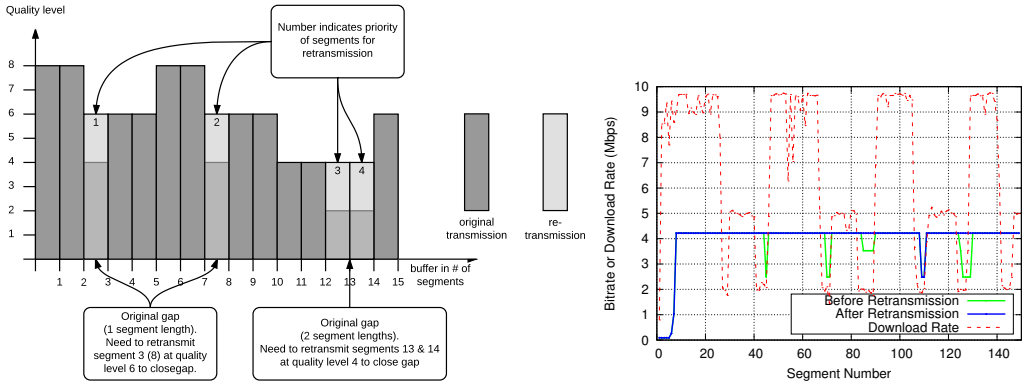


Figure 5.9: High-level sketch of the SQUAD algorithm. The calculation block to get the sustainable quality is given in Fig. 5.7. The buffer drain consideration in the decreasing player state is given in Fig. 5.8.

5.4.4 Segment Retransmission Scheduling

Traditional ABR approaches stream the video segments in the order provided by the MPD file. Looking closely at the segment qualities buffered at the client at any point in time we find that these reflect the recent quality decisions made by the adaptation algorithm, which, in turn, are based on the specific interpretation of the measured download rate and the corresponding buffer filling. Looking at the buffer filling in *retrospect* as in Fig. 5.10 (a) we identify quality switches that we denote as quality "gaps". The emergence of these quality gaps is complex as it describes the instantaneous interaction of the adaptation algorithm with the buffer filling state and the download rate. In the following, we illustrate how to increase the QoE by filling some of these quality gaps. Fig. 5.10 (a) shows a simplified example of a sample path of segment qualities inside the player buffer with different possible gaps. We define gaps as the downward variation from the quality level which minimizes the spectrum (5.5).

In the following, we introduce a retransmission scheduling algorithm that detects the quality gaps in the client buffer, and enables the replacement of low quality buffered segments with higher quality ones upon download rate improvement. In addition to gap detection, we predict the segment download finishing time based on the download rate history to decide the feasibility of retransmission and prioritize the gaps that need to be filled. Furthermore, to ensure the retransmitted segments can be downloaded without impacting the regular transmitted segments, we monitor the segment downloading process in real time and abandon the retransmitted segment if it cannot be downloaded before playout. Hence, by enabling retransmissions we aim to further reduce the number of quality switches leading to higher QoE. Video segment retransmission method has been initially introduced by Zink et al. [107]. While retransmissions can be regarded as an additional burden on the available bandwidth we note that recent works such as [95] suggest different types of redundant transmission to provide higher QoS. In contrast, in our work, we only invoke retransmissions when it is nearly guaranteed that this will improve QoE, as shown in a session of measurements in Fig.5.10 (b).



(a) Retransmission illustration.

(b) Single run retransmission example.

Figure 5.10: Example of SQUAD retransmissions (sketch and actual measurement run).

Next, we introduce two retransmission approaches along with the retransmission segment abandonment mechanism.

Segment download time estimation: To ensure a smooth playback experience, segments need to be completely retransmitted before the player starts rendering them. The available fetch time for a segment can be represented by:

$$\hat{t}_{A_{r,q}} = t_r - t_{curr} \tag{5.12}$$

where *(i)* $\hat{t}_{A_{r,q}}$ denotes the estimated available download time for segment r in quality q, *(ii)* t_r represents the playback time of targeted retransmission segment, and *(iii)* t_{curr} denotes the time of current played segment.

With the help of estimated download rate $\hat{d}$, we estimate the fetch time of the retransmission segment r as:

$$\hat{t}_{r,q} = \frac{s_{r,q}}{\hat{d}^{\varepsilon}_{s_{r,q}}}. \tag{5.13}$$

If $t_{A_{r,q}} > t_{r,q}$ we have more time to retransmit the new segment before segment r has to be rendered. Retransmissions are only possible in steady state as described in Sect. 5.4.3. Given that we detect multiple quality gaps in the playout buffer as in Fig. 5.10 we need to prioritize the gaps to be filled. The spectrum-minimizing prioritization scheme can be mapped to the computationally intractable Knapsack problem [107]. Hence, we devise the following heuristic that has been observed to provide the best performance. Here, we prioritize the gaps to be filled according to the following rules:

1. **Narrowest gap:** The first priority goes to closing as many gaps as possible, therefore, we start with the gaps with least number of segments.

2. **Largest quality switch magnitude:** As a second priority we close the gaps with largest quality switch magnitude. This is due to the fact that we need to retransmit entire segments in DASH, hence, we aim at improving the quality by as many layers as possible, which also minimizes the spectrum in (5.5).

3. **Lowest quality layer:** Finally, we close gaps with the lowest quality layer first, in case we have multiple gaps with the same width and quality switch magnitude.

Based on the estimated segment fetching time (5.13), we design two retransmission mechanisms based on 1) segment downloading rates, that we denote *SQUAD-RR* and 2) client buffer level, denoted *SQUAD-BR*.

Rate based retransmission: In this case, we decide whether to carry out retransmission by comparing the estimated download rate to the segment bitrate. The retransmission can be triggered only when: $\min\{\hat{d}^{\varepsilon}_{s_{r,q}}, \hat{d}^{\varepsilon}_{s_{n+1,q}}\} > \kappa(q_r + q_{n+1})$, i.e., when the effective download rate sustains both the new segment $n+1$ and the retransmitted segment r. In our experiments we use $\kappa = 1.2$ to account for bandwidth fluctuations.

Buffer based retransmission: The second retransmission approach is based on the client buffer level. Here, we enable retransmissions when the buffer filling is in the high region c_h (see Fig. 5.8). During retransmissions we monitor the buffer level and allow retransmissions to continue as long as the playout buffer filling is in high or medium region. If the buffer drops below the threshold c_l, we stop retransmissions and resume the normal segment downloading process to avoid further buffer draining.

Retransmission segment abandonment: The retransmission process bears the risk of wasting valuable download bandwidth when retransmitting segments in addition to transmitting regular ones. Therefore, we design a segment abandonment mechanism which only applies to segments that are being retransmitted. In this case, we monitor the download rate on sub-segment level, i.e.,

$$\hat{d}_s = \frac{s}{t_r^{c_j,\texttt{delivered}} - t_r^{\texttt{GET}}} \tag{5.14}$$

where $t_r^{c_j,\texttt{delivered}}$ is the time required to download c_j bits of segment r in quality q, where $c_j \leq s_{r,q}$. We abandon the retransmission process when we observe that the segment will not be downloaded on time according to the estimated download rate $\hat{d}^{\varepsilon}_{s_{r,q}}$, i.e., $t_{A_{r,q}} < \delta \cdot t_{r,q}$. In our implementation, we use a conservative factor $\delta = 1.5$, in order to avoid frequent interruption of the retransmission process.

5.5 Experimental Evaluation

In the following, we conduct a number of experiments in a controlled testbed, the GENI testbed, as well as in the public Internet to evaluate the performance of SQUAD.

For all experiments we make use of an excerpt of the `BigBuckBunny` dataset [63] that comprises a video that is 300 seconds long and an MPD that describes attributes of the video. We extended the MPD file by providing the size of each segment in each of the available quality levels. The quality bitrates available in this MPD are the following {`0.09, 0.13, 0.18, 0.22, 0.26, 0.33, 0.59, 0.79, 1.03, 1.24, 1.54, 2.48, 3.52, 4.21`} Mbps.

For better judgement of the performance of SQUAD we compare its performance with the ones of three additional algorithms, which we are denoted "VLC", "SARA", "Buffer-based" and "BOLA". We briefly describe them in the following:

VLC: The first algorithm we decided to chose for comparison is a basic quality adaptation algorithm from [72], as introduced previously in Section 4.4.

SARA: The second algorithm we use for comparison has been proposed in [54] and coined segment aware rate adaptation (SARA). The algorithm predicts the time required for fetching a segment based on its size and the available bandwidth estimate through a weighted harmonic mean. Further, SARA selects the bitrate depending on the current buffer filling and drops to the lowest bitrate if the buffer filling falls below a certain threshold. Note that the SARA implementation accompanying [54] uses non-persistent HTTP by default.

SQUAD: Our proposed spectrum-based quality adaptation algorithm is explained in detail in Sect. 5.4.

Buffer-based: This algorithm is denoted BBA-0 in [50] and is implemented as part of the Python DASH client emulator accompanying [54]. In a nutshell, the algorithm defines a class of functions that map current buffer occupancy to a quality bitrate (denoted rate map) to avoid unnecessary rebuffering and maximize the average video rate. This algorithm was part of a wide-scale Netflix experiment in [50].

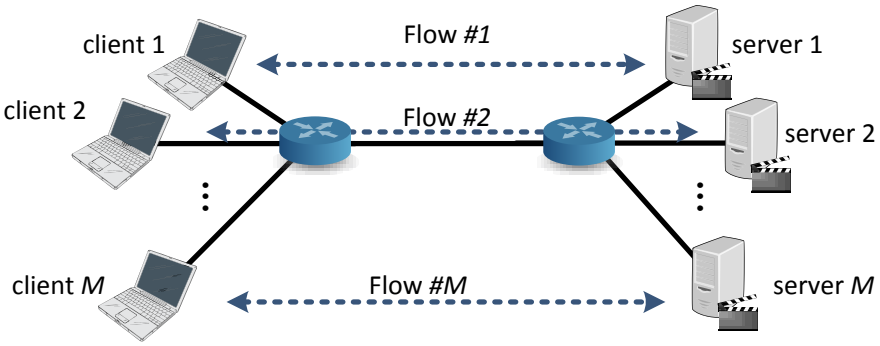


Figure 5.11: Butterfly evaluation topology.

BOLA: This buffer-based algorithm [88] is currently implemented in the DASH-JS player. In this chapter, we compare our SQUAD algorithms with two variations of BOLA: BOLA-U maximizes a weighted combination of the bitrate and the smoothness measured in the average rebuffering time; and BOLA-O addresses the trade-off between maximizing the bitrate and reducing oscillations during playback.

5.5.1 GENI Single Run Experiments

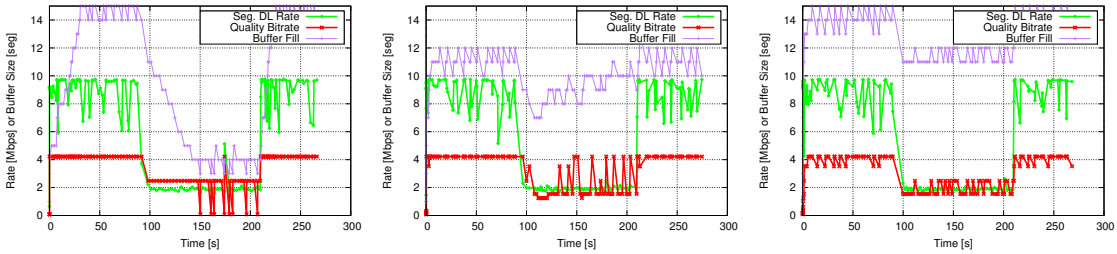
The GENI (Global Environment for Networking Innovation) testbed is a distributed virtual laboratory sponsored by the U.S. National Science Foundation (NSF). It allows researchers to obtain a virtualized and isolated slice of compute, storage, and networking resources for the development and validation of new approaches in networking and distributed systems [27, 28]. GENI allows the setup of larger and wide-area topologies.

For the evaluation in the GENI testbed we create a slice that comprises a butterfly topology as shown in Fig. 5.11. In this section, we show the single-run experiments, in order to take a close look at the playback performance amongst different algorithms. In the following experiments, we stream the DASH video from server i to client i for $i \in \{1, 2, 3\}$. For the experiments with one DASH flow we utilize server 1 and client 1 and for the cross traffic flow we utilize server 2 and client 2. All links possess a capacity of 10 Mbps.

We first take a close examination of all the algorithms by looking into single-run experiments. In Fig. 5.12 to 5.20 we run (i) contiguous UDP cross traffic of 8 Mbps for 2 minutes

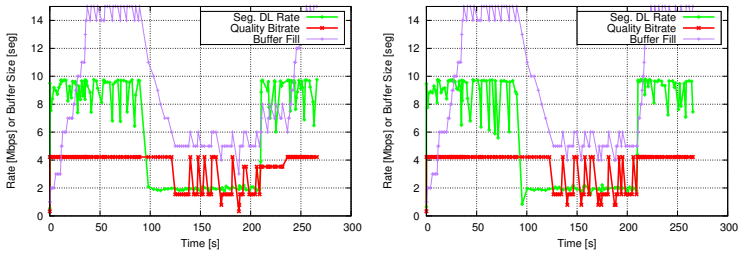
(available bandwidth has U-shape), (ii) two-level UDP cross traffic of 8 and 5 Mbps (available bandwidth has W-shape) and (iii) alternating ON-OFF UDP cross traffic of 8 Mbps in the ON state. Fig. 5.12 to 5.20 depict 5 minute sample runs for each of the studied algorithms showing the following: $1)$ The empirical segment download rate (denoted in figures as seg. DL rate), $2)$ the quality bitrate (important QoE metric), $3)$ the buffer filling over time, as well as, $4)$ instantaneous rate measurements of the cross traffic (shown as crosses) and of the DASH flow (shown as dots). In all following figures, the y-axis denotes the rate in Mbps *and* for the buffer filling curves it denotes the buffer length in segments. Note that one segment is 2 seconds long in this dataset and that we set the maximum buffer size to 30 seconds, i.e., 15 segments.

From Fig. 5.12 to 5.20 we deduce the following observations: First, the VLC algorithm is highly aggressive in choosing the quality bitrate which may substantially drain its playout buffer, enforcing it to significantly reduce the quality bitrate when the buffer reaches 25%. This leads to high quality jump magnitudes which are detrimental to QoE. Secondly, SARA introduces many oscillations of the fetched quality bitrate around the available bandwidth which is harmful to the QoE. BOLA-O and BOLA-U can provide good results in certain cases, e.g., the one with UDP ON-OFF cross traffic, however, in the case with other types of cross traffic, e.g., UDP-U, BOLA can introduce many quality switches. On the contrary SQUAD performance is smooth: With the latent fallback and the percentile, respectively, average rate estimation it holds the quality bitrate over temporary available bandwidth fluctuation. The buffer-based algorithm BBA possesses many unnecessary quality switches. The quantitative results of the average quality bitrate, the number of quality jumps, as well as, the spectrum H are given in Tab. 5.2. These metrics show that SQUAD provides a significant QoE improvement as seen by the strong reduction in the number of quality jumps [107] while sacrificing little or no average quality bitrate. SQUAD also outperforms its competitors in minimizing the variation of the quality bitrates (spectrum).



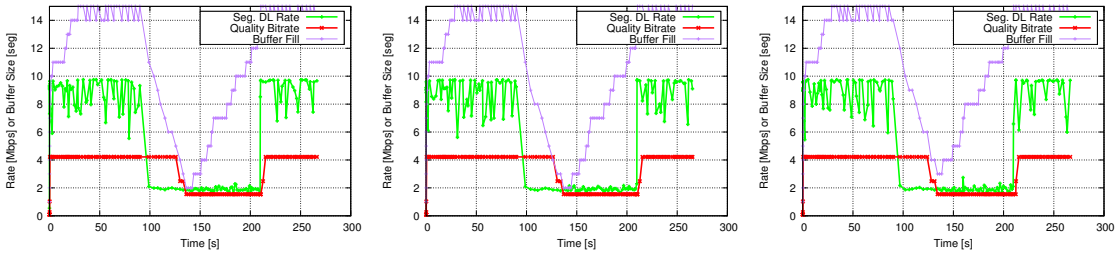
(a) VLC. (b) SARA. (c) BBA.

Figure 5.12: Quality bitrate with UDP-U cross traffic for VLC, SARA and Buffer-based [50] algorithm (BBA). (Implementation accompanying [54]).



(a) BOLA-O. (b) BOLA-U.

Figure 5.13: Quality bitrate with UDP-U cross traffic for BOLA-O and BOLA-U.

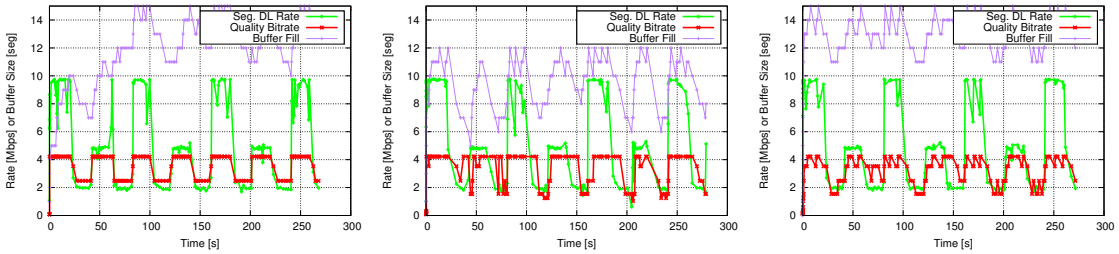


(a) SQUAD. (b) SQUAD-BR. (c) SQUAD-RR.

Figure 5.14: Quality bitrate with UDP-U cross traffic for SQUAD, SQUAD with buffer based retransmission (SQUAD-BR) and SQUAD with rate based retransmission (SQUAD-RR).

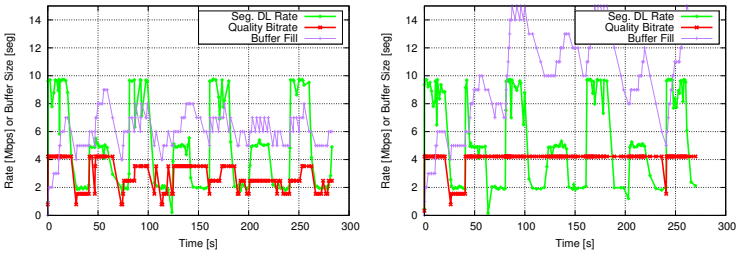
5.5.2 GENI Multi-Run Experiments

In this section we show the GENI experiments with multiple runs. These experiments allow us to get a comprehensive idea of the performance amongst the algorithms in the



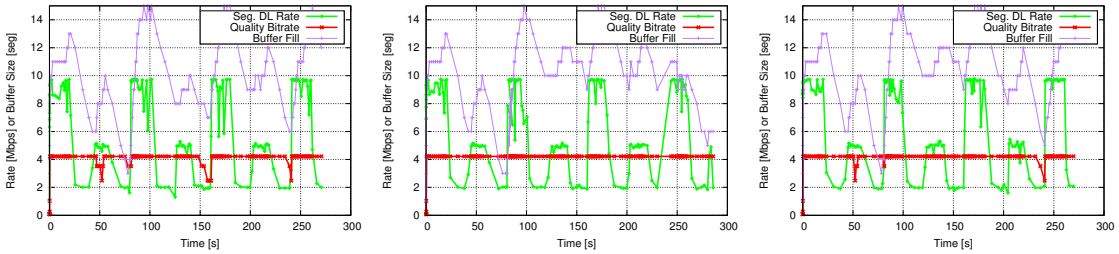
(a) VLC. (b) SARA. (c) BBA.

Figure 5.15: Quality bitrate with UDP-W cross traffic for VLC, SARA and BBA.



(a) BOLA-O. (b) BOLA-U.

Figure 5.16: Quality bitrate with UDP-W cross traffic for BOLA-O and BOLA-U.

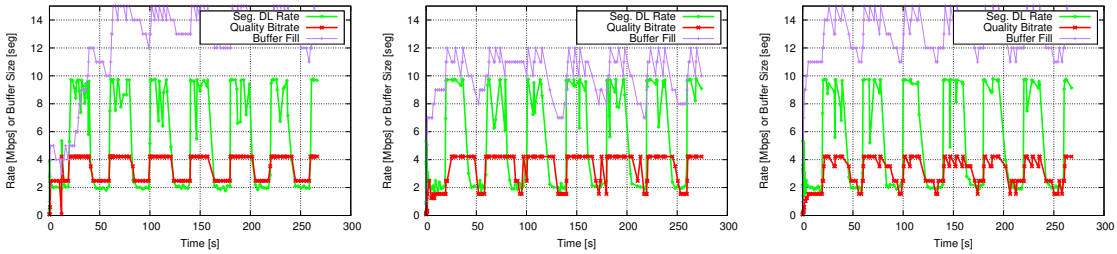


(a) SQUAD. (b) SQUAD-BR. (c) SQUAD-RR.

Figure 5.17: Quality bitrate with UDP-W cross traffic for SQUAD, SQUAD-BR and SQUAD-RR.

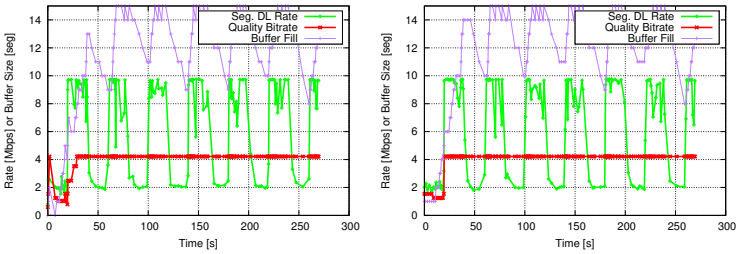
controlled experimental environment. In the first set of experiments we individually run the DASH algorithms solely in parallel to UDP and TCP cross traffic.

In addition to the UDP experiments mentioned in 5.5.1, we evaluate the performance of the DASH algorithms while competing with aggregate TCP cross traffic. In this case, we



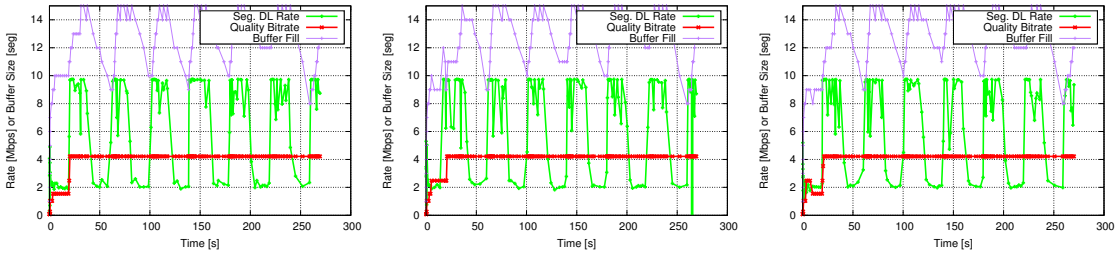
(a) VLC. (b) SARA. (c) BBA.

Figure 5.18: Quality bitrate with UDP ON-OFF cross traffic for VLC, SARA and BBA.



(a) BOLA-O. (b) BOLA-U.

Figure 5.19: Quality bitrate with UDP ON-OFF cross traffic for BOLA-O and BOLA-U.



(a) SQUAD. (b) SQUAD-BR. (c) SQUAD-RR.

Figure 5.20: Quality bitrate with UDP ON-OFF cross traffic for SQUAD, SQUAD-BR and SQUAD-RR.

set up the butterfly topology with $M = 10$ client/server pairs, and 20 Mbps capacity on all links. Note that in this experiment the fair share amounts to 2 Mbps per client/server pair. We run a single DASH client, along with 9 concurrent TCP cross traffic streams.

Average Quality Bitrate [Mbps]																
Cross traffic	VLC		BBA		SARA		SQUAD		SQUAD-BR		SQUAD-RR		BOLA-U		BOLA-O	
	avg.	SD	avg.	SD	avg.	SD	avg.	SD	avg.	SD	avg.	SD	avg.	SD	avg.	SD
UDP-U	3.42	0.01	2.91	0.01	3.15	0.01	3.03	0.02	3.03	0.03	3.03	0.02	3.56	0.02	3.47	0.02
UDP-W	3.60	0.02	3.09	0.02	3.16	0.05	3.95	0.03	3.99	0.01	3.95	0.03	3.94	0.04	3.04	0.37
UDP-ON/OFF	3.47	0.03	2.95	0.03	3.16	0.02	3.80	0.24	3.84	0.03	3.81	0.03	3.93	0.01	3.87	0.02
TCP	1.67	0.16	1.69	0.15	1.53	01.5	1.69	0.23	1.64	0.18	1.67	0.27	1.74	0.23	1.25	0.13
# of Quality Switches																
Cross traffic	VLC		BBA		SARA		SQUAD		SQUAD-BR		SQUAD-RR		BOLA-U		BOLA-O	
	avg.	SD	avg.	SD	avg.	SD	avg.	SD	avg.	SD	avg.	SD	avg.	SD	avg.	SD
UDP-U	17.1	1.98	66.0	3.77	39.8	2.66	7.05	0.21	7.15	0.35	7.0	0.2	19.7	3.4	20.9	2.71
UDP-W	18.2	2.09	69.7	4.21	49.9	4.28	9.1	2.46	4.4	1.49	9.7	2.26	9.5	2.1	30.5	9.4
UDP-ON/OFF	23.9	2.9	66.8	3.1	37.1	2.6	6.7	0.95	4.7	1.1	6.6	1.0	5.3	0.7	7.5	1.8
TCP	91.3	5.79	86.2	18.1	57.1	9.38	18.5	6.36	18.9	6.83	20.2	5.76	51.1	15.1	30.3	9.81
Spectrum																
Cross traffic	VLC		BBA		SARA		SQUAD		SQUAD-BR		SQUAD-RR		BOLA-U		BOLA-O	
	avg.	SD	avg.	SD	avg.	SD	avg.	SD	avg.	SD	avg.	SD	avg.	SD	avg.	SD
UDP-U	1587	237	6602	391	3037	223	461	44	464	47	447	27	1505	193	1361	153
UDP-W	1749	278	7228	505	4211	416	766	281	237	155	836	257	889	191	2328	811
UDP-ON/OFF	1811	381	6962	334	2899	205	438	103	252	108	412	94	339	58	610	119
TCP	3704	346	2969	624	3211	694	1162	421	1152	491	1256	379	2179	609	1375	370

Table 5.2: QoE metrics for the GENI experiments with controlled UDP and TCP cross traffic. The table includes averages as well as standard deviations for 20 runs.

Fig. 5.21 to 5.23 show empirical CDFs (eCDFs) of three quantitative QoE metrics of interest, i.e., the chosen (quality) bitrate, the number of quality switches and the spectrum H as defined in (5.5). Table 5.2 shows corresponding average and standard deviation values. The eCDFs are calculated using 20 independent runs for all the DASH algorithms. From Figs. 5.21 to 5.23 we deduce the following observations: First, the algorithms $\{$VLC, BBA, SARA$\}$ are highly aggressive in choosing the quality bitrate which may substantially drain the playout buffer. This leads to higher quality jump magnitudes which are detrimental to QoE. Especially the $\{$BBA and SARA$\}$ algorithms introduce many quality oscillations around the available bandwidth which is harmful to the QoE. On the contrary SQUAD (with and without retransmissions) has a smooth performance: With the latent fallback and the percentile/average rate estimation SQUAD keeps the quality bitrate relatively stable over temporary bandwidth fluctuation. Comparing SQUAD with BOLA reveals a very similar performance and one or the other performs slightly better depending on the cross traffic scenario. Generally, there can be seen a trade-off between the two approaches, which is

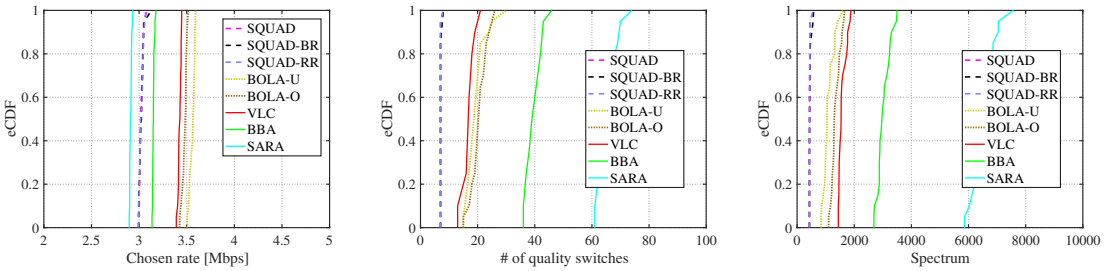
most prominent in the UDP U-shape case. While the BOLA algorithms achieve a slightly higher quality rate, the number of quality switches and the spectrum H are significantly lower for the SQUAD algorithms.

UDP-U: While the BOLA algorithms achieve a slightly higher quality rate, the number of quality switches and the spectrum H are significantly lower for the SQUAD algorithms. This indicates that the SQUAD algorithms result in better QoE in the case of longer-term, significant changes in available bandwidth.

UDP-W: In this case, the performance of both algorithm classes (BOLA and SQUAD) is almost identical when it comes to average quality bitrate. The results for number of quality switches and spectrum presented in Table 5.2 show that the buffer-based retransmission algorithm (SQUAD-BR) significantly improves the QoE.

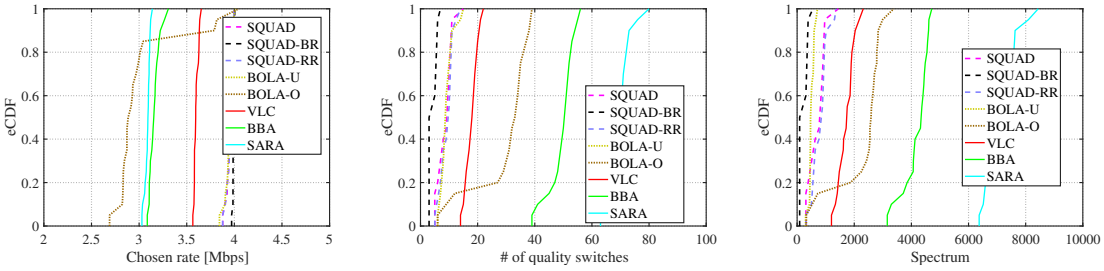
UDP-ONOFF: In the UDP-ONOFF case, the algorithm classes perform almost identical. This indicates that they can handle rapid changes in available bandwidth quite well. Taking a closer look at the results presented in Table 5.2 reveals that the buffer-based retransmission approach (SQUAD-BR, see Sect. 5.4.4) has a positive impact on the QoE. While the average quality bitrate is similar to the other SQUAD and BOLA algorithms, the number of quality switches and the value of the spectrum are reduced. The reduction is more significant in the case of the spectrum.

TCP: In the case of aggregate TCP cross traffic, BOLA-U achieves a higher quality bitrate, however, its aggressiveness in increasing the quality bitrate leads to a higher number of quality switches and a higher spectrum. BOLA-O achieves a low number of quality switches (low spectrum), however, it suffers from a lower quality bitrate. SQUAD algorithms achieve comparable quality bitrates, yet they are able to significantly improve the QoE in terms of quality switches and spectrum.



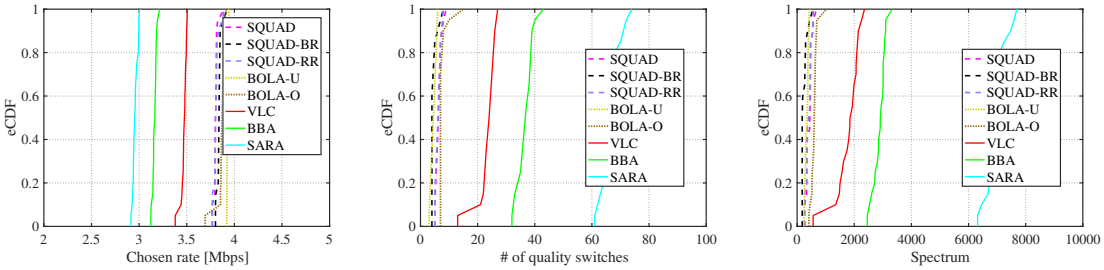
(a) Average bitrate. (b) # of quality switches. (c) Spectrum.

Figure 5.21: UDP-U cross traffic.



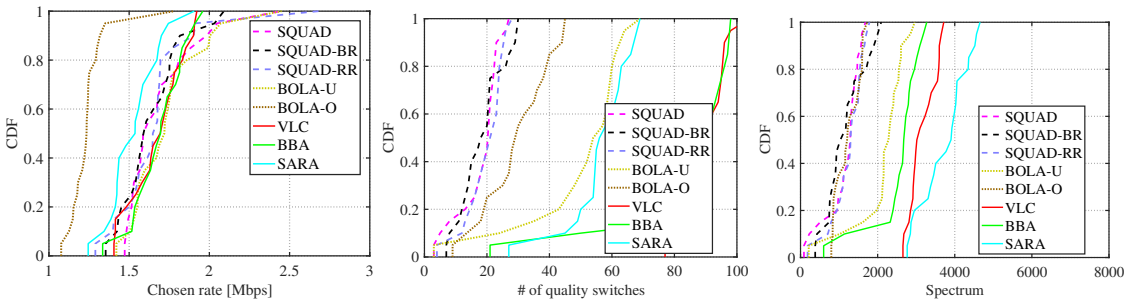
(a) Average bitrate. (b) # of quality switches. (c) Spectrum.

Figure 5.22: UDP-W cross traffic.



(a) Average bitrate. (b) # of quality switches. (c) Spectrum.

Figure 5.23: UDP ON-OFF cross traffic.



(a) Average bitrate. (b) # of quality switches. (c) Spectrum.

Figure 5.24: TCP cross traffic.

Concurrent Homogeneous 10 Streaming Sessions										
QoE Metric	SQUAD		SQUAD-BR		SQUAD-RR		BOLA-O		BOLA-U	
	avg.	SD	avg.	SD	avg.	SD	avg.	SD	avg.	SD
Quality bitrate [Mbps]	2.57	0.62	2.55	0.6	2.57	0.62	2.56	0.81	2.81	0.63
# of quality switches	14.08	3.97	13.77	4.04	13.67	3.90	13.85	6.44	47.82	17.21
Spectrum	1069.1	377.4	1099.6	421.9	1090.4	471.8	1132	423.3	3626	1320

Table 5.3: QoE Metrics for 10 concurrent homogeneous streaming sessions.

Concurrent clients: The goal of the second set of measurements performed in the GENI testbed is to investigate the fairness among multiple streams of the same DASH algorithm. We conducted this experiment by running 10 concurrent homogeneous DASH clients with *20 Mbps capacity* on the bottleneck link. The topology for this experiment is shown in Fig. 5.11, where $M = 10$. Due to the better performance of the BOLA and SQUAD algorithms in the previous set of experiments, we decided to run this set of experiments only with those two algorithm classes. In this experiment, each new stream starts with a *10 seconds delay offset*. We executed each experiment 20 times and show the average and standard deviation of the 10 clients in Table 5.3. The results show that BOLA-U achieves the highest average bitrate but also the highest number of quality switches. In contrast, BOLA-O and all three SQUAD algorithms achieve a slightly lower average bitrate in exchange for much less quality switches. Note that although BOLA-O achieves similar average bitrate

and number of quality switches, it has a considerably higher standard deviation for these two metrics. This indicates that SQUAD is able to achieve less variations among all the clients in terms of quality bitrate and quality switches. Note that on average all clients achieve a bitrate that is higher than 2Mbps. This is because the 10 second delay allows a subset of clients to temporarily get a higher bitrate than 2Mbps. E.g., client 1 can stream at the highest quality rate without any competing traffic for the first 10 seconds. Since this is a scenario for a highly congested bottleneck link, there is not much room for retransmissions. This is reflected by virtually no improvement of the three metrics in the case of SQUAD-BR and -RR.

Further, we use Jain's fairness index [53] to quantify the fairness of QoE among the clients, i.e., in terms of the quality bitrate, the number of quality switches, and the spectrum. An index value closer to 1 indicates a better fairness. Figure 5.25 shows that SQUAD achieves comparable fairness to BOLA in terms of bitrate and spectrum and achieves a better fairness in terms of the number of quality switches. Also, amongst the 20 runs, we show that the SQUAD algorithms can achieve slightly less fairness fluctuations in terms of standard deviation. In this case, BOLA-U achieved the highest fairness index (0.95) for playback bitrate, since all the clients are more aggressive in increasing the playback quality; yet it has a lower fairness in terms of quality switches and spectrum comparing to SQUAD. Also SQUAD achieved slightly less standard deviation comparing to BOLA. E.g., in the case of spectrum, SQUAD algorithms got standard deviation of 17%, which is lower comparing to BOLA (21%).

5.5.3 Internet Experiments

Besides evaluating SQUAD performance in controlled testbeds, we are also interested in its performance in the "wild". In this section, we use a set of experiments to take a comprehensive examination of SQUAD (SQUAD, SQUAD-RR and SQUAD-BR) as well as BOLA (BOLA-O and BOLA-U) algorithms, in the Internet environment. We use these

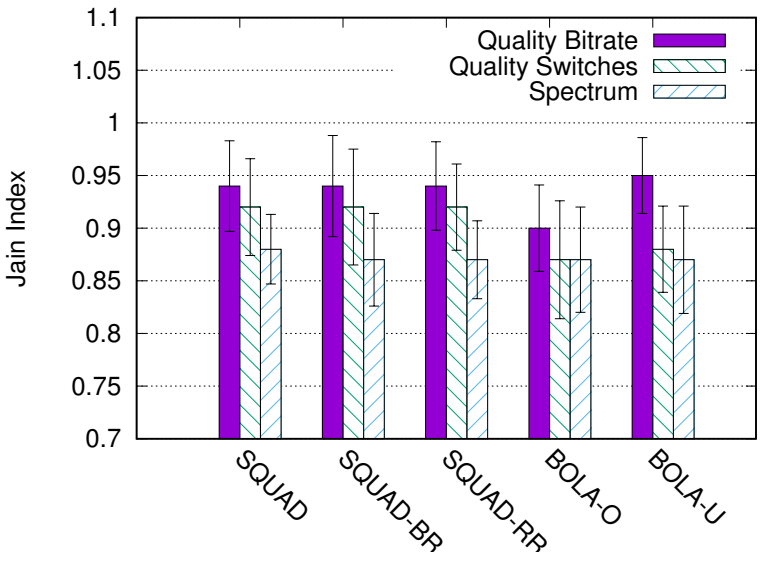


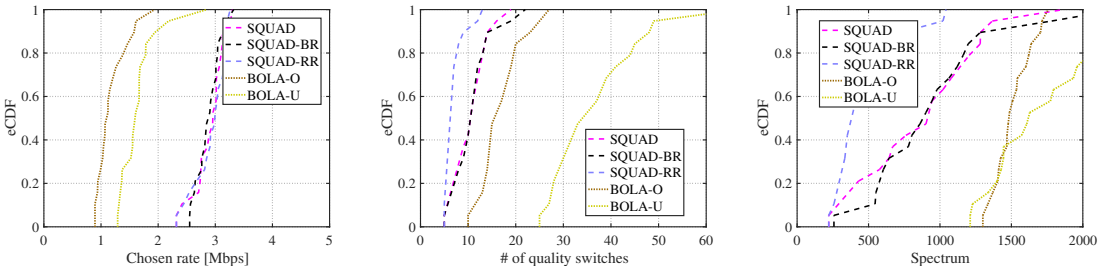
Figure 5.25: QoE metric fairness for 10 concurrent homogeneous streaming sessions.

two sets of algorithms because they provided the highest quantitative QoE in the GENI experiments. We make use of Amazon EC2 servers in Sydney and Sao Paulo to host the video repository. The client is located in a residential home in western Massachusetts of US. We measured a maximum bandwidth using a long-lived TCP flow to the EC2 servers in Sydney and Sao Paulo that amount to 3.2 Mbps and 2 Mbps, respectively. Results of the measurements are shown in Figures 5.26 and 5.27, as well as, in Table 5.4. We show that the SQUAD algorithms outperform BOLA. Interestingly we note the QoE improvement achieved by SQUAD-RR, which effectively retransmits segments that impact the number of quality switches, leading to an improved QoE in Internet-based streaming scenario.

Note that for all presented testbed and Internet measurements, the rebuffering time amounts to less than 1% of total playback time for all algorithms, except for the cases of aggregate TCP cross traffic in combination with SQUAD and BOLA algorithms. Due to the large amount of TCP cross traffic in the bottleneck link, SQUAD and BOLA have 3.3% and 2.7% average rebuffering time, respectively.

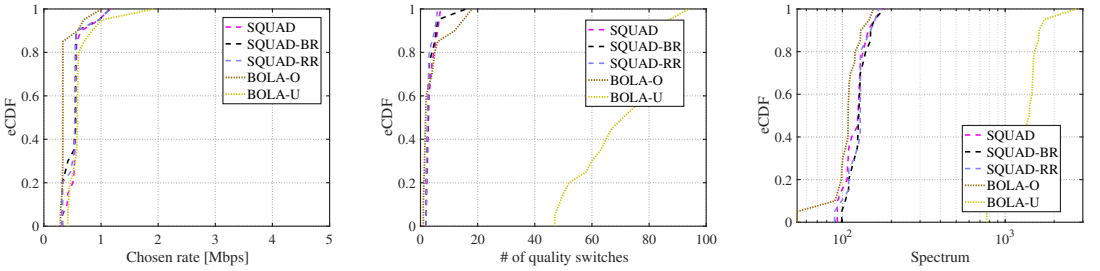
DASH Server: Sydney										
QoE Metric	SQUAD		SQUAD-BR		SQUAD-RR		BOLA-O		BOLA-U	
	avg.	SD	avg.	SD	avg.	SD	avg.	SD	avg.	SD
Quality bitrate [Mbps]	2.91	0.22	2.90	0.21	2.94	1.17	1.19	0.26	1.66	0.34
# of quality switches	11.6	3.2	10.6	3.9	7.2	2.1	17.3	12.71	37.7	9.7
Spectrum	995.4	376	835.7	458.0	472.3	218.4	1415	630.3	2679	721.4
DASH Server: Sao Paulo										
QoE Metric	SQUAD		SQUAD-BR		SQUAD-RR		BOLA-O		BOLA-U	
	avg.	SD	avg.	SD	avg.	SD	avg.	SD	avg.	SD
Quality bitrate [Mbps]	0.58	0.17	0.54	0.19	0.55	0.19	0.39	0.17	0.66	0.32
# of quality switches	4.0	1.44	3.9	2.99	3.25	1.13	4.5	4.6	63.2	19.15
Spectrum	105	28	140	39	106	44	102	25	1323	878

Table 5.4: QoE metrics for the Internet measurement campaign.



(a) Average bitrate. (b) # of quality switches. (c) Spectrum.

Figure 5.26: Internet measurements: US East Coast - Amazon EC2, Sydney, Australia.



(a) Average bitrate. (b) # of quality switches. (c) Spectrum.

Figure 5.27: Internet measurements: US East Coast - Amazon EC2, Sao Paulo, Brazil.

5.6 Summary

In this chapter, we provide a QoE tailored quality adaptation algorithm denoted SQUAD. A quick dissection of the DASH control loop over TCP shows the discrepancy of the available bandwidth estimation time scale between the quality adaptation algorithm and the underlying transport protocol. Bearing this in mind, we construct SQUAD such that it uses rate estimates on the appropriate time scales. SQUAD takes multiple QoE metrics into account, i.e., the average quality bitrate and most importantly its variation.

We test our player implementation against state-of-the-art quality adaptation algorithms of different objectives in a controlled network environment, as well as, in streaming experiments across the Internet. The experiments show that by sacrificing little or no average quality bitrate, SQUAD provides significantly better QoE. With an additional retransmission scheduling algorithm, we show that SQUAD is able to further minimize quality switching frequency and magnitude by retransmitting and replacing the segments that have already been filled into the client buffer.

CHAPTER 6

ENERGY EFFICIENT DESIGN FOR CLOUD-BASED ABR VIDEO TRANSCODING SERVICES

6.1 Introduction

As shown in recent studies, cloud based video streaming applications have become a significant contributor for energy consumption. In 2011, the year studied by [85], Americans streamed 3.2 billion hours of video, which consumed approximately 25 Peta Joules of energy (enough to power about 175,000 U.S. households for one year) and emitted 1.3 billion kilograms of CO_2. By today, these numbers have further increased given the increasing popularity of video streaming services. With the raising awareness of energy consumption, leading IT companies have been striving to power their infrastructure with green and renewable energy. Amazon claims that 40% of their operations are powered by renewable energy, while Microsoft has also committed to a goal of 44% renewable energy for its Azure data centers [81]. All of these trends clearly indicate the needs of reducing "dirty" energy usage as well as expensive energy supply from the grid.

Today's grid is grossly inefficient, while utilities begin to experiment with demand response programs, in most cases, utilities still balance supply and demand largely by only regulating supply, while granting consumers the freedom to *use as much power as they want, whenever they want.* This freedom imposes a steep price along multiple dimensions, resulting in wasted capital investments, high operational costs, and limited renewable penetration. The grid's inefficiencies have motivated recent "smart" grid initiatives to reduce peak demands and better handle intermittent renewables using demand-side management, which balances supply and demand, in part, by regulating electrical loads' energy usage,

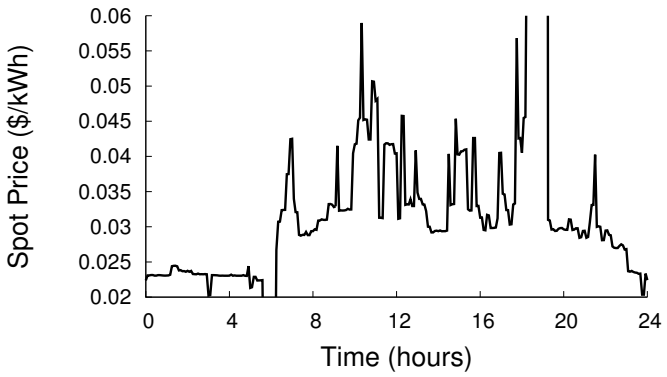


Figure 6.1: Electricity's real-time price fluctuates significantly every few minutes.

e.g., via real-time pricing or demand response. These initiatives offer consumers the potential for significantly lower costs, while also enabling consumers to use more green energy from renewable sources. As one example, Figure 6.1 plots electricity's real-time price (in New England's five-minute spot market) on October 5, 2013 from 12am to 11:59pm to highlight that it varies dramatically even over short time-scales. These fluctuations enable consumers to reduce costs by increasing energy usage when prices drop, and decreasing it when prices rise. In addition, the ability to adapt to power fluctuations also enables platforms to reduce their carbon emissions by increasing their reliance on local renewable energy sources, which generate energy intermittently based on environmental conditions.

In this chapter, we focus on improving the energy efficiency for transcoding cloud of ABR services. Our goal is to provide insights into designing energy management policies that effectively execute video transcoding tasks in the presence of time-varying, dynamic power constraints, and make use of least amount of energy supplied from the grid.

In this part of the dissertation, our contributions are the following:

- We model the video transcoding task scheduling problem, and design a series of power management policies for large scale, deadline-driven video transcoding jobs.
- We evaluate the performance of energy agile video transcoding services based on a small synthetic video dataset and a large scale video trace collected from Akamai's video CDN.

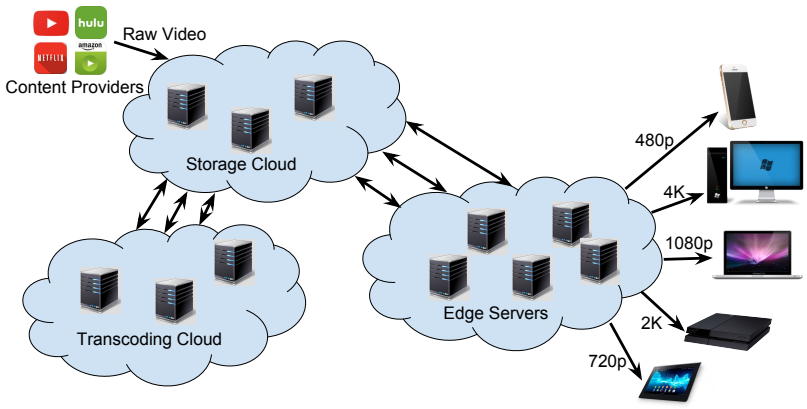


Figure 6.2: CDN and Online transcoding architecture.

We show that, our energy management policies can provide low rebuffering ratio (<0.5%), and can also significantly reduce the grid energy usage by 73-83%. As a result, by making efficient use of the low-price renewable energy sources, the total energy cost for completing transcoding tasks in the cloud can reduce by 14-28%.

6.2 Methodologies

In this section, we provide an overview of a cloud-based video transcoding service architecture. In addition, we present the energy-aware transcoding workload model, the power managing policies for the ABR video transcoding services, and the video data sets for evaluating the energy aware transcoding policies.

6.2.1 Cloud-based Real Time Transcoding

In order to ensure a smooth playback for ABR streaming, a new video needs to be transcoded into suitable resolutions, segment durations and bitrates before the user actually starts watching. E.g., Netflix transcodes each video 120 times in order to serve different user needs [82]. This approach guarantees that the videos are ready no matter what resolution, bitrate or segment duration is requested by the users. However, transcoding and storing such huge amount of videos has become challenging for content providers, due to

the ever-growing requirements for storage space and computational resources. A common alternative solution is to perform transcoding while the video is being streamed to the end user, commonly known as real time transcoding [92]. This approach greatly reduces storage space requirements, yet it introduces strict scheduling requirements, since the video segments must be transcoded and streamed before the client buffer drains.

The transcoding tasks are usually performed either by the content provider or by Content Delivery Networks (CDN). In this dissertation, we focus on transcoding performed by CDN clouds. As shown in Figure 6.2, the CDN transcoding architecture consists of three components. The *transcoding cloud* provides real time transcoding services to encode the raw video into target formats and bitrates. The *storage cloud* stores the raw videos published by video content providers, as well as the transcoded videos. The *Edge Servers* are distributed around the world, and located geographically close to the end users. By caching and prefetching the contents, the edge servers can provide users with a low delay, high throughput video streaming experience.

When a client starts to play a video, it initializes the playback by sending an HTTP request for video content with specific resolution and bitrate to the CDN. Then the following steps happen inside the CDN:

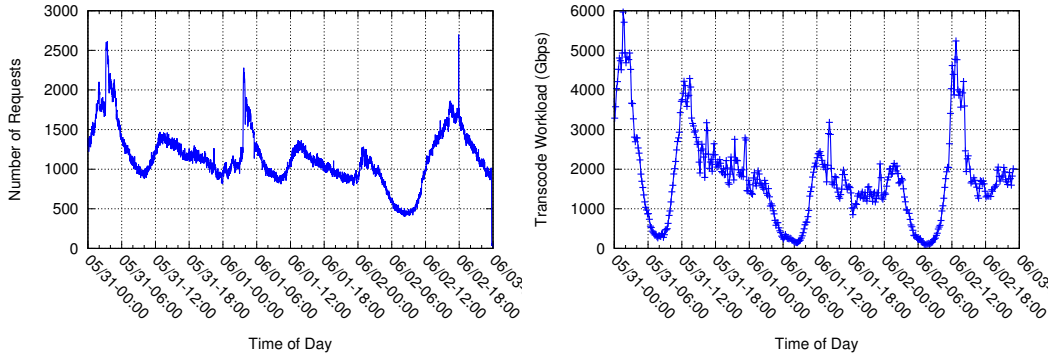
1. The CDN redirects the request to an edge server with shortest geographic distance.

2. If the requested segment, with specific bitrate and resolution, is available on the edge server, it is directly sent from the edge server to the client. Otherwise, the edge server downloads this segment from the storage cloud and forwards it to the client.

3. If the request is forwarded to the storage cloud, the CDN checks if the segment is available. If not, the storage cloud sends a transcoding request to the transcoding cloud, where the raw video is encoded into the requested resolution and bitrate, and sent back to the edge sever.

The interplay of CDN cloud and transcoding services provides tremendous improvement for cloud storage space, since the service providers do not need to pre-transcode and store all the videos in the cloud. Instead, the videos are only transcoded into desired quality levels for target video segments. The high capacity of servers and high transportation link bandwidth inside the CDN ensures that the videos can be transmitted to users on time and therefore provide a low buffering ratio and short video start up delay.

6.2.2 Energy Aware Transcoding Workload



(a) Number of requests. (b) Transcoding workload.

Figure 6.3: The transcoding work load in terms of number of transcoding requests (6.3a) and data rate (6.3b).

The cloud-based ABR transcoding applications have their unique characteristics. First, the ABR transcoding tasks are usually batch-like, CPU intensive, and with short-durations. Therefore, they are well suited for distributed computing clusters subject to power management, because the jobs are highly parallelizable, and impose less dependency on other computing nodes. Therefore, the nodes can easily switch between sleep and awake states as long as the current batch jobs are completed, and result files are transferred to the storage nodes. Also the ABR transcoding tasks are highly predictable by the client OS, network type, recommendation list, etc. In this case, we perform predictive transcoding in the later sections in order to reduce the transcoding workloads. Finally, the transcoding tasks are

strictly deadline-driven, and the workloads are highly variable according to the time-of-day. For example, we show the number of requests per unit time for a video request trace collected from the Akamai video CDN in Figure 6.3a. The corresponding transcoding workload is shown in Figure 6.3b. In order to ensure a satisfying user experiments, we utilize the grid energy in combination to the renewable energy to keep up with the varying energy requirements.

We model the video transcoding system as a series of batch workloads, each with request arrival time and required transcoding deadline. When transcoding jobs are being executed, they are distributed to N worker nodes, while the exact scheduling is subject to dynamic power constraints and grid energy prices. Since renewable energy is often intermittent and weather dependent, we use a combination of renewable energy and grid energy. Our goal is to finish executing the transcoding tasks such that specified job deadlines and operational goals (e.g., minimize grid energy usage and cost) can be satisfied. We aim to minimize the grid energy usage, since a large portion of the grid energy generation is formed by non-renewable sources. For example, as shown in the report [8] from the New England Independent System Operator (ISO), over 90% of the energy generation is from non-renewable energy sources. More formally, we aim to schedule the transcoding jobs within a set of task deadlines $TR(t)$, given a renewable power signal $P_r(t)$ and grid spot market power price signal $C_g(t)$, which dictates an average power cap over each interval $(t-\tau, t]$. Here, τ is the length of each interval, which we assume is dictated by the energy storage capacity.[1] In this work, we assume a τ value of 120 seconds.

We employ a set of power management policies to distribute the renewable power over the transcoding cluster. If the renewable energy generation is not sufficient to support all the transcoding requests, we purchase additional energy from the grid based on the spot market price and transcoding workload. We divide the energy price into 3 regions: *high, medium*,

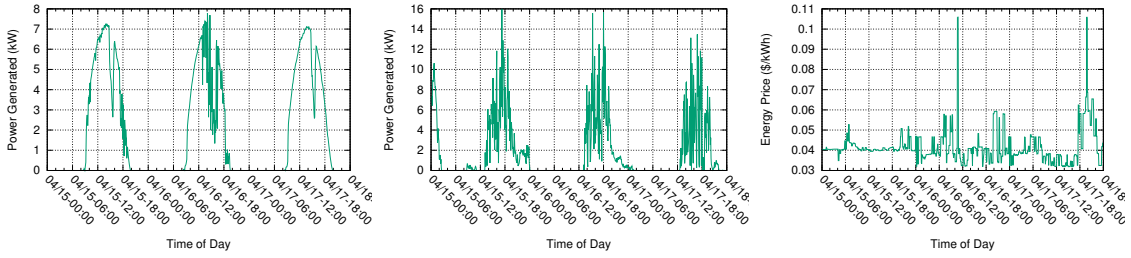
[1] In the future, grid energy usage can be further reduced by using larger energy storeage devices once energy storage becomes more efficient.

and *low*. We use a simple policy to decide how much grid energy should be purchased in addition to the renewable energy:

- When the energy price is high, the total power should be able to execute at least 85% of all requests.
- When the energy price is medium, the total power should be able to support at least 95% of all transcoding tasks.
- When the energy price is low, we purchase the power that is enough to support all the transcoding tasks.

In this energy trace, as we show in Figure 6.4c, the high prices only lasts for short periods (<7%). With the help of predictive transcoding, as we show in Section 6.3.3, the overall transcoding performance seldom gets compromised.

A set of optimization and prediction mechanisms for deciding the grid energy purchasing may further improve the system performance, we leave the design of such mechanisms to the future work.



(a) Solar power. (b) Wind power. (c) Spot market power.

Figure 6.4: The solar (6.4a) and wind (6.4b) power and the real-time electricity prices in the five-minute spot market (6.4c).

6.2.3 Power Capping Mechanisms

Given an average power budget for each interval τ, we must distribute the available power among N nodes that are executing a transcoding task. We leverage existing node

power capping mechanisms, which ensure a node's power usage does not exceed a set threshold, to enforce a given distribution of power. Power capping may be either active or inactive. *Active power capping* uses Dynamic Voltage and Frequency Scaling (DVFS) or C-state throttling, which rapidly toggles processors between low-power idle C-states, to cap power without deactivating a node. The advantage of active power capping is that it keeps nodes active (albeit at a degraded performance level), is transparent to application software, is highly responsive (as power cap changes occur near instantly), and imposes a low overhead to transition power states. However, the primary disadvantage of active power capping is that it typically only targets CPU power, which accounts for only a fraction of node power usage, and thus its dynamic power range is limited. In practice, active power capping is typically only able to lower a node's power usage to at most 50% of its peak power [22].

In contrast, *inactive power capping* puts nodes to inactive state. While inactive power capping reduces a node's power usage to near zero, it incurs a high temporal transition overhead, varying from tens of seconds to minutes. The temporal overhead also wastes energy, since the node consumes energy while transitioning, but performs no computation. Thus, ABR video transcoding clusters are well suited for inactive power capping, since the task durations are typically on the level of segments (2-10 seconds), and transcoding tasks among segments are independent from each other. In our experiments, we assume an inactive power capping delay of 60 seconds. Due to its high overhead, inactive power capping is only used when capping the power of clusters, since it enables a wider dynamic power range than when only using active power capping [91].

6.2.4 Power Management Policies

In this section, we make use of the power capping mechanisms to control the video transcoding. We first propose policies that focus on *rigid transcoding applications*, which only rely on active power capping. This is because rigid applications cannot adjust the num-

ber of nodes being used while the transcoding task is running. While active power capping affects performance, it is transparent to the application, which enables rigid applications to make use of it. Since inactive power capping deactivates nodes, it has the effect of periodically altering the number of nodes an application is using; rigid applications cannot handle such reductions, as they will be perceived as failures.

Therefore, we introduce a simple *balanced* power management policy, which can be suitable for rigid cloud transcoding applications. The balanced policy equally distributes power among the nodes, such that each of the N nodes' active power cap is set to i.e., $P_{cap} = P_{available}(t)/N$ at time t for each node. In this way, the available energy is given to as many nodes as possible, which can be useful for scaling up the transcoding tasks when the amount of available power is high.

In contrast to the balanced policy, deactivating nodes using inactive power capping can be beneficial in reducing the fraction of the available power that contributes to an active node's idle power. In addition to actively capping server power consumption by limiting the maximum server CPU capacity, these policies stretch and contract an *elastic transcoding application* while it is running by activating and deactivating nodes to maintain a platform-wide power cap. For example, since each node's idle power is roughly 50% peak power, simply activating a node without doing any useful work consumes a significant amount of power. This portion of idle power effectively represents wasted energy that is due to powering non-energy-proportional components, such as disk and memory. Here, we define this portion of idle power consumption as *overhead power*, and In contrast, the remaining power usage represents *effective power*, since only the effective power contributes to actual transcoding. With regard to the idle power, the balanced policy incurs high idle power across all nodes, since more nodes are active regardless of the available power. Thus, we define a *greedy* power management policy that concentrates available power to the least number of active nodes, i.e., $N_{active} = \lceil P_{available}/P_{max} \rceil$. The last node will, therefore,

receive the remaining power, if not the full power. The greedy allocation, which uses inactive power capping, can achieve higher utilization in terms of effective power.

In addition to the greedy policy, to further reduce the idle power, we define an *agile* policy that determines the optimal subset of active nodes. To do this, we compute the least idle power for each possible set of active nodes, from 1 to N, based on the available power. Once these nodes are active, the agile policy then *equally* distributes the available power to the set of active nodes upon needs.

For each of the above three *static* power policies, we introduce a *dynamic* variation that also uses active power capping, for which we continuously redistribute the available power based on node utilization. In contrast, a static variation does not change power capping after fixed amount of power is allocated.

Our dynamic policies measure node CPU utilization and power per second, and adjust active power caps based on node utilization levels. In particular, we reduce the power cap for nodes that are being idle ($< 95\%$ CPU utilization), and redistribute the power to the nodes that are operating above 95% CPU utilization [97]. In essence, our dynamic power management policy reduces wasted power by taking power away from nodes that are being idle, and reallocating the power to the nodes that are executing transcoding tasks.

Table 6.1 summarizes the design space for our energy-agile policies. The basic idea for our power policies is to determine the optimal energy reallocation based on an application's characteristics, e.g., rigid vs. elastic, and the available power capping mechanisms. The balanced policy only utilizes active power capping by limiting maximum CPU capacity and therefore is suitable for rigid applications. With elastic applications, greedy and agile policies can make use of both active and inactive power capping by determining the optimal subset of nodes that minimizes overhead power. For all three energy management policies, we have two variations: the static variation, i.e., keep power allocation unchanged once an application starts, and the dynamic variation, i.e., keep tracking CPU utilization and reallocate power cap.

Metric	Balanced	Greedy	Agile
Power Capping	Active	Active & Inactive	Active & Inactive
Best Application	Rigid	Elastic	Elastic

Table 6.1: Summary of design space for energy-agile policies.

Resolution	Bitrates		
480p	1 Mbps	2 Mbps	-
720p	1 Mbps	2 Mbps	4 Mbps
1080p	2 Mbps	4 Mbps	8 Mbps
1440p (2K)	4 Mbps	8 Mbps	16 Mbps
2160p (4K)	8 Mbps	16 Mbps	32 Mbps

Table 6.2: Summary of transcoded resolutions and bitrates.

6.2.5 Video Data Sets

In order to evaluate the performance of energy-aware transcoding, we make use of a mini video data set and a large scale data set collected from Akamai's CDN. For the mini video set, we use a set of 50 demo videos, each with 1 minute length provided by Harmonic [6]. We transcode each short video into 5 resolutions, 14 bitrate layers, 6-second segment duration. The resolutions and bitrates are summarized in Table 6.2.

For the second data set, we make use of an extensive, anonymized log collected from Akamai's video CDN. Akamai [73] is the world's largest CDN provider that delivers 15-30% of global Internet traffic. Akamai's CDN contains over 150,000 edge servers distributed in 90+ countries and 1200 ISPs around the world. In this video log, user video requests from across Akamai's global CDN over a 3-day period in June 2014 were collected. The ABR streaming traffic in this log contains 5 million video sessions originating from over 200,000 unique clients who were served by 1294 video servers around the world.

6.3 Experimental Evaluation

In this section, we present the evaluation for our transcoding power management policies. We first show the results for transcoding with the mini video dataset with both un-

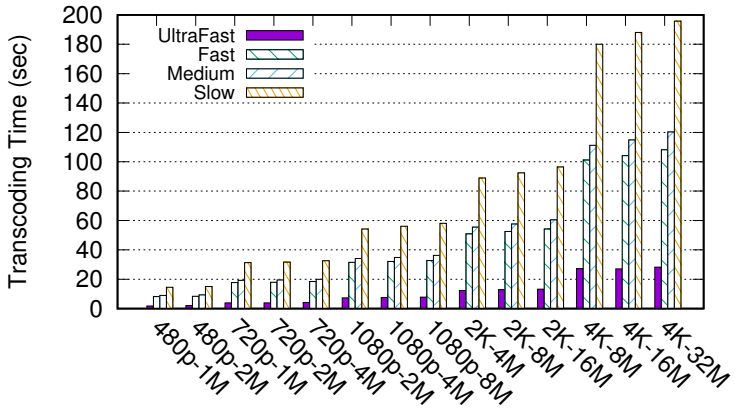


Figure 6.5: Transcoding time for 1 minute video with different resolutions and bitrates.

limited power and renewable power. Then we show the transcoding performance with the Akamai CDN trace described in Section 6.2.5.

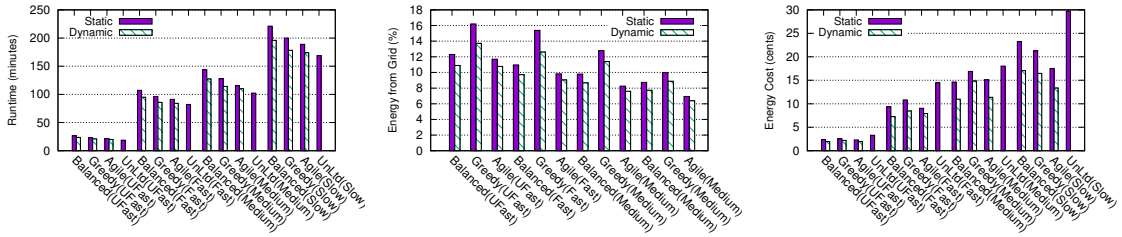
6.3.1 Transcoding with Unlimited Energy

In this section, we show the transcoding performance when operating under unlimited power as a baseline for future comparisons. Since we target real time transcoding in this work, we show the average transcoding time compared to the actual video length for different video resolutions and bitrates shown in Table 6.2. We use this transcoding runtime in the simulation of our large scale transcoding system. For the transcoding time analysis, we use the FFmpeg tool [4] to divide the videos into segments and x264 encoder [16] to transcode the video segments into different bitrates.

This experiment was performed on the CloudLab testbed [80] with a single Intel-based Dell R720 server containing 32 cores and 64GB memory. We transcode each 1-minute video into 5 resolutions, 14 quality layers varying from 480p, 1 Mbps to 2160p, 32 Mbps, with different encoding speed options (slow, medium, fast and ultrafast).

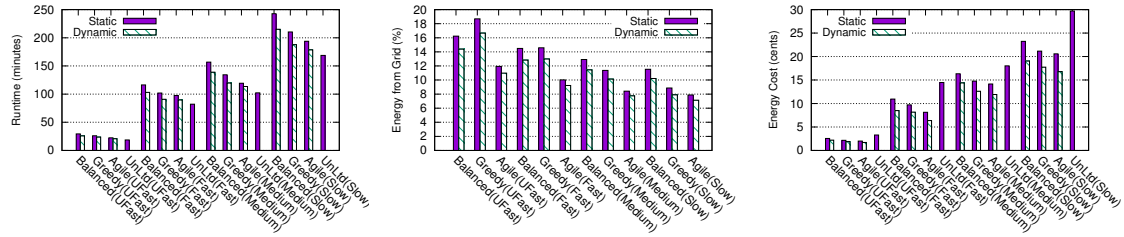
Figure 6.5 shows the average time for transcoding a one-minute video. The time taken for different bitrates but same resolution is relatively stable, while scaling up the resolu-

tion significantly increases the video transcoding time. In the case of medium transcoding speed, videos can be transcoded in real time (<60 seconds) up to a 2K resolution and a 16 Mbps bitrate. We use the default medium speed for future transcoding analysis in this work, because its runtime is close to fast, and it provides better video quality [12].



(a) Runtime. (b) Percentage. (c) Energy cost.

Figure 6.6: Transcoding with solar energy: runtime (6.6a), percentage of grid power usage (6.6b) and energy cost (6.6c)).



(a) Runtime. (b) Percentage. (c) Energy cost.

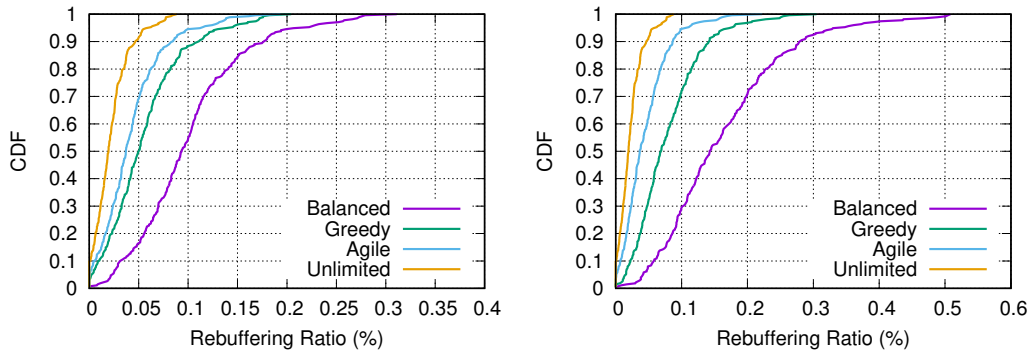
Figure 6.7: Transcoding with wind energy: runtime (6.7a), percentage of grid power usage (6.7b) and energy cost (6.7c)).

6.3.2 Transcoding with Renewable Energy

In this section, we show the transcoding performance with solar and wind renewable energy sources in Figure 6.6 and Figure 6.7, respectively. In this experiment, we make use of 16 Dell R720 servers in the CloudLab testbed. We record the total runtime for each of the power management policies and transcoding speeds.

Our experiments utilize power signals from real solar panel and wind turbine deployments at UMass Amherst, in combination with the grid power from the wholesale electricity market. For the renewable power signals, we select a representative 3-day period from April 15, 2016 12 AM to April 18, 2016 11:59 PM, as shown in Figure 6.4a and Figure 6.4b. For grid power, we use the five-minute spot price (shown in Figure 6.4c) from the New England Independent System Operator (ISO) for the same three-day period. To make a fair comparison among the power signals, we normalize the solar and wind power such that they have the same average power.

We show the total transcoding runtime, percentage of grid power usage, and grid energy cost for solar energy source and wind energy source in Figures 6.6 and 6.7, respectively. We estimate the renewable energy cost using the data from Lazard cost of energy analysis report [10]. The average cost is $43/MWh for solar and $31/MWh for wind. As can be observed, the agile policy yields the lowest runtime among the renewable power policies. This is because the agile policy is able to iterate through the power allocation options and find the optimal allocation strategy. The dynamic policies can further reduce the runtime and energy consumption by 7-19% compared to static policies. Our power policies enables the usage of renewable energy sources with the trade-off of runtime increase of 9-22% compared to transcoding with unlimited power. However, the utilization of renewable energy can significantly reduce the grid energy usage from the electricity suppliers, and therefore reduce the overall energy cost. As shown in Figure 6.6b and 6.7b, the transcoding process only makes use of 6-19% of power from the electricity grid. As shown in Figure 6.6c and 6.7c, since the renewable energy price is lower than spot market energy price, the utilization of renewable energy introduces 17%-49% energy cost reduction compared to transcoding with unlimited grid power.



(a) Solar. (b) Wind.

Figure 6.8: The rebuffering ratio with different power management policies.

6.3.3 Energy Aware Transcoding for CDNs

In this section, we simulate the power management policies based on the Akamai video CDN trace. We employ Earliest Deadline First (EDF) [33] for scheduling video requests. Based on the transcoding requests, we use our power management policies to control the transcoding job execution. In the simulation, we start with an empty video storage in the storage cloud. However, this introduces much additional transcoding workload on the first day, which is not typical in a real system. Therefore, we follow the methodology from [60], and leave the first day for warming up the system and prepare for the next two days' transcoding tasks. Since the solar and wind generation facility is relatively small comparing to the scale of the Akamai transcoding cloud, we normalize the renewable energy (solar and wind) signal shown in Figure 6.4 such that the peak power production can support full workload of the Akamai transcoding data center.

We measure the actual performance of the energy-aware transcoding policies in terms of the *rebuffering ratio*, which represents the length of rebuffering state at the client as percentage of the total playback. In our case, the playback rebuffering is caused by two factors: *prediction error* and *power insufficiency*. In the prediction phase, we use the Markov model based network type-OS tuple prediction policy from [60] to generate transcoding tasks. The prediction error occurs in case of a misprediction and due to that error requested video seg-

Energy source		Balanced	Greedy	Agile
Solar	Usage (kWh)	2775	2501	1921
	Cost ($)	525.2	512.1	488.1
Wind	Usage (kWh)	3021	2668	2421
	Cost ($)	495.6	455.8	436.9
Unlimited	Usage (kWh)	12115		
	Cost ($)	614.9		

Table 6.3: Comparison of total transcoding grid energy usage (kWh) and energy cost ($).

ments are not pre-transcoded before the deadline. We determine this error by comparing the predicted video requests and the actual requests from Akamai's CDN trace. In the case of insufficient power, the current power budget E_{budget} is not sufficient to power the minimum number of servers required to execute all transcoding requests on time. Therefore, the rebuferring ratio is determined by:

$$T_{buffer} = N_{seg} \times (E_{pred} + E_{budget}) \times T_{transcode}, \tag{6.1}$$

where $T_{transcode}$ denotes the required transcoding time that is determined by requested bitrates and resolutions.

Figure 6.8 shows the Cumulative Distribution Function (CDF) of rebuffering ratio for transcoding with solar and wind renewable power in comparison with the case of unlimited power. Since we have known that dynamic policies always provide better performance comparing to static policies [97], we only show the performance of dynamic policies in this section. Most of the policies work better in the case of solar energy due to its higher stability. The agile policy performs significantly better than other policies. It provides an average rebuffering ratio of 0.041% and 0.052% under solar and wind renewable energy sources, which is very close to the unlimited power case with average rebuffering ratio of 0.035%. In addition, all the policies can achieve a rebuffering ratio $<$0.5%, which is known to be very low in video streaming scenarios [25].

In addition to the rebuffering ratio, we show the total energy consumption and cost for each of the policies in Table 6.3. As can be observed, the power management policies can greatly reduce the grid energy consumption and therefore lead to a significant energy cost reduction. The renewable energy policies can reduce grid energy usage by 73-83%, which leads to electricity cost reductions of 14-28% compared to unlimited grid power.

6.4 Summary

In this chapter, we look into real time ABR video transcoding on large scale, renewable energy powered data centers in combination with grid energy sources. We show that transcoding services are suitable to be conducted on green clusters. With the effective use of power management policies, the grid energy usage can be reduced by 73-83%, and the corresponding energy cost can be reduced by 14-28% while sacrificing little to none of the user experience.

CHAPTER 7

EXTENSION OF ENERGY MANAGEMENT FOR PARALLEL CLOUD APPLICATIONS

7.1 Introduction

In Chapter 6, we show that, for cloud based transcoding services, through a set of power management policies, the energy consumption can be significantly reduced by either turning idle nodes off, or limiting the CPU capacity of idle nodes. In this chapter, we generalize the power management policies to parallel cloud applications.

As outlined below, cloud based applications are well-suited to exploit such demand-side optimizations for four reasons.

- **Sophisticated Power Management**. Cloud platforms already include advanced, remotely programmable power management mechanisms, making them capable of rapidly and precisely controlling their energy usage over a wide dynamic range.

- **Delay-Tolerant Workloads**. Many cloud workloads consist of non-interactive batch jobs that are tolerant to delays in execution, providing them the flexibility to adjust their energy usage over time [103]. The price elasticity of demand is higher for these workloads than many household and industrial loads, which are often interactive and not highly responsive to price fluctuations.

- **Large Energy Consumers**. The power requirements of future cloud platforms ($>$20MW) will position them as some of largest industrial energy consumers with the highest electricity costs. As a result, they will have the most to gain from adapting their energy usage in response to changing grid conditions.

- **Rapid Growth Sector**. Despite energy-efficiency improvements, the power demands of cloud data centers continue to rise, increasing by an estimated 56% from 2005-2010 and accounting for 1.7-2.2% of U.S. electricity usage [57], with usage expected to double every five years [93]. Thus, regulating their demand holds significant potential to improve grid efficiency, and reduce costs.

Thus, in this chapter, our goal is to provide insights into utilizing the energy management policies to efficiently execute parallel tasks. Our hypothesis is that a policy that combines active power capping—to continuously reallocate available power among nodes based on their real-time utilization—with inactive power capping—to minimize the aggregate power consumption overhead—outperforms other policies in the design space. In evaluating our hypothesis, we make the following contributions in this chapter.

- **Green Energy Challenges.** We outline the challenges associated with optimizing parallel applications for green energy sources with variable power. In particular, we focus on parallel applications that exhibit cross-node dependencies during execution, since these applications complicate both active and inactive power capping. We then present a model for a representative parallel application that leverages MPI (Message Passing Interface) as a reference for presenting our energy management policies.

- **Dynamic Energy Management Policies.** We extend the dynamic energy management policies from Chapter 6 to maximize parallel application performance subject to variable power constraints.

- **Implementation and Evaluation.** We implement our policies on a real, 65-node prototype cluster, and evaluate their performance using multiple parallel applications. Our results demonstrate the importance of designing energy agile management policies for variable power. For example, we show that the Graph500 benchmark requires 17% more time and 9% more energy to complete when power varies based on

real-time electricity prices than the case with unlimited power. However, since real-time prices are lower than fixed prices, the total cost of our best energy management policy when using real-time prices is 67% less than when using unlimited power.

Our results show that, for rigid jobs, and representative solar/wind power signals, a dynamic policy, which continuously reallocates power based on node utilization, outperforms a static policy, which only reallocates power when available power changes, by 55%. We also show that an elastic task that uses inactive power capping outperforms an equivalent rigid task by 41%, which demonstrates the importance of elasticity to energy-agile design.

7.2 System Architecture

In this section, we present our problem statement, as well as provide background information on power capping techniques and our general model for parallel tasks.

7.2.1 Problem Statement

In this chapter, our work assumes a parallel application that is subject to dynamic power constraints, which may arise from either the use of renewable energy sources or participation in a utility demand response program. Our goal is to finish executing a parallel application as fast as possible given these dynamic constraints. More formally, we aim to minimize a parallel task's running time given a power signal $P(t)$, which dictates an average power cap over each interval $(t - \tau, t]$. Here, τ is the length of each interval, which we assume is dictated by the amount of energy storage capacity available. Since energy storage is expensive to install and maintain, smaller values of τ are better. The application defines its power constraint for each interval $(t, t + \tau]$ based on the energy it stored during the previous interval $(t - \tau, t]$. Thus, the power constraint at each interval is known at the beginning of the interval.

7.2.2 Parallel Task Model

We utilize the energy management policies presented in Chapter 6 to determine how to distribute the available power for each interval $[t - \tau, t)$. The goal of our policies is to minimize the running time of a parallel task. Unlike the the cloud video transcoding tasks introduced in Chapter 6, the parallel cloud application workloads are usually interactive, i.e., the tasks may exhibit a wide variety of communication patterns and inter-node dependencies during their execution. Below, we illustrate key elements of our energy management policies using a simple representative example—a parallel breadth first search (P-BFS) algorithm—which serves as a building block for designing policies for tasks with more complex communication patterns.

P-BFS and other graph algorithms are a frequent subproblem in a variety of data-intensive cloud analytics applications, which is the primary reason P-BFS was chosen as the foundation of the recently introduced Graph500 benchmark [5]. In practice, P-BFSs are often massive in scope, searching graphs with tens of billions of edges and vertices on platforms with tens of thousands of cores [32]. Importantly, since the classic P-BFS [76] implementation is "level-synchronous," it requires all-to-all communication among nodes at each level of the graph to determine whether a remote vertex has already been visited, i.e., by transmitting all remote edges on each node to the node that owns them, or transmitting all remote edges to a master node. Most non-embarrassingly parallel tasks will include similar types of barriers to synchronize their operation between phases.

While all-to-all synchronization barriers, which represent a dependency between each pair of nodes, are already the primary bottleneck for a P-BFS, consider the implications when using variable power. When using active power capping, reducing the power cap and performance level of one node will affect the performance of all other nodes, since to complete each level-synchronous step at each level of the graph, each node must communicate with all other nodes. However, setting all active power caps to the same value for each node may not guarantee the same per-node progress, since each nodes' progress depends on the

portion of the input graph it is given at each level. As a result, each node's utilization and power will vary within each phase. Inactive power capping imposes a similar constraint, since a parallel task cannot continue until every node completes its work. Thus, tasks may be finished with different speeds which creates synchronization barriers, and the overall progress is dictated by the slowest node to complete any phase.

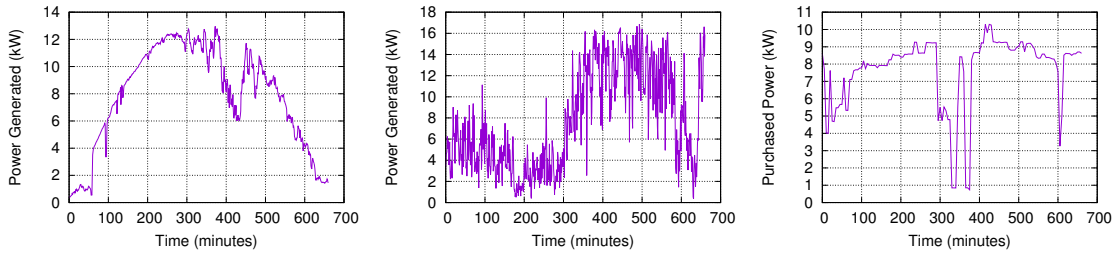
Thus, the presence of both inter-node dependencies and unpredictable performance across nodes (due to differences in the partition of input data they receive) poses challenges when power varies. Any data-intensive application, such as P-BFS, which consists of a series of synchronization barriers will exhibit such inter-node dependencies. Of course, optimizing embarrassingly parallel tasks, where each node operates at 100% utilization and exhibits no inter-node dependencies, is straightforward, since any policy that uses all the available energy will yield the same performance. We also note that the all-to-all barriers in a P-BFS are a particularly challenging type of communication pattern, since they require synchronization across every node. Thus, our energy-agile policies may be extended to a broader range of parallel computing applications with more complex communication patterns. In fact, patterns that do not require all-to-all synchronization should permit more flexibility and greater performance improvements when optimizing for energy-agility.

7.3 Implementation

In this section, we discuss our reference cloud applications, and the input power signal that we use for further evaluations.

7.3.1 Reference Applications

We prototype our policies on a real cloud platform with all dedicated servers, and evaluate the policies using three parallel, MPI based applications: Graph500, WRF and Jacobi, with a single process on each node. The Graph500 benchmark [5] implements a P-BFS and forms the basis for a large set of data-intensive parallel applications. We use Graph500



(a) Solar energy (b) Wind energy (c) Spot price based

Figure 7.1: The solar (7.1a) and wind power (7.1b) generated over a day, as well as a power signal based on using a fixed budget to purchase electricity at real-time prices in the five-minute spot market (7.1c).

as the main benchmark in this work, since it is a widely-accepted platform benchmark for parallel application performance. We make use of a moderate input graph scale of 26, i.e., the input graph contains 2^{26} vertices. The Weather Research and Forecasting (WRF) application [15] is a mesoscale numerical weather prediction application. In our prototype, we make use of WRF with an input scale of 1km$\times$1km, and time step of 60 seconds. The Jacobi algorithm is a well-known numerical method for solving linear algebraic systems of *n* equations with *n* unknowns. We make use of an MPI based parallel Jacobi implementation with random input matrix size $n = 10,000$. The WRF power consumption behavior is relatively stable; while the communication pattern of Jacobi is similar to Graph500 with synchronization barriers that make power consumption more variable. By default, the three MPI applications are rigid, i.e., we cannot adjust the number of worker nodes during its execution. However, we also experiment with an elastic variant of Graph500 by applying a method proposed by Raveendran et al. [79] to transform a rigid parallel task into an elastic one. In this case, the elastic MPI applications add a *decision layer* to decide the amount of jobs to be performed for each iteration, and then collect current result and automatically redistribute the tasks to the new set of nodes for the next iteration.

7.3.2 Input Power Signal

Our experiments utilize power signals from real solar and wind deployments, as well as signals based on real power prices from the wholesale electricity market. For each power signal, we select a representative day-long period with average power readings every minute.[1] The power signal for the solar and wind traces are collected from the solar panel and wind turbine deployments located in western Massachusetts, United States, and are shown in Figure 7.1a and Figure 7.1b. For the energy price trace, we use the five-minute spot price from the New England Independent System Operator (ISO) on October 1, 2014 from 12am to 11:59pm. In this case, we assume our system has a fixed budget for purchasing energy, such that during a low price period it may purchase more power, and during a high price period it must purchase less. The resulting power signal is shown in Figure 7.1c. Finally, to make a fair comparison among different power signals, we normalized all three traces such that they have the same average power.

7.4 Performance Evaluation

We evaluate our prototype policies in the CloudLab testbed [80] using two types of servers: the ARM-based HP Moonshot servers with 8 cores and 64GB memory, which are low-power nodes designed for energy-efficiency, and the Intel-based Dell R720 servers with 32 cores and 64GB memory, which are high-power nodes designed for maximum performance. The low-power cluster consists of 65 nodes, while the high-power cluster consists of 6 nodes. The servers in each of the two clusters are equipped with IPMI management card in order to monitor and control the real time power consumption of each node. We use MPICH-3.2 [11] for all three MPI based applications. We use the CloudLab public interface with a star topology. All the nodes are connected to the main switch via a 10 Gbps bandwidth link. Our 6-node small scale experiments are conducted on the Mas-

[1]For solar power, we selected an early Fall day, September 28, 2014 from 12am to 11:59pm. For wind power, we selected a typical spring day, April 26, 2014 from 12am to 11:59pm.

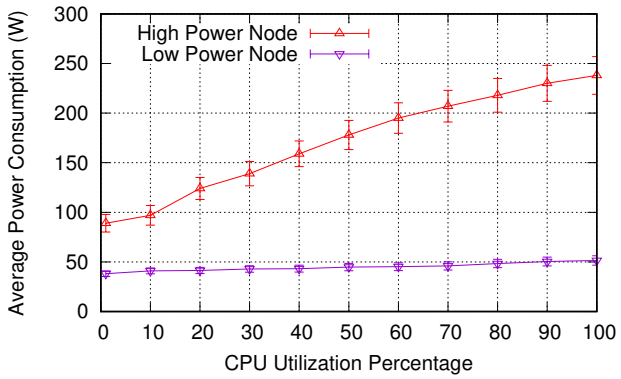
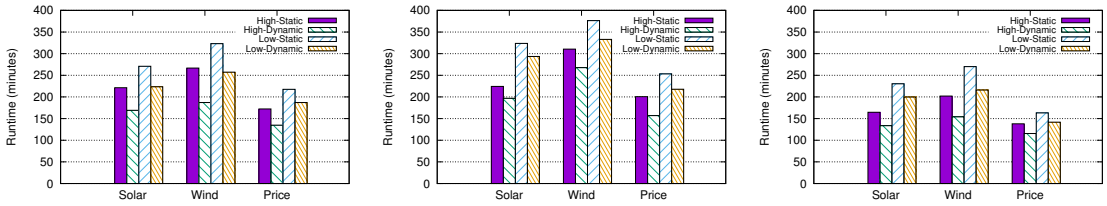


Figure 7.2: Power usage at different CPU load levels.

sachusetts Green High Performance Computing Center (MGHPCC) [75], where 94.3% of the electricity is generated from carbon-free sources including hydro-electric, nuclear and solar. For the Utah CloudLab cluster, the energy generation mix is unknown to us.

7.4.1 Node Power Usage

To determine the correlation of CPU load and power consumption, we perform a CPU stress experiment on a single node. In the future experiments, we collect the real time power readings from each node, and enforce the active power capping by limiting the CPU frequency on the corresponding nodes. For each CPU level, we add synthetic workload, and measure the CPU utilization over 10 minutes period. Figure 7.2 shows the dynamic power range of the two types of servers, where the x axis is the average CPU utilization across all cores, and the y axis shows the average and standard deviation for power consumption. As shown, the high-power nodes have a significantly greater dynamic power range. The idle power usage is 85W and the peak power usage is 235W. Thus, the dynamic power range is 150W/63% of peak power. The active power range decreases to 14W/21% in the case of the low-power nodes, which gives our power policies less room to reallocate the power.



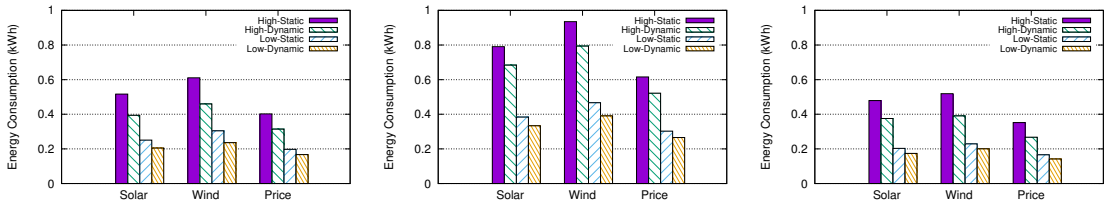
(a) Solar energy (b) Wind energy (c) Spot price based

Figure 7.3: Runtime of rigid Graph500 (7.3a), WRF (7.3b) and Jacobi (7.3c) for solar, wind and spot price-based power signals using both the dynamic policy and static balanced policy that employ active power capping.

7.4.2 Rigid Parallel Applications

In this section, we evaluate the performance of our power management policies for rigid applications, which only employs active power capping, with solar, wind, and price-based power signals. We show the runtime and total energy consumption in both the low-power cluster and the high-power cluster. To make them comparable, we utilize 6 nodes on each cluster, i.e., 1 master node and 5 worker nodes.

Figure 7.3 compares the runtime between static-balanced, which sets the power cap at the beginning of each interval based only on available power, and dynamic-balanced policy, which continuously shifts power based on which nodes are most effectively using it. All three applications show similar behavior. In the low-power cluster, the dynamic balanced policy reduces the runtime by up to 17% for solar, 21% for wind, and 14% for price-based power traces, compared to the static balanced policy. Likewise, in the high-power cluster, the dynamic balanced policy reduces the runtime by 23%, 27% and 20% for the three power signals, respectively. Similarly, Figure 7.4 shows Graph500 consumes less overall energy when using a dynamic balanced policy. All three applications show a similar runtime and energy consumption behavior. The runtime of the WRF application is slightly longer than the other two, which imposes a higher total energy consumption.



(a) (b) (c)

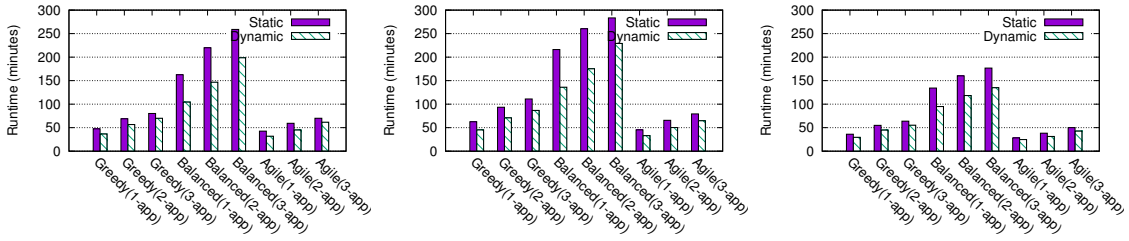
Figure 7.4: Energy consumption of rigid Graph500 (7.4a), WRF (7.4b) and Jacobi (7.4c) for solar, wind and spot price-based power signals using both the dynamic and static policies.

This experiment demonstrates the effectiveness of fine-grained power capping adjustment. By continuously reallocating power based on node utilization via active power capping, the application is able to make better use of the available power and improve its performance. The high-power cluster shows more improvement for two reasons: its servers *i)* exhibit a higher power variance between nodes, which provides better opportunity to adjust the power, and *ii)* have a wider active power range, enabling the power policies to reallocate more power among nodes.

Results: *Our energy management policies significantly decrease both the runtime and the energy consumption and improve the performance of rigid applications by continuously reallocating power (using active power capping) to the nodes that can use it most effectively.*

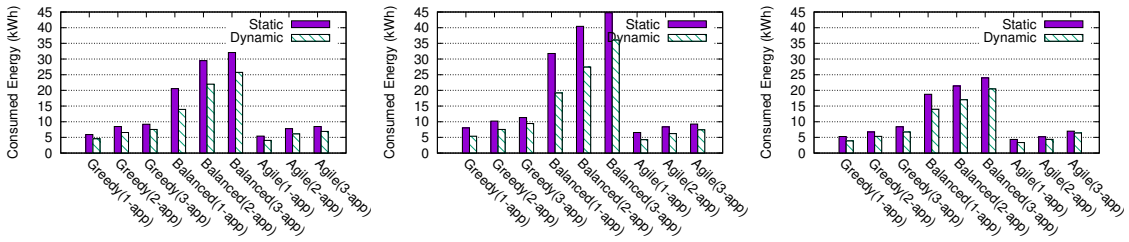
7.4.3 Elastic Parallel Applications

The results above are limited to rigid parallel applications that cannot handle activating and deactivating nodes while the application is running. In this section, we demonstrate the effectiveness of our energy management policies when employing both active and inactive power capping for elastic applications. To maximize the flexibility, we make use of all the 65 nodes in the cluster with low-power nodes. The experiments again use solar, wind, and price-based power signals from Figure 7.1. In addition, we run multiple parallel



(a) (b) (c)

Figure 7.5: Runtime of elastic Graph500 for greedy, balanced and agile policies with solar (7.5a), wind (7.5b), and spot price-based power signals (7.5c) for cases with multiple applications.



(a) (b) (c)

Figure 7.6: Total energy consumption for greedy, balanced and agile policies with solar (7.6a), wind (7.6b), and spot price-based power signals (7.6c) for cases with multiple applications.

applications in parallel, including Graph500, WRF, and the Jacobi solver. We show the performance of power policies when running a single application (Graph500), two applications (Graph500 + WRF), and three applications (Graph500 + WRF + Jacobi), respectively. In all cases, we monitor the overall power consumption, and apply the policies on the nodes, regardless of which applications they are running. We examine the impact of elasticity, power variations, energy storage and transition time on the performance.

Impact of Elasticity. Figure 7.5 shows the runtime of three policies: the dynamic balanced policy (rigid), greedy (elastic) and agile (elastic) policies. Comparing the elastic policies with rigid policy quantifies the benefit of elasticity: our elastic energy management policies

		Balanced	Greedy	Agile
Solar	Static	66.2	42.3	42.4
	Dynamic	56.3	35.9	32.8
Wind	Static	75.2	55.9	45.2
	Dynamic	68.6	37.6	35.6
Price	Static	62.6	44.5	41.8
	Dynamic	56.8	37.0	34.5

Table 7.1: Comparison of overhead power for all policies in terms of percentage of total available power (%).

achieve up to 40% less runtime for solar power signals, 35% less runtime for wind power signals, and 31% less runtime for price-based power signals.

Here we take Graph500 as an example to show the overhead power improvement for each of the policies as summarized in Table 7.1. For rigid tasks, the dynamic policies can reduce the power overhead by 14.9%, 12.1%, and 9% for solar, wind, and price power cases, respectively, while the average reduction of overhead power can be up to 18.8%, 23.0%, and 15.1% in the case of elastic application. In other words, the wasted power can be effectively reduced by dynamic energy management.

Due to the high fluctuation of available energy in the case of wind-generated energy, the static rigid balanced policy can waste up to 75% of total power as overhead, i.e., only 25% of the total available power was consumed for actual computational work. However, when using the agile policy running with elastic applications, only 35.6% of total power was consumed as overhead. The effective power for computational workload has increased by over 150%. With the significant reduction of power overhead, more energy can be devoted into actual computational workload.

The benefit of elasticity derives from reducing the overhead power by deactivating nodes and concentrating more power on performing actual computation. While many parallel applications may not be elastic, our results indicate the importance of elasticity when designing systems for variable power.

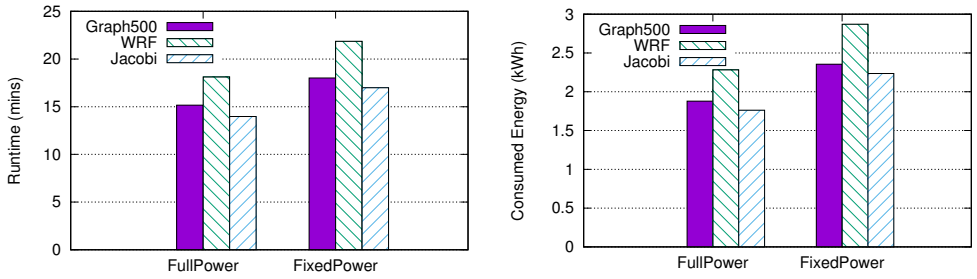
Result: *Energy management policies that are capable of elasticity, i.e., activating and*

deactivating entire nodes, further improve performance over rigid policies that are only capable of active power capping (by 33%-41% for our power signals).

Impact of Power Variations. Next, we examine the impact of power variations on performance in two scenarios: when power is stable and when power is unlimited. For the stable power budget, we set a static power cap equal to the average available power of the variable power signal. For the unlimited power budget, the nodes are free to use as much power as necessary. Figure 7.7a and Figure 7.7b show the runtime and energy consumption, respectively, of Graph500 when running *i)* with unlimited power and *ii)* with a fixed power cap equal to the average power.

As expected, when using variable power, as shown in Figure 7.5, the applications take longer to finish and consume more energy than with unlimited power or with a stable power cap. In particular, in comparison to unlimited power, the greedy elastic policy takes up to 30% longer and uses 21% more energy, the balanced rigid policy takes up to 77% longer and uses up to 72% more energy, and the agile elastic policy takes up to 26% longer and uses up to 17% more energy. Likewise, comparing to a stable power cap, the greedy elastic policy takes up to 27% longer and uses up to 25% more energy, the balanced rigid policy takes up to 74% longer and uses up to 67% more energy, and the agile elastic policy takes up to 13% longer and uses up to 9% more energy.

The results illustrate the importance of energy agile design for variable power. Since adapting to variable power introduces overheads that slows down an application, it typically uses more energy overall than in the case of unlimited or stable power. The goal of our policies is to limit this additional energy; results show that our agile elastic policy uses only 17% more energy than when using unlimited power and only 9% more energy than when using stable power. These comparisons are based on the worst case scenario, i.e., we compare the unlimited and stable power policies with the longest runtime and the highest energy usage among the three power varying cases.



(a) (b)

Figure 7.7: The runtime (7.7a) and power consumption (7.7b) of elastic Graph500 when operating under full power and fixed power budget.

Power Policy	Energy Cost (cents)
Full Power	21.27
Stable Power	27.44
Spot Price Greedy	15.62
Spot Price Balanced	42.07
Spot Price Agile	12.77

Table 7.2: Comparison of energy cost of each application running for non-renewable energy powered cluster.

Based on the energy consumption data and energy prices in Figure 6.1, we calculate the overall energy cost for our policies when using unlimited power and stable power, as well as using our price-based power signal with the greedy, balanced, and agile policies. The results in Table 7.2 show that although using unlimited power yields the shortest runtime and lowest energy consumption, it costs 36% more than using the greedy policy and 67% more than using the agile policy. This price advantage occurs because using unlimited power or stable power does not react to changes in price.

Result: *Adapting to a variable power source introduces overheads that increase application runtime and energy consumption (by 9%-17% in our experiments). Our policies aim to reduce these overheads and reduce the overall cost of non-renewable energy usage.*

Energy Storage. Our elastic policies are able to activate and deactivate a subset of nodes.

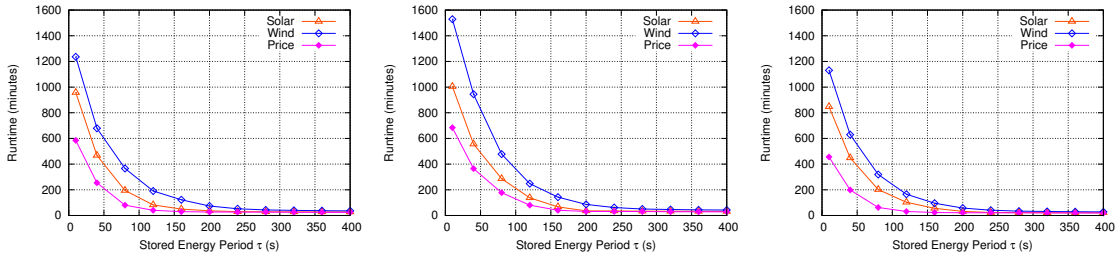
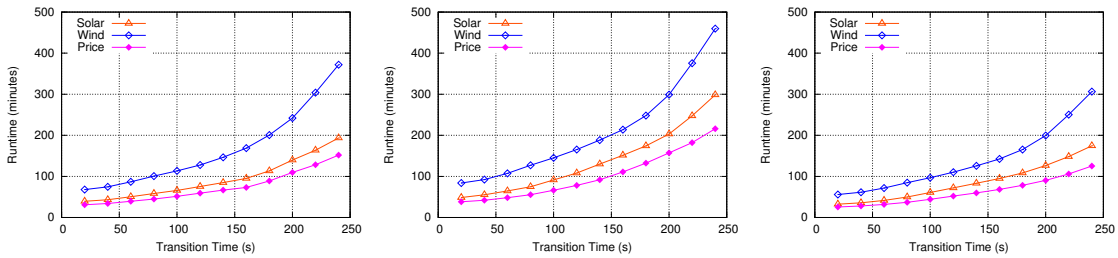


Figure 7.8: The runtime of elastic Graph500 (7.8a), WRF (7.8b) and Jacobi (7.8c) with different energy storage capacities (τ).

However, as discussed earlier, activating and deactivating nodes imposes a long transition time along with the overhead power, which may impact overall performance. One way to reduce the number of transitions is to introduce energy storage capacity, which we capture using the time interval τ over which power is known and stable. In Figure 7.8, we show how the application runtime vary for different values of τ. In this dissertation, we assume a transition time of 10 seconds between active and inactive states, which can be achieved by ACPI's S3 state transitions or S4 for servers with SSDs [84].

As shown, when τ increases, the application runtime decreases, due to fewer numbers of transitions. For example, in the wind power trace, the runtime decreases by up to 7$\times$ when τ increases from 5 to 100 seconds. The results show that the energy storage capacity has a significant affect on performance, and a relatively small amount of energy storage capacity ($<$200 seconds) can provide significant performance improvements. In particular, in this experiment, the runtime decreases to nearly the minimum for $\tau = 200$ seconds. Since storage is expensive to install and maintain, quantifying the effect of energy storage capacity on performance is important in assessing its costs and benefits.

Result: *A small amount of energy storage capacity (enough to support $\tau = 200$ seconds in our experiments) can significantly improve the runtime performance in the case of using variable power.*



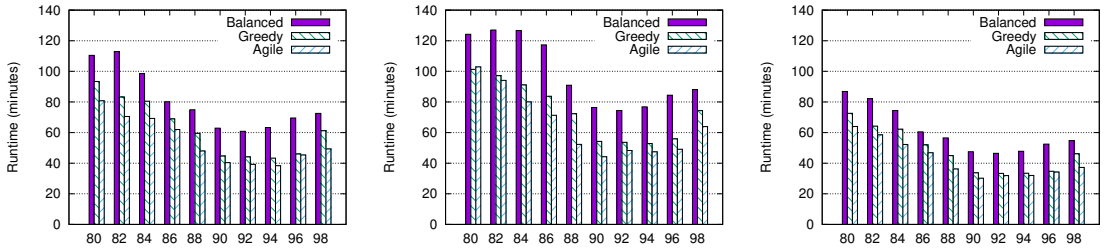
(a) (b) (c)

Figure 7.9: The runtime of elastic Graph500 (7.9a), WRF (7.9b) and Jacobi (7.9c) as a function of the inactive transition time.

Transition Time. In Figure 7.9, we show the impact of transition time, i.e., the time required for servers to change states between active and inactive, on the 64-node cluster. The runtime increases relatively slow when the transition time is low (< 160 seconds), but beyond that, the increment of application runtime becomes significantly greater as the transition time increases. This is because the transition time is approaching the energy storage capacity τ (200 seconds), which makes the system less resilient to the power variations.

Result: *Shorter transition times can greatly improve the performance of energy-agility.*

Value of K. In Figure 7.10, we show the impact on performance when varying the value of CPU threshold, i.e., the parameter K. Here, we use Graph500 as an reference application. In our system, we determine if a node is busy or not using a pre-assigned K value. The power cap is adjusted only when a node's CPU utilization is less than $K\%$. In this set of experiments, we vary the K value from 80% to 98% to evaluate how the runtime varies with different K values. The figures show similar behavior for all three policies. The runtime is inversely correlated to K when K is low. The runtime stays low for a K value between 90 and 94 (90-96 for greedy and agile policies). The runtime increases again when the K value is close to 100. This is because for a low K value, the system reallocates power less often, which usually causes certain nodes to be power hungry and therefore reduces their program execution speed; when the K value approaches 100, the system frequently



(a) (b) (c)

Figure 7.10: The runtime of elastic Graph500 with different K values when utilizing solar energy (7.10a), wind energy (7.10b) and spot price-based energy (7.10c) sources.

switches the power cap, i.e., even when the nodes are busy, it is still possible to take energy away, which increases runtimes.

Result: *For lowest runtime, the value of K should be in the 90-96 range.*

Impact of Available Power. In the previous evaluations, we only consider the available power with a fixed power budget, i.e., solar, wind and spot price based traces. In this section, we consider the impact of power budget on the application performance. We employ agile policy for this evaluation. We vary the total available power, in kW, for the cluster. The total power budget stays the same throughout the application runtime.

As shown in Figure 7.11, when the power budget is in the low region (¡3 kW), the runtime can be significantly improved if the power budget increases: in our case, the runtime is reduced by 94% when the power budget increases from 0.5 kW to 3 kW. When the power budget is in the high region (over 3 kW), the runtime can further reduce, but the reduction is less significant: the runtime decreases by 65% when the power budget increases from 3 kW to 6 kW.

Result: *The application runtime has a better improvement when the power budget increases in the low-power region.*

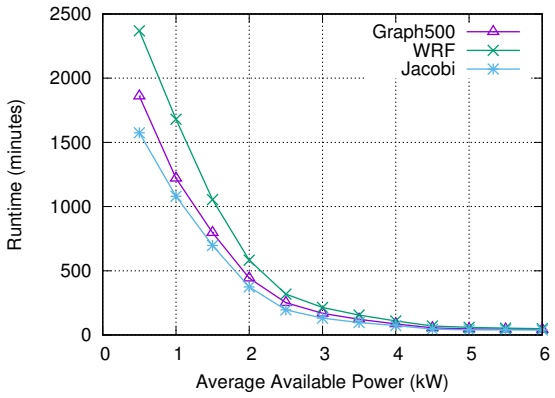
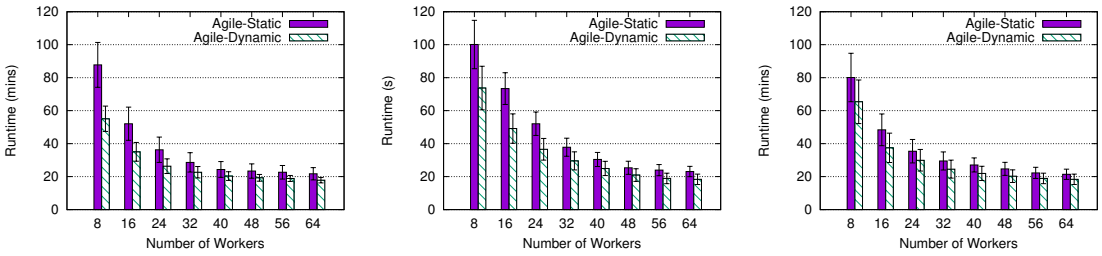


Figure 7.11: The application runtime with different power budgets.

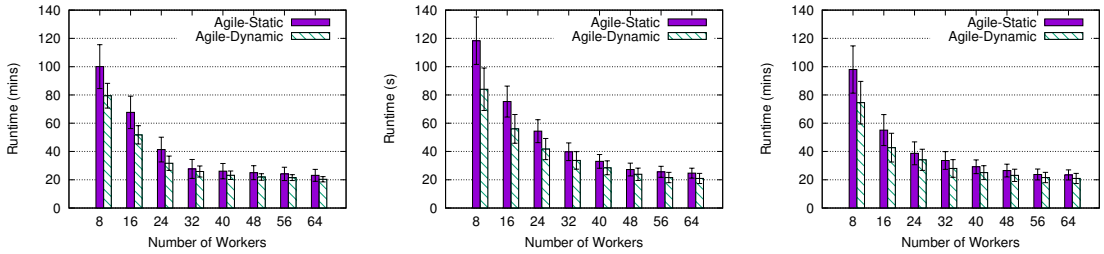
Scalability. We further evaluate the policies on the 64-node CloudLab cluster to test its scalability. Since the Agile policy is known to perform the best among all three policies, in this section, we only compare the performance of individual applications between agile-static and agile-dynamic policies. In addition, we show the runtime of our algorithm in case of different sizes of data centers.



(a) (b) (c)

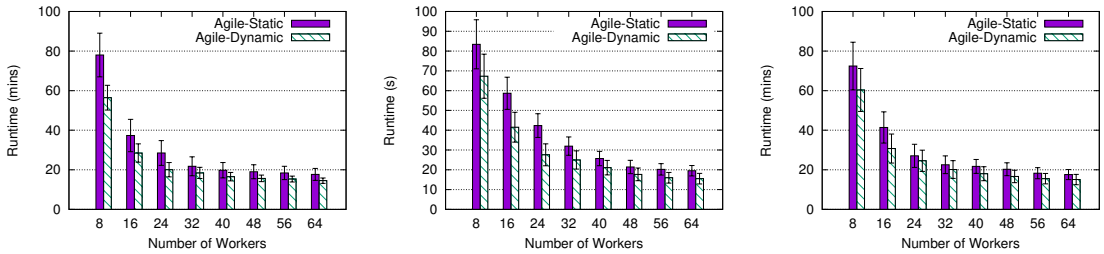
Figure 7.12: The runtime of our elastic agile policy when running Graph500 with solar (7.12a), wind (7.12b), and spot price based power signal (7.12c).

Figure 7.12, 7.13 and 7.14 present the runtimes for three individual applications (Graph500, WRF and Jacobi) as the number of worker nodes changes from 8 to 64. For the three applications, the runtime decreases smoothly while the number of worker nodes increases. The runtimes for both agile static and agile dynamic policies are lower for the case with



(a) (b) (c)

Figure 7.13: The runtime of our elastic agile policy when running WRF with solar (7.13a), wind (7.13b), and spot price based power signal (7.13c).



(a) (b) (c)

Figure 7.14: The runtime of our elastic agile policy when running Jacobi with solar (7.14a), wind (7.14b), and spot price based power signal (7.14c).

the price-based power traces. This is because there are fewer power variations and higher overall power availability in the price-based signal.

For all three applications, the dynamic agile policy results in significantly less running time than the static due to the freedom of changing power allocations in fine granularity. In the case of price based power signal, the runtime for both agile static and agile dynamic policies is lower for the case of the price-based power trace in comparison to the solar and wind traces. This is because there are fewer power variations and higher overall power availability in the price-based signal. By contrast, the solar power signal shows slightly longer runtime than the price signal (shown in Figures 7.12a, 7.13a) and 7.14a. This is because available solar power gradually increases and then decreases, providing more time for an energy agile policy to adjust the power allocation across the nodes. Figures 7.12b,

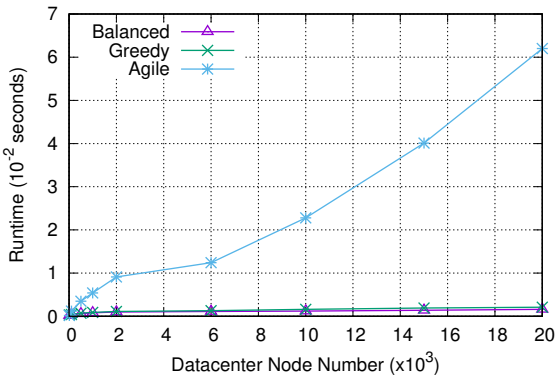


Figure 7.15: Our algorithm runtime for different data center scales.

7.13b, and 7.14b show that the wind-based power trace results in the longest runtime and highest variation. The long running time is consistent with the high variation, and low availability of wind power.

Finally, we show the runtime of our algorithm in case of different data center scales in Figure 7.15. As can be observed, the balanced and greedy policies run in O(1) time complexity, therefore, their runtime stays low when data center scale increases. The agile policy runtime increases linearly when number of nodes increases. However, even for the data center with 20,000 nodes (the current fastest Tianhe-2 cluster has 16,000 nodes), the runtime is less than 0.1 seconds.

Result: *Regardless of cluster scale, energy agile policies significantly benefits from being dynamic. The proposed light-weight energy management policies can determine the power allocation strategy in a relatively short time.*

7.5 Summary

In this chapter, we extend the energy management policies introduced in Chapter 6 to support parallel cloud applications that run on green energy sources. We implement our energy management policies on two different clusters in the CloudLab testbed with three different applications, and evaluate the effectiveness of proposed policies based on real so-

lar, wind, and spot-price-based power signals. We show that P-BFS application Graph500 requires 17% more time and 9% more energy to complete when power varies based on real-time electricity prices versus when power is unlimited at a fixed price. However, since real-time prices are lower than fixed prices, the total electricity cost of our energy-agile policy with real-time spot prices is 67% less than when using fixed prices.

CHAPTER 8

CONCLUSIONS AND FUTURE WORK

In this dissertation, we presented a set of studies on the performance of ABR streaming and cloud parallel applications. In the first part of this dissertation, we present a measurement study for ABR streaming applications in a long distance, wireless, mobile scenario. Using data collected from the application, network, and physical layers, we identify characteristics of the network that directly impact the quality of video services. Then we develop and evaluate a new quality adaption algorithm, named SQUAD, to provide users a smooth playback experience. Our results show that the proposed adaptation algorithm significantly improves Quality of Experience (QoE) by up to 96% in terms of QoE metric spectrum.

In the second part of this dissertation, we explore the options for better application efficiency for ABR video transcoding services from the renewable energy perspective. We define a series of dynamic and static energy management policies that apply to distributed ABR video transcoding tasks. In addition, we extend the these policies to the parallel cloud applications and show the general applicability of our approach. We show that, by utilizing the renewable energy sources, the transcoding grid energy usage can be reduced by 73-83%, and the corresponding energy cost can be reduced by 14-28% with satisfying viewer experience. When coming to the parallel cloud applications, with the effective use of power management policies, the total cost can be reduced by up to 67% comparing to when using fixed prices. This can be achieved by only increasing up to 17% of the application runtime time and 9% of the total energy consumption.

As future work, we intend to further investigate the possible ways to improve the performance of ABR streaming and transcoding services. This includes using QoE metrics to

achieve a balance between variable renewable power and the price variation of grid power. Also our energy management policies can be further extended for increasing the resilience of renewable energy powered data centers and handling grid failure, e.g., in case of hazardous weather or intermittent grid brownouts.

BIBLIOGRAPHY

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

[2] Amazon's Elastic Compute Cloud. http://aws.amazon.com/ec2/.

[3] ExoGENI. http://wiki.exogeni.net.

[4] FFmpeg. https://www.ffmpeg.org/.

[5] The Graph 500 List. http://www.graph500.org/.

[6] Harmonic free 4K demo footage center. https://www.harmonicinc.com/free-4k-demo-footage/.

[7] *IRODS*. `https://www.irods.org/`.

[8] ISO New England energy generation mix. https://www.iso-ne.com/about/key-stats/resource-mix.

[9] *Large-scale GENI Instrumentation and Measurement Infrastructure*. `http://gimi.ecs.umass.edu/`.

[10] Levelized cost of energy analysis 10.0. https://www.lazard.com/perspective/levelized-cost-of-energy-analysis-100/.

[11] MPICH 3.2. https://www.mpich.org/.

[12] Preset settings in x264: the quality and compression speed test. http://www.videoquality.pl/preset-settings-x264-quality-compression-speed-test/.

[13] Rackspace Cloud. http://www.rackspace.com.

[14] Sandvine global Internet phenomena report 2016. https://www.sandvine.com/downloads/general/global-internet-phenomena/2016/global-internet-phenomena-report-latin-america-and-north-america.pdf.

[15] The weather forecasting model. http://www.wrf-model.org.

[16] X264. http://www.videolan.org/developers/x264.html.

[17] The Opportunities and Challenges of Exascale Computing: Summary Report of the Advanced Scientific Computing Advisory Committee (ASCAC) Subcommittee. Tech. rep., U.S. Department of Energy, Office of Science, Fall 2010.

[18] U.S. Department of Energy, Office of Science. The Challenges of Exascale. http://science.energy.gov/ascr/research/scidac/exascale-challenges/, Accessed January 2017.

[19] Adhikari, V.K., Guo, Yang, Hao, Fang, Varvello, M., Hilt, V., Steiner, M., and Zhang, Z. Unreeling Netflix: Understanding and improving multi-CDN movie delivery. In *INFOCOM, 2012 Proceedings IEEE* (March 2012), pp. 1620–1628.

[20] Akhshabi, S., Anantakrishnan, L., Begen, A. C., and Dovrolis, C. What happens when HTTP adaptive streaming players compete for bandwidth? In *Proc. of NOSSDAV* (2012), pp. 9–14.

[21] Akhshabi, S., Begen, A. C., and Dovrolis, C. An experimental evaluation of rate-adaptation algorithms in adaptive streaming over HTTP. In *Proc. of MMSys* (2011), pp. 157–168.

[22] Andersen, D. G., Franklin, J., Kaminsky, M., Phanishayee, A., Tan, L., and Vasudevan, V. Fawn: A fast array of wimpy nodes. In *Proceedings of SOSP* (2009), pp. 1–14.

[23] Andrews, J. G., Ghosh, A., and Muhamed, R. *Fundamentals of WiMAX: Understanding Broadband Wireless Networking (Prentice Hall Communications Engineering and Emerging Technologies Series)*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2007.

[24] Apple and the Environment. http://www.apple.com/environment/renewable-energy/, Accessed January 2017.

[25] Balachandran, A., Sekar, V., Akella, A., Seshan, S., Stoica, I., and Zhang, H. Developing a predictive model of quality of experience for Internet video. In *Proceedings of SIGCOMM* (2013), pp. 339–350.

[26] Beben, A., Wiśniewski, P., Batalla, J. Mongay, and Krawiec, P. ABMA+: Lightweight and efficient algorithm for http adaptive streaming. In *Proc. of MMSys* (2016), pp. 2:1–2:11.

[27] Berman, M., Chase, J. S., Landweber, L., Nakao, A., Ott, M., Raychaudhuri, D., Ricci, R., and Seskar, I. GENI: A federated testbed for innovative network experiments. *Computer Networks 61*, 0 (2014), 5 – 23. Special issue on Future Internet Testbeds – Part I.

[28] Berman, M., Demeester, P., Lee, J. W., Nagaraja, K., Zink, M., Colle, D., Krishnappa, D. K., Raychaudhuri, D., Schulzrinne, H., Seskar, I., and Sharma, S. Future Internets escape the simulator. *Communications of the ACM 58*, 6 (May 2015), 78–89.

[29] Bhanage, G., Seskar, I., Mahindra, R., and Raychaudhuri, D. Virtual basestation: architecture for an open shared WiMAX framework. In *Proceedings of the second ACM SIGCOMM workshop on Virtualized infrastructure systems and architectures* (New York, NY, USA, 2010), VISA '10, ACM, pp. 1–8.

[30] Bhat, D., Wang, C., Rizk, A., and Zink, M. A load balancing approach for adaptive bitrate streaming in information centric networks. In *International Conference on Multimedia Expo Workshops (ICMEW)* (June 2015), pp. 1–6.

[31] Bitmovin, Inc. Optimal segment length for adaptive streaming formats like MPEG-DASH & HLS, 2016.

[32] Buluç, A., and Madduri, K. Parallel breadth-first search on distributed memory systems. In *Proceedings of SC* (November 2011), pp. 65:1–65:12.

[33] Chetto, H., and Chetto, M. Some results of the earliest deadline scheduling algorithm. *IEEE Trans. Softw. Eng. 15*, 10 (Oct. 1989), 1261–1269.

[34] Cho, Y., Mikhail, O., Paek, Y., and Ko, K. Energy-reduction offloading technique for streaming media servers. *Mobile Information Systems* (2016).

[35] Cisco. Cisco visual networking index: Global mobile data traffic forecast update, 20152020 white paper. *Cisco white paper* (2016).

[36] De Cicco, L., Mascolo, S., and Palmisano, V. Feedback control for adaptive live video streaming. In *Proc. of MMSys* (2011), pp. 145–156.

[37] DMR. DMR Netflix statistic report 2016. Accessed: Jan, 30, 2016.

[38] DMR. DMR YouTube statistic report 2016. Accessed: Jan, 30, 2016.

[39] Erman, J., Gerber, A., Ramadrishnan, K. K., Sen, S., and Spatscheck, O. Over the top video: the gorilla in cellular networks. In *Proceedings of IMC* (2011), ACM, pp. 127–136.

[40] Fund, F., Wang, C., Korakis, T., Zink, M., and Panwar, S. GENI WiMAX performance: Evaluation and comparison of two campus testbeds. In *Proceedings of GREE Workshop* (2013).

[41] Fund, F., Wang, C., Liu, Y., Korakis, T., Zink, M., and Panwar, S.S. Performance of DASH and WebRTC video services for mobile users. In *IEEE Packet Video Workshop (PV)* (Dec 2013), pp. 1–8.

[42] Ganjam, A., Siddiqui, F., Zhan, J., Liu, X., Stoica, I., Jiang, J., Sekar, V., and Zhang, H. C3: Internet-scale control plane for video quality optimization. In *Proceedings of NSDI* (May 2015).

[43] Gao, G., Wen, Y., and Westphal, C. Dynamic resource provisioning with QoS guarantee for video transcoding in online video sharing service. In *Multimedia Conference* (2016), pp. 868–877.

[44] Giambene, G. *Queuing Theory and Telecommunications: Networks and Applications*. Springer US, 2005.

[45] Goiri, I., Katsak, W., Le, K., Nguyen, T., and Bianchini, R. Parasol and GreenSwitch: Managing Datacenters Powered by Renewable Energy. In *Proceedings of ASPLOS* (March 2013), pp. 51–64.

[46] Govindan, S., Sivasubramaniam, A., and Urgaonkar, B. Benefits and Limitations of Tapping into Stored Energy for Datacenters. In *Proceedings of ISCA* (June 2011), pp. 341–352.

[47] Hanks, S., Li, T., Farinacci, D., and Traina, P. Generic Routing Encapsulation (GRE). RFC 1701 (Informational), Oct. 1994.

[48] Huang, J., Qian, F., Gerber, A., Mao, Z. M., Sen, S., and Spatscheck, O. A close examination of performance and power characteristics of 4G LTE networks. In *Proceedings of MobiSys* (2012), pp. 225–238.

[49] Huang, T., Handigol, N., Heller, B., McKeown, N., and Johari, R. Confused, timid, and unstable: Picking a video streaming rate is hard. In *Proc. of IMC* (2012), pp. 225–238.

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

[51] ITU. E.800: Terms and definitions related to quality of service and network performance including dependability. Accessed: Jan, 30, 2016.

[52] Jain, M., and Dovrolis, C. End-to-end available bandwidth: Measurement methodology, dynamics, and relation with TCP throughput. *IEEE/ACM Transactions on Networking 11*, 4 (Aug. 2003), 537–549.

[53] Jain, R. *The art of computer systems performance analysis - techniques for experimental design, measurement, simulation, and modeling.* Wiley professional computing. Wiley, 1991.

[54] Juluri, P., Tamarapalli, V., and Medhi, D. Sara: Segment-aware rate adaptation algorithm for dynamic adaptive streaming over HTTP. In *IEEE ICC QoE-FI Workshop* (June 2015), pp. 1765–1770.

[55] Kappiah, N., Freeh, Vincent W., and Lowenthal, D.K. Just in time dynamic voltage scaling: Exploiting inter-node slack to save energy in MPI programs. In *Proceedings of Supercomputing,* (Nov 2005), pp. 33–33.

[56] Kohler, E., Morris, R., Chen, B., Jannotti, J., and Kaashoek, M. F. The Click modular router. *ACM Trans. Comput. Syst. 18*, 3 (Aug. 2000), 263–297.

[57] Koomey, J. Data Center Electricity Use 2005 to 2010. Tech. rep., Analytics Press, August 2011.

[58] Krioukov, A., Alspaugh, S., Mohan, P., Dawson-Haggerty, S., Culler, D. E., and Katz, R. H. Design and Evaluation of an Energy Agile Computing Cluster. In *Technical Report UCB/EECS-2012-13, EECS Department, University of California, Berkeley* (January 2012).

[59] Krishnan, S. S., and Sitaraman, R. K. Video stream quality impacts viewer behavior: inferring causality using quasi-experimental designs. In *Proceedings of IMC* (2012), pp. 211–224.

[60] Krishnappa, D. K., Zink, M., and Sitaraman, R. K. Optimizing the video transcoding workflow in Content Delivery Networks. In *MMSys* (2015), pp. 37–48.

[61] Le, K., Bianchini, R., Zhang, J., Jaluria, Y., Meng, J., and Nguyen, T. D. Reducing electricity cost through virtual machine placement in high performance computing clouds. In *Proceedings of Supercomputing* (2011), pp. 22:1–22:12.

[62] Lederer, S., Mueller, C., Rainer, B., Timmerer, C., and Hellwagner, H. An experimental analysis of Dynamic Adaptive Streaming over HTTP in Content Centric Networks. In *Proceedings of ICME* (July 2013), pp. 1–6.

[63] Lederer, S., Müller, C., and Timmerer, C. Dynamic adaptive streaming over HTTP dataset. In *Proceedings of MMSys* (2012), pp. 89–94.

[64] Lederer, S., Müller, C., and Timmerer, C. Towards peer-assisted dynamic adaptive streaming over HTTP. In *Packet Video Workshop (PV), 2012 19th International* (2012), pp. 161–166.

[65] Li, Z., Begen, A. C., Gahm, J., Shan, Y., Osler, B., and Oran, D. Streaming video over HTTP with consistent quality. In *Proceedings of MMSys* (2014), pp. 248–258.

[66] Li, Z., Zhu, X., Gahm, J., Pan, R., Hu, H., Begen, A.C., and Oran, D. Probe and adapt: Rate adaptation for HTTP video streaming at scale. *IEEE JSAC 32*, 4 (April 2014), 719–733.

[67] Liebeherr, J., Fidler, M., and Valaee, S. A system-theoretic approach to bandwidth estimation. *IEEE/ACM Transactions on Networking 18*, 4 (2010), 1040–1053.

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

[69] Marpe, D., Wiegand, T., and Sullivan, G. J. The h.264/mpeg4 advanced video coding standard and its applications. *IEEE Communications Magazine 44*, 8 (Aug 2006), 134–143.

[70] Mehani, O., Jourjon, G., Rakotoarivelo, T., and Ott, M. An instrumentation framework for the critical task of measurement collection in the future internet. Tech. rep., NICTA, Eveleigh, Sydney, NSW, Australia, October 2012.

[71] Mok, R. K. P., Luo, X., Chan, E. W. W., and Chang, Rocky K. C. QDASH: a QoE-aware DASH system. In *Proceedings of MMSys* (2012), pp. 11–22.

[72] Müller, C., and Timmerer, C. A VLC media player plugin enabling dynamic adaptive streaming over HTTP. In *Proc. of MMSys* (2011), pp. 723–726.

[73] Nygren, E., Sitaraman, R. K., and Sun, J. The akamai network: A platform for high-performance Internet applications. *SIGOPS Oper. Syst. Rev. 44*, 3 (Aug. 2010), 2–19.

[74] Oyman, O., and Singh, S. Quality of experience for HTTP adaptive streaming services. *IEEE Communications Magazine 50*, 4 (April 2012), 20–27.

[75] Pegus, II, P., Varghese, B., Guo, T., Irwin, D., Shenoy, P., Mahanti, A., Culbert, J., Goodhue, J., and Hill, C. Analyzing the efficiency of a green university data center. In *Proceedings of ICPE* (2016), pp. 63–73.

[76] Quinn, M., and Deo, N. Parallel Graph Algorithms. *ACM Computing Survey 16*, 3 (1984), 319–348.

[77] Rakotoarivelo, T., Ott, M., Jourjon, G., and Seskar, I. OMF: a control and management framework for networking testbeds. *SIGOPS Oper. Syst. Rev. 43*, 4 (Jan. 2010), 54–59.

[78] Rao, A., Legout, A., Lim, Y., Towsley, D., Barakat, C., and Dabbous, W. Network characteristics of video streaming traffic. In *Proceedings of CoNEXT* (2011), pp. 25:1–25:12.

[79] Raveendran, A, Bicer, T., and Agrawal, G. A Framework for Elastic Execution of Existing MPI Programs. In *In proceedings of IPDPS* (May 2011), pp. 940–947.

[80] Ricci, R., Eide, E., and the CloudLab Team. Introducing CloudLab: scientific infrastructure for advancing cloud architectures and applications.

[81] Richey, E. Why big tech companies are investing in renewable energy, July 2014.

[82] Roettgers, J. To stream everywhere, Netflix encodes each movie 120 times. https://gigaom.com/2012/12/18/netflix-encoding/.

[83] Sharma, N., Barker, S., Irwin, D., and Shenoy, P. Blink: Managing Server Clusters on Intermittent Power. In *Proceedings of ASPLOS* (March 2011), pp. 185–198.

[84] Sharma, N., Barker, S., Irwin, D., and Shenoy, P. A Distributed File System for Intermittent Power. In *Proceedings of IGCC* (June 2013), pp. 1–10.

[85] Shehabi, A., Walker, B., and Masanet, E. The energy and greenhouse-gas implications of Internet video streaming in the United States. *Environmental Research Letters 9*, 5 (2014), 054007.

[86] Sodagar, I. The MPEG-DASH standard for multimedia streaming over the Internet. *IEEE MultiMedia 18*, 4 (April 2011), 62–67.

[87] Song, M., Lee, Y., and Park, J. Scheduling a video transcoding server to save energy. *ACM Trans. Multimedia Comput. Commun. Appl. 11*, 2s (Feb. 2015), 45:1–45:23.

[88] Spiteri, K., Urgaonkar, R., and Sitaraman, R. K. Bola: Near-optimal bitrate adaptation for online videos. In *Proc. of IEEE INFOCOM* (April 2016), pp. 1–9.

[89] Stockhammer, T. Dynamic Adaptive Streaming over HTTP –: Standards and Design Principles. In *MMSys* (February 2011).

[90] Tian, G., and Liu, Y. Towards agile and smooth video adaptation in dynamic HTTP streaming. In *Proceedings of CoNEXT* (2012), pp. 109–120.

[91] Tolia, N., Wang, Z., Marwah, M., Bash, C., Ranganathan, P., and Zhu, X. Delivering Energy Proportionality with Non-Energy-Proportional Systems: Optimizing the Ensemble. In *Proceedings of HotPower* (December 2008), pp. 2–2.

[92] Tudor, P. N., and Werner, O. H. Real-time transcoding of MPEG-2 video bit streams. In *International Broadcasting Convention* (Sep 1997), pp. 296–301.

[93] U.S. Environmental Protection Agency, Report to Congress on Server and Data Center Energy Efficiency: Public Law 109-431.

[94] Villa, B.J., and Heegaard, P.E. Group based traffic shaping for adaptive HTTP video streaming by segment duration control. In *Proc. of AINA* (March 2013), pp. 830–837.

[95] Vulimiri, A., Godfrey, P. B., Mittal, R., Sherry, J., Ratnasamy, S., and Shenker, S. Low latency via redundancy. In *Proc. of CoNEXT* (2013), pp. 283–294.

[96] Wang, C., Rizk, A., and Zink, M. SQUAD: A Spectrum-based Quality Adaptation for Dynamic Adaptive Streaming over HTTP. In *Proc. of MMSys* (2016), pp. 1:1–1:12.

[97] Wang, C., Zink, M., and Irwin, D. Optimizing parallel HPC applications for green energy sources. In *IGSC* (2015), pp. 1–8.

[98] Wei, L., Cai, J., Foh, C. H., and He, B. QoS-aware resource allocation for video transcoding in clouds. *IEEE Transactions on Circuits and Systems for Video Technology 27*, 1 (Jan 2017), 49–61.

[99] White, B., Lepreau, J., Stoller, L., Ricci, R., Guruprasad, S., Newbold, M., Hibler, M., Barb, C., and Joglekar, A. An integrated experimental environment for distributed systems and networks. In *Proc. of OSDI* (Dec. 2002), pp. 255–270.

[100] Whiteaker, J., Schneider, F., and Teixeira, R. Explaining packet delays under virtualization. *SIGCOMM Computer Communication Review 41*, 1 (Jan. 2011), 38–44.

[101] Xiang, S., Cai, L., and Pan, J. Adaptive scalable video streaming in wireless networks. In *Proceedings of MMSys* (2012), pp. 167–172.

[102] Xu, J., Xing, L., Perkis, A., and Jiang, Y. On the properties of mean opinion scores for quality of experience management. In *Proceedings of IEEE ISM* (Dec 2011), pp. 500–505.

[103] Yao, Y., Huang, L., Sharma, A., Golubchik, L., and Neely, M. Data centers power reduction: A two time scale approach for delay tolerant workloads. In *Proceedings of INFOCOM* (March 2012), pp. 1431–1439.

[104] Yin, X., Jindal, A., Sekar, V., and Sinopoli, B. A control-theoretic approach for Dynamic Adaptive Video Streaming over HTTP. In *Proceedings of SIGCOMM* (2015), pp. 325–338.

[105] Zambelli, A. IIS smooth streaming technical overview. *Microsoft Corporation 3* (2009).

[106] Zhang, Y., Zhang, J., Dong, D., Nie, X, Liu, G., and Zhang, P. A novel spatial autocorrelation model of shadow fading in urban macro environments. In *Proceedings of GLOBECOM* (2008), pp. 1–5.

[107] Zink, M., Schmitt, J., and Steinmetz, R. Layer-encoded video in scalable adaptive streaming. *IEEE Transactions on Multimedia 7*, 1 (Feb 2005), 75–84.