# VIDEO STREAMING QUALITY OF EXPERIENCE (QOE): IN-NETWORK CACHE PREFETCHING AND MOVING QOE MODELS

A Dissertation
IN
Telecommunications and Computer Networking
AND
Computer Science

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

DOCTOR OF PHILOSOPHY

by
SHEYDA KIANI MEHR

PhD , University of Missouri-Kansas City, Missouri, USA, 2021

Kansas City, Missouri
2021

# VIDEO STREAMING QUALITY OF EXPERIENCE (QOE): IN-NETWORK CACHE PREFETCHING AND MOVING QOE MODELS

Sheyda Kiani Mehr, Candidate for the Doctor of Philosophy Degree

University of Missouri–Kansas City, 2021

## ABSTRACT

Video streaming accounts for a significant amount of traffic on the Internet. Users expect a high quality of experience with online video streaming. Service and content providers desire to provide a satisfying experience for end users. Therefore, developing metrics to measure users satisfaction of such services is crucial. Quality of Experience (QoE) of a user for a video streaming service is important for content providers. For video streaming, the DASH (Dynamic Adaptive Streaming over HTTP) standard is one of the common approaches for streaming used by content providers in which a video is divided into segments of different bit rates for delivery. In this research, we studied two inter-related problems for DASH video streaming: 1) in-network caching techniques, prefetching, and 2) QoE measurement and monitoring.

In the first research direction, we focus on content providers utilizing in-network caching and prefetching in order to reduce video delivery latency to provide users a higher quality of experience and reduce the traffic load on the core network. The issue with current prefetching methods is that they do not utilize available resources well; thus, the end users are not able to receive the best possible QoE. These approaches are either mostly naive or they are not compatible with the DASH protocol and they are too complex consuming too much time and compute resources.

We propose a smart video cache prefetching scheme for segment bitrates. Our prefetching approach is based on throughput values in the cache that are forecasted using previous throughput values from clients. Since in a cache environment, multiple clients contend for video segments in the cache, we assess the cache performance and also consider the impact on QoE for each client during contention. When comparing our scheme with an existing scheme, results show that our smart prefetching increases the cache hit rate and reduces the number of unused prefetches for the cache, thereby improving QoE of the clients.

In our second research direction, we focus on objective QoE, for which a number of QoE models has been proposed. The limitations of the current models are that the QoE is provided after the entire video is delivered; also, the models are on a per client basis. We refer to such models as static QoE models. In many situations, such as live events, ensemble QoE during the session is important to understand, especially for multiple clients together, for network and content providers. For this need, we propose two QoE models to capture QoE periodically during video streaming by multiple clients simultaneously, which we refer to as Moving QoE (MQoE) models.

Our first model, MQoE_RF takes into consideration the nonlinear effect due to the bitrate gain and sensitivity from the bitrate switching frequency. Our second model, MQoE_SD focuses on capturing the standard deviation in the bitrate switching magnitude among segments. Then, we study the effectiveness of both the models in a multi-user mobile client environment. We compared our models with an extension of the Model Predictive Control (MPC) QoE model (referred to as MQoE_MO). Our study shows the robustness of our MQoE models. The results show how the MQoE models is able to more accurately capture the overall QoE behavior than the static QoE model and its extension.

APPROVAL PAGE

The faculty listed below, appointed by the Dean of the Graduate Studies, have examined a dissertation titled “Video Streaming Quality of Experience (QoE): In-network Cache Prefetching and Moving QoE Models,” presented by Sheyda Kiani Mehr, candidate for the Doctor of Philosophy degree, and hereby certify that in their opinion it is worthy of acceptance.

Supervisory Committee

Deep Medhi, Ph.D., Committee Chair
Department of Computer Science & Electrical Engineering

Prasad Jogalekar, Ph.D.
Ericsson Inc.

Praveen Rao, Ph.D.
Department of Computer Science & Electrical Engineering

Zhu Li, Ph.D.
Department of Computer Science & Electrical Engineering

Cory Beard, Ph.D.
Department of Computer Science & Electrical Engineering

Beak-Young Choi, Ph.D.
Department of Computer Science & Electrical Engineering

CONTENTS

ILLUSTRATIONS

# TABLES

## ACKNOWLEDGEMENTS

I would like to thank my research advisor Dr. Deep Medhi for his support and courage he gave me to achieve my goal. Dr. Medhi was always motivating for me, he emphasizes the importance of gaining experiences, learning, and growing. He listened to my ideas and let me explore, experience, and build my ideas. He has such a positive attitude and personality with creating a very friendly environment for his students. I am honored to know him and have the chance of working with him.

Besides my advisor, I would like to thank my manager and mentor at Ericsson, and my co-supervisor, Dr. Prasad Jogalekar for all the help and support he gave me. His advice and guidance helped me through my research goals.

I also would like to thank my committee members Dr. Praveen Rao, Dr. Cory Beard, Dr. Zhu Li, and Dr. Baek-Yong Choi for their willingness to serve on my committee and their insightful feedback.

Most importantly, I would like to thank my parents Heshmat Kiani Mehr and Goli Majidi for making this possible, and my sisters Shaghayegh, Niloofar, and Nikoo Kiani Mehr for their constant support and encouragement.

Last, I would like to thank my friends and colleagues who motivated me toward my goals.

# CHAPTER 1

# INTRODUCTION

During the past decade, transition of traditional broadcast television to online video streaming, the advent of the ubiquitous mobile devices, and the increasing demand for high video quality with leads to the fact that a notable amount of Internet traffic belongs to video streaming users. Cisco estimates that by 2022 video streaming volume is projected to be 82% of all the Internet traffic [13].

In this dissertation, we propose a new in-network caching prefetching to improve cache performance and user Quality of Experience (QoE) measurements [25], [26]. Then we propose two Moving QoE models [24] for ISPs and content providers monitoring purposes.

## 1.1 Quality of Experience (QoE); Background

Service and content providers desire to provide a satisfying experience for users, in order to sustain their current users and attract more future users. Therefore, considering a metric to measure users satisfaction of a given service is crucial.

Traditionally, Quality-of-Service (QoS) used to be the metric to measure the performance of online services. However, QoS is a network-centric metric and indicates the performance of the network [5]. Therefore, QoS is not powerful enough to fully express all aspects of a user experience in a given service, specially a video streaming service.

A user-centric metric is required to measure users satisfaction. Quality-of-experience (QoE) is used in many areas such as multimedia services and systems ( [17], [30]). QoE is an extension to QoS and it is not only limited to the use of a system or service, as it is also related to some other end-to-end factors such as users psychological and environmental settings, and the content itself [12]. QoE metrics are classified into subjective and objective metrics.

The Mean Opinion Score (MOS) is the de facto metric for subjective QoE assessment. QoE assessment in terms of MOS demonstrate the users real perception. However, collecting the MOS feedback directly from users is not easy, and it is difficult to automate the collection of MOS in large-scale systems. To address these problems, several objective QoE metrics have been recommended by ITU-T video streaming service, as perceived by the viewer [14].

To support the quantification of QoE by the service provider, QoE models should thus account for both users perception of video quality and users preferences [57]. The following are the common objective QoE metrics [ [22], [46]]:

- Bitrate gain: The quality of a video stream based on the encoding rate.
- Bitrate switching events: the count of bitrate codec changes based on network and buffer conditions.
- Bitrate switching magnitude: amplitude of the Bitrate codec changes based on network and buffer conditions.
- Playback startup delay: The time duration before a video starts to playout.

- Interruption events: If the download rate falls below the buffers playback rate, the buffer gets depleted and the player waits for the buffer to be partially filled before resuming the playback (re-buffering).
- Interruption duration: How long is a stall event in re-buffering.

These metrics are what a video user can directly experience during a video session, without knowing about the details behind the scene. However, an acceptable QoE metric does not always imply that users will have a viewing experience that is considered pleasant. Based one preferences of users, any of these metrics may have more affect on a user experience.

The factors that affect QoE metrics and overall QoE evaluation are more of a concern for content providers and ISPs include the QoS related metrics such as network bandwidth, delay/latency, jitter; the parameter values for various network entities such as clients buffer size, segment duration, bitrate codec, cache size, device resolution; and algorithms such as ABR, prefetching scheme, replacement algorithms etc.

## 1.2 HTTP-based Adaptive Bitrate Streaming; Background

There are some general video streaming techniques such as real-time video streaming, Real-time messaging protocol streaming, progressive download streaming, and HTTP based adaptive bitrate streaming. HTTP based adaptive bitrate streaming attained a special attention for its scalability benefits. The advantages of HTTP for multimedia streaming include the ability to traverse firewalls/NATs, HTTP-based services do not have to make any major modifications to the existing web-servers (a server does not need to maintain session), and furthermore, the existing Content Distribution Networks (CDNs), proxies, and caches already support HTTP traffi [22].

Dynamic Adaptive Streaming over HTTP (DASH) for video streaming has been standardized by ISO/IEC MPEG [6]. Streaming and media companies such as YouTube, Hulu, and Netflix use DASH for streaming video contents.

DASH streams video contents from traditional HTTP servers. A video that uses the DASH standard is available with multiple codec representations. Each video representation is divided into segments with constant playback duration; this is done in order to enable different representations during the playback of the video at the user end. In DASH, each video title is associated with a metadata file called Media Presentation Description (MPD), wherein each segment for a particular resolution is associated with a Universal Resource Locater (URL). Before playback begins, the client requests and fetches the MPD file and parses it to determine the available bitrates and URLs for all segments. A DASH client starts by downloading the first segment, usually with the lowest available bitrate representation, and uses an Adaptation BitRate (ABR) algorithm to determine the most suitable bitrate to be requested for each of the subsequent segments (see Fig. 1). Designing an optimize ABR is a major challenge for video content service providers. The most widely used ABR the Dynamic or Hybrid that makes decisions based on two factors: the network condition represented by the throughput measurements for the downloaded segments and the current status of the client buffer.

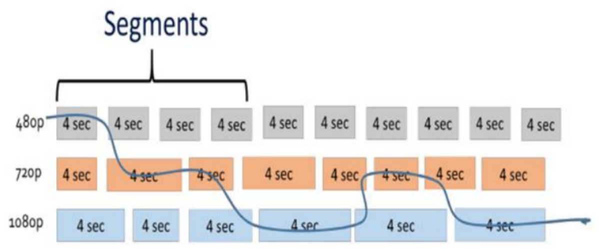


Figure 1: ABR segment bitrate selection based on network throughput and buffer status.

## 1.3 In-network Cache Prefetching; Motivation and Contribution

### 1.3.1 Motivation

A technique that content and service providers utilize in content delivery is caching, by bringing the contents closer to users to increase the hitrate and finally the end users QoE. With hitrate rise, users can avoid to route to a long distant congested path to request a content from an origin server. Bringing the content closer to the user can reduce the latency, a QoS metric which is very effective on QoE metrics and in some literature it is part of QoE metrics. So, caching can improve the QoE of video streaming users. However, cache servers have smaller storage size than that of the main servers. And there are some methods to utilize cache resources and increasing hitrate:

- Replacement: optimize the storage of cache server by removing some less required contents. Most popular cache replacement algorithms are First in first out (FIFO), Least recently used (LRU), Time aware least recently used (TLRU), Least-frequently used (LFU), and Least frequent recently used (LFRU).

- Prefetching: prefetch a content before a user request it.

While there are many studies and work for replacement algorithms in cache, there are not many studies on prefetching a bitrate segment in cache. The reason is that, prefetching is a very difficult decision with the unpredictable network condition, specially when it is implemented in another entity other than the client. The most popular work is [33] which we call it basic prefetching scheme in this dissertation. In addition, prefetching for video content is more challenging that other contents as they have noticeably larger sizes. Also, the expectation of the end users in video streaming is very different that other contents. Low bitrate quality, frequent changes in bitrate, and high latency will significantly create an unpleasant experience for users. Unlike the other contents the multimedia service is very time sensitive so users expect smooth play with no interruption and frequent changes of quality.

The challenge of video content providers is to design a prefetching cache scheme that can prefetch the correct representation for a next segment request of a client. Otherwise, the number of misses increases, and resources such as storage will be filled up with wrong contents. With misses, the load on the backbone links will increase by fetching the wrong segment bitrates prefetches from the origin server. The resources will be wasted and it causes even more degradation of cache performance and QoE. Therefore, prefetching is a very risky process but if it has the right scheme, its benefits are significant. With enhancing the prefetching in cache, hitrate can decrease the number of unused prefetch segments, and improves the bandwidth usage of a link that is connected from cache to video content servers.

When multiple clients simultaneously contend for a particular video from the cache, the overall performance and QoE may be impacted negatively. There are two main prefetching scheme modes, cache-driven or a client-driven prefetching that each has its own drawbacks. For example, with cache-driven prefetching scheme, service providers may not be able to guarantee a premium quality of service with DASH.

### 1.3.2 Scope and Contributions

This dissertation proposes a smart cache prefetching scheme for DASH video streaming.

The issue with the previous methods is that they do not utilize the available resources, such as bandwidth of the link, and the users are not able to receive the best possible QoE. These approaches are not smart enough or they are mostly cache-based or hybrid (cache assistant) prefetching approaches which are not compatible with a typical DASH.

We proposed a smart cache prefetching scheme that along with a comprehensive set of studies on cache, the cache interaction with two clients, it provides an optimized cache and QoE performance. Firstly, the method increases the hitrate, and decrease the storage usage, while reduce the load between cache server and origin server link. Secondly, it improves the QoS and QoE metrics of each user individually. The scheme prefetches segment bitrate based on forecasting client throughput at the cache entity, by using previous throughput values from a client. Cache applies each throughput value in a rate based ABR and prefetches the required video segment with an appropriate representation.

The smart cache framework is a modular system including prefetching and replacement modules. For the cache to handle multiple clients, we have implemented a multi-threading cache that allows multiple clients to request the same video in order to observe cache performance under contention. Having a multi-threading cache allows us to answer a number of questions. For example, how does the cache function if multiple different clients request the same video at nearly the same time? How will cache performance and QoE be affected with a traffic on the link among client, cache, and origin server? Thus, our comprehensive study focuses on understanding the impact that conditions such as these will have on the function of the cache. In this work, we study two different experiment setup for the network bottleneck. First, implement a bottleneck between client and cache server. Second, we implement a bottleneck between cache server and origin server. For the second setup, a replacement algorithm was considered when the number of clients were added up.

In our smart cache approach, the clients assist the cache ,the DASH client decision on a bitrate segment is independent from the cache server. It means, client doesn not align its bitrate decision based on whatever segment bitrate that is already available on the cache at the moment. The DASH client will recive what it asks for based on the DASH protocol selection. We propose a smart client-driven cache prefetching scheme to predict the most accurate segment for clients requests. It may be noted that several works considered multiple clients in their experiments; however, these works did not study each QoE metric and the impact of their proposed method on each client. In our work, we study the performance impact on each client separately.

## 1.4 Moving QoE Measurement and Monitoring Models; Motivation and Contribution

### 1.4.1 Motivation

Service providers, network providers, device manufacturers, and end users desire better experience and more economic approach for their experience while save resources such as network, compute, and storage. A major challenge that video content service providers face is to understand whether the users are receiving a fulfilling video streaming experience. They can use the QoE metrics and QoE models for optimization of service delivery (encoding pipeline, load balancing, resource allocation, etc.) to provide a pleasant QoE to the end user while optimize resources usage [9].

There have been a number of works, which proposed QoE models with objective QoE metrics ( [9], [22]). The metric selection and formulation of a QoE model is the key point of a strong QoE model. Traditional QoE models use metrics such as Peak Signal to Noise Ratio (PSNR) and Structural Similarity (SSIM) index, Video Quality Metric (VQM), etc. These metrics do not convey user experience well enough, because they provide the impairments due to compression or packet losses. The new metrics express the psychological and emotion aspects of the users expections. The most popular objective model is MPC [58] where considered a liner relation of bitrate gain, bitrate switching magnitude and rebuffering duration. However, the choice of QoE model and metrics depends on the application and the interests of the providers. There are various parties of video streaming service delivery that benefit from QoE modeling from [9] that are mentioned below:

- Network providers: QoE models with important QoE metrics are used by the network providers to detect an issue and to decide the right actions such as resource allocation, network throttling, load balancing, caching and network provisioning.

- Service providers: The service providers should consider user requirements and expectations to keep their users satisfied. Therefore, suitable QoE models can provide an insight into the metrics and their impact on the service, and in turn allow the service provider to take appropriate decisions to ensure acceptable end user QoE.

- Device Manufactures: Devices have different capabilities and one of QoE factors is the device screen size. Thus, good QoE models can help device manufacturers for considering the device features such as display size, display resolution, CPU, RAM.

- End users: The most important part of the service delivery that will benefit from an appropriate QoE model is the end user. It is important to provide a service that meets users requirements and it depends on the QoE metrics and the QoE formulation that define the user satisfaction.

### 1.4.2 Scope and contributions

In this dissertation, we propose two Moving QoE (MQoE, in short) models for QoE monitoring that periodically measure the aggregated QoE among multiple clients. While there are a number of objective QoE models, a limitation of the these models is that the QoE is provided after the entire video is delivered; also, the models are on a per client basis. For content service providers, observed QoE is important to monitor and understand ensemble performance during streaming such as for live events or concurrent streaming when multiple clients are streaming. Since the content providers serve numerous users, at any instance, the interest is to understand on ensemble how their services are being received by the users.

Moving QoE (MQoE) model, which in that QoE is measured periodically for multiple client to inform a controller is formulated in two versions. One version is non-linear which it depends on the bitrate switching counts as a negative metric. And the other version is linear which it depends on the bitrate switching magnitude. Then we study the effectiveness and robustness of both the models in a multi-user mobile client environment, with the mobility patterns being based on traces such as car, train and, ferry.

In order to study our MQoE_MOdels and to mimic the perspective of a content provider to monitor the video streaming service, we implemented a multi-client environment accessing video streaming service. Our goal was to consider two aspects in our study: first one is related to the specific video being streamed and the second one on user mobility. For the first aspect, we consider the same video being streamed to multiple clients to conduct a controlled study. Interestingly, the situation of multiple clients accessing the same video is also common in practical situations such as when a video lecture is delivered by an instructor watched by students, or live events when multiple users want to watch the same event such as a sport event. It may be noted that for delivering live contents, YouTube also provides API that uses DASH [2]. Considering the same video being watched by multiple clients, let us observe whether QoE is being equitably allocated among clients. For the second aspect in our study, we conducted our study of the MQoE_MOdels with mobile clients for users in transit to fully capture the moving QoE behavior. For this, our work considers three different mobility patterns for the clients to emulate traveling in a car, ferry or train.

The novelty of our work is that, to our knowledge, we are the first one to present moving QoE models for video stream monitoring from a content service providers perspective. Furthermore, our work helps content providers to apply moving QoE models to monitor performance on a continual basis. Since in our experience, content providers give different priorities to different factors, we present two QoE models so that content providers have the freedom to choose one of the models that suit their needs based on their preferences. As we show through our work, a static QoE model such as MPC QoE model is not applicable for use in a content monitoring environment.

## 1.5 Additional Contributions

There are other work that we collaborated with some colleagues on similar to or different than the work we present in this dissertation. The first work is a QoE modeling and ABR evaluation study [57]. The other work is a study of Wi-Fi signal strength and application throughput which was implemented in the University of Missouri-Kansas City campus network [20].

### 1.5.1 QoE model and DASH ABR algorithm

One approach to improve QoE of the video streaming users is to provide an ABR algorithm that can make a better segment bitrate representation selection. Most ABR algorithms are based one the network bandwidth, however the segment sizes vary for a given bitrate codec representation. So, client only by considering bandwidth changes does not provide an accurate segment bitrate prediction. One of the previously proposed ABRs, the Segment-Aware Rate Adatation (SARA) algorithm accurately predicts the download time for the next segment, then it makes an informed bitrate selection by considering segment size information from a modified MPD file.

In this work, we implemented and studied the SARA algorithm, on an Android mobile device running a browser-based video player along with other Rate-based and Buffer-Based ABRs within three different mobile traffic such as car, ferry, and train. We evaluated SARA in two different set of experiments, objective and subjective. For objective evaluation, we developed a new QoE model and measured the QoE of users when using SARA compared to other ABRs. This QoE model considers QoE metrics such as bitrate gain, bitrate switching amplitude and interruption ratio. In addition, we created two reward QoE metrics that help a more accurate QoE evaluation. The score obtained from the new objective QoE model demonstrates that the SARA algorithm for mobile clients gives a better QoE among all the other exiting ABR algorithms. Second, we implemented a subjective evaluation among 54 non-expert users where the users watch the video and rate it. The results showed that SARA for mobile clients outperforms others ABRs in terms of the Mean Opinion Score (MOS), while achieving improvement in terms of the interruption ratio.

### 1.5.2 Assessing Wi-Fi Signal Strength and Application Throughput in a Campus Network Environment

Wi-Fi network users in a university campus are students, faculty, staff and visitors. Due to a variety of reasons, including budgetary issues or physical restrictions, the Campus Information Services (CIS) Division is limited on the number of Wi-Fi Access Point (AP) installations in different campus buildings and where they are installed. Sometimes users complain that they are not getting good Wi-Fi access. The CIS staff members often use a Wi-Fi analyzer to test the signal strength to determine if additional Wi-Fi APs should be added, with the general understanding that if the Wi-Fi signal is better, the users will have a better experience.

In this work, we study and assess the possible relationship between Wi-Fi signal strength and the application performance in University of Missouri-Kansas City campus network. We collect our data with a software-based Wi-Fi analyzer along with a scripting tool for application performance. We collected data in three different buildings over multiple floors on our campus network.

We found that there is a statistically conclusive evidence that an association exists between Wi-Fi signal strength and the application throughput. This association can be described through a nonlinear regression fit when the server is located within the proximity of the end device. In addition to the broader question on relationship, we sought answers questions such as: 1) How is the signal strength measured in dBm related to the application performance for different conditions? and 2) Is there a connection between how far the client is from an AP to the application performance?

## 1.6 Organization

The rest of this dissertation is organized as follows: Chapter 2 presents literature survey of two problems, in-network cache prefetching schemes and QoE modeling. In Chapter 3, we present our smart cache framework including the modularized model and algorithm. The evaluation of this model has been demonstrated in Chapter 4. Next, we propose a novel Moving QoE for DASH video streaming in Chapter 5 followed by a comparative evaluation in Chapter 6. We end with a concluding remark in Chapter 7.

# CHAPTER 2

# LITERATURE SURVEY

## 2.1 Cache Prefetching

Numerous studies have been done in order to improve user QoE along with cache performance, hitrate, and link utilization.

Incorporating the network entities as an assistance to video stream client will add advancement to the current video streaming delivery and users QoE. Employing cache entity to improve QoE has been studied in some work with different methods. Poliakov et al. [43] built a caching-aware rate decision algorithm for the client that has no collaboration with other entities. The algorithm is designed based on a model in the presence of a cache. Benno et al. [10] presented a framework that applies a client-driven HAS rate-determination algorithm (RDA); their experiment shows the positive effect of caching on video quality and its overall benefit to the clients. Mueller et al. [39] propose fair adaptation scheme (FAS), which uses a probe method to identify the available bandwidth for the next segment by downloading the first few bytes of the next segment. They only consider two representations.

Cache technique such as replacement and prefetchig can improve the cache performance and hitrate immersively, if they are designed well enough for the content providers, user preferences, and overall configuration such as contents and available resources. There are not a plentiful number of works for cache segment bitrate prefetching. Cache prefetching compared to replacement technique has many risks and uncertainty, because the prefetching is implemented for a group of remote entities (clients) and each of them has their own personalized requests and expectations.

The prefetching is based on the information from a client such as video segments, throughput measurements, latencies and etc.

If the number of cache hit increases, then the cache performance improves. A side effect of a prefetching algorithm with a low hitrate is that it increases the load on the back bone from cache server to origin server as for each missed request, cache needs to refer to origin server and double the effort to deliver the request of the clients.

The two main work on cache prefetching for a video segment bitrates are [33] and [44]. Liang et al. [33] assumes that there is high probability that a client requests the same bitrate as the previous bitrate request, so prefetching the next segments is based on the current bitrate. The issue with this basic method is that it does not utilize the available resources, such as bandwidth of the link, and a user is not able to receive the best possible QoE for a segment. Rejaie et al. [44] designed a cache server that prefetches the segments based on the average bandwidth between the origin server and the cache server. This is not comparable with a typical DASH adaptation scheme that is based on the throughput of the link between the client and the cache when there is a cache hit. If a bottleneck is on the client-cache link, all segment requests will be cache miss.

There are also a few works where the client chooses the bitrates based on cache information (cache-based), or the client receives assistance from the cache (hybrid). Pham et al. [41] propose a client ABR that is assisted by a cache server in order to get better QoE. The cache monitors the bandwidth from the origin server and sends it to the client. Mok et al. [37] propose QDASH, a cache that measures the available bandwidth and is responsible for helping clients to select the most suitable video quality level. Krishnamoorthi et al. [29] suggest cooperative buffer-aware prefetching in which a client continually shares its buffer occupancy with the cache, and the cache shares its fragments and segment fetch timing with the client. The client can thereby give preference to downloading fragments that are already stored within the cache. Liu et al. [34] propose a joint client-driven prefetching and rate adaptation algorithm (CLICRA), in which the client sends a signal to a cache that includes information about the anticipated segment(s). Also, the cache sends a signal to the client that includes information, such as the caching status and the fetching time of a segment. The cache affects the client decision for the next segment(s). This does not match the regular DASH technologies used today, where the client is independent from cache. In the real network, cache is transparent for a client, client only sends out a request and expect to receive a response as fast as possible. Furthermore, sending all those information from the cache to each client who is watching a video is challenging. In general these method with existence of multiple clients will not be very efficient. Cache-driven prefetching methods generally have limitations when being used in DASH because the cache-driven prefetching and client-driven rate adaptation for DASH cannot operate well together. In summary, the client and the cache are dependent on each other in [34] where the client follows the cache orders, meaning that the client decision is effected or based on cache decisions.

Thus, a client-driven technique can be better as the cache will take care of everything centrally without giving more loads to the clients or interrupting their regular process (DASH protocol). We design a smart cache can handle multiple clients and each client is not aware of the existence of other clients and the cache server. The Clients do not need modification and they are independent of any cache algorithm and they just share information with cache server. We propose a smart cache prefetching approach, the cache follows the client requests, and the client is independent from the cache. the smart client-driven cache forecasts the most accurate segment for multiple clients. It may be noted that some works considered multiple clients in their experiments; however, these works did not study the impact of their proposed method on each client. In our work, we study the performance impact on each client separately.

## 2.2 QoE Modeling

As we discussed before, there are two categories of QoE assessment, subjective and objective. The former is the direct feedback of the users which will be very helpful to have a sense of real result of opinions of the users in a large scale subjects. However, this method for evaluating QoE is time consuming and there is a probability that the users may not always be honest about their answers and they might consider their perspective for rating, we will discuss more about subjective QoE for our method in the end of this section.

For the latter one, there have been a number of work, which proposed QoE models with objective QoE metrics. Some work proposed models to capture the exponential relation between the QoE and QoS parameters [23], [47], [7], [48], [49] and [8]. However, QoS metrics are not sufficient to measure the satisfaction of the users. Hoßfeld et al. [21] discussed factors that influence a QoE model. Due to the nonlinear relationship between these metrics, it is not easy to construct a simple model [22]. On the other hand, there are other approaches formulated a linear parametric model with QoE objective metrics. Yin et al. [58] proposed a QoE Model Predictive Control (MPC) approach. This QoE model was used in assessing QoE for video streaming in the later work [56]. They considered QoE metrics such as bitrate gain, rebuffering and the difference between the quality level of consecutive chunks (switching amplitude). Yarnagula et al. [57] formulated a complex parametric QoE model over a number of metrics. De Vriendt et al. [16] addressed the problem of how to assess QoE of an end user under the form of a prediction for the MOS. For surveys on QoE models for DASH, see [ [9, 22]]. Wang et al. [53] proposed a model to maximize the QoE by considering the average video bitrate, frequency of variations and the amplitude of variations. The variation metric is a centralized measure for the variation of the video quality around the average quality that is denoted as spectrum equation in [59]. Moldovan et al. [38] proposed a quadratic problem formulation which maximize both service quality and fairness. They define the objective as being to maximize the average quality, minimize the number of quality switches, and ensure equal utility (QoE) among users. Xue et al. [55] proposed a model which combines instantaneous qualities and cumulative quality taking into account video segment quality. The instantaneous quality was obtained using a linear model using Quantization Parameter (QP) values and instantaneous rebuffering. Guo et al. [19] proposed a model which estimates the overall quality using a linear combination of median and minimum of the instantaneous quality. The instantaneous quality was obtained from QP values using the normalized quality vs. inverted normalized quantization step size (NQQ) model. Tran et al. [52] presented a model considering encoded video quality and quality variation. The quality of the encoded video is calculated for each segment considering the average QP.

We presented two moving QoE models that can report ensemble QoE in review windows for multiple clients together on a periodic basis. To our knowledge, we are the first to propose MQoE models for video stream performance monitoring that can be used by content service providers. Secondly, the multi-client scenario has rarely been studied before, which is important to consider for content providers.

Note that our work focuses only on quantitative moving QoE models. On the other hand, there are many work that implement objective and subjective assessment of QoE. Subjective QoE, measured using Mean Opinion Score (MOS), is studied for video delivery [ [57], [36]]. It is also pointed out in [36] that subjective assessments are costly, time consuming, and not scalable. First, we note that no previous work has studied subjective QoE from a content service providers perspective. Secondly, subjective QoE assessment, for moving QoE snapshots, is impractical from a content providers perspective for video quality monitoring. Consider a content service provider streaming a live event to thousands of end users. If subjects were to be employed to mimic understanding this situation, a significant number of subjects would be necessary. If monitoring is to be captured, say every minute for video stream monitoring, then this would require each subject to be reminded every minute to record the MOS score, which may potentially require additional

10 to 15 seconds to record, the user would be distracted from continuing to watch the video for the next minute. In addition to that, the bias that can happen in a subjective study, specially with MOS methos and with less number of subject is substantial. The viewers preferences and expectations are different. In other words, the definition of a satisfying experience is not similar for all the users. For example, some users prefer continuous streaming without interruption, although with degraded quality of bitare as a very satisfying experience, however there are other users that this conditions will not meet their expectation and they define a satisfying experience with a video stream with high quality of bitrate but they have a great tolerance for interruptions. Therefore, for subjective study, psychological factors and individual preferences should be studies, too. Furthermore, fatigue could quickly set in even after a few minutes of scoring, resulting in noisy measures.

Thus, as we proposed here, a quantitative MQoE approach is a more viable approach for monitoring video streaming for content service providers. Most of the quantitative models discussed above were formulated for assessing QoE for an individual user or the QoE is reported at the end of a video session. Secondly, they were not readily adaptable in a multi-client scenario, especially in a scenario like a live event and on a rolling basis. Our proposed QoE models fill the void in the current literature.

Live streaming has also been investigated in a number of work on video streaming [ [35, 51, 54]]. But these works focused on the QoE for a single client, and none are from the perspective of a content service provider.

Our QoE models are the first work to address moving QoE (collectively for multiple clients) for video streaming which is more suited for a service provider point of view monitoring instead of individual client perspective. The benefit of these models is that the providers can monitor users experiences periodically and keep track of variations with different conditions of the environment that cause a pleasant or unpleasant experience. By observing the changes and root cause of the unpleasant changes in users experience. Then providers can take appropriate actions such as different resource allocation for the sake of QoE fairness among the users or giving a priority to the premium users. In addition, if the models are implemented in the clients along with designing a QoE-driven ABR, which is another challenge for the content providers as we mentioned in the previous section, each client can achieve QoE improvement in an equitable way or individually.

# CHAPTER 3

# SMART CACHE PREFETCHING

The main goal of caching and its advance techniques is to provide the maximum hitrate. There are a few client-driven in-network cache approaches for segment bitrate prefetching that are compatible with DASH protocol. The popular work is [33], which we refer it as basic approach in this work. This section will briefly discuss the basic approach and then present the smart cache prefetching.

## 3.1 Basic Cache Prefetching

Similar to many content cache server, the demonstrated processes for the basic cache follows these few steps. A client sends a request, then the cache server receives this request and if the requested content is already stored in the cache server storage (cache hit), the client request will be served from the cache. Otherwise, cache will ask the origin server for the content (cache miss). At the time the origin server sends the content, cache server stores a copy of the content to its storage and sends a copy to the client. In some cache servers, whenever a cache miss happens, the cache server ask the origin server for the missed content but without making a copy in its storage. In this case, the client will be served directly from the origin server. In our algorithm, there is no direct communication between a client and the origin server.

A caching technique for DASH video streaming purposes are trickier than other content caching techniques. A caching technique for DASH videos should consider the details of the DASH protocol, the segmented video with various bitrates which all are mostly provided in a DASH MPD file, etc.

In the basic cache prefetching approach, in the beginning of a session, a client asks for the MPD of a video file that is a must-have file for the desired video. The cache receives that request and then asks for it from the origin server. The MPD file will be sent back to the client immediately by the cache. In this step, MPD does not have a very decisive role for basic cache server and also it does not take up much of its space. Next the client receives the MPD file, parses it, and starts requesting the video segments.

Basic cache provided a simple solution for prefetching the next segments with the right bitrate codec representation. Prefetching the next segment request as the same bitrate as the current segment bitrate request. For example, if a client is asking for segment $S_n$ with bitrate $B_n$, then cache will use these information of request $n$, which will end up prefetching segment $S_{n+1}$ with bitrate $B_n$. This decision of cache would be the same for the rest of the segments till the moment that client DASH protocol sees a change in the network or in its buffer and request another bitrate for the next segment request.

The advantage of this approach is that it is independent of the DASH ABR and it is very simple and speedy with the least computation power that it can be done on the fly for each segment request of multiple users. The prefetched segments can be stored in the cache server before the next segment request from a client arrives at the cache. However, the prefetches are not accurate most of the time. specially, when a link with a very high loaded traffic delaying the whole process and affect the next segment bitrate selection of

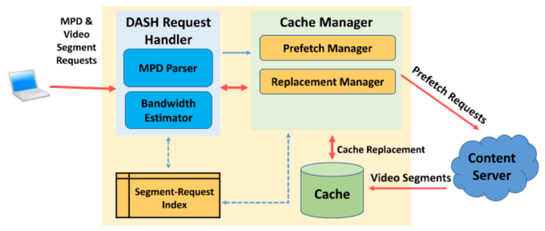


Figure 2: High level architecture

the client.

When the bandwidth of the links is very fluctuating, basic cache has difficulty to make the right decision as it relays purely on the previous segment bitrate requests of the clien. The cache is not sync to the current status of the client and there is a lag between actual bitrate any prefetched bitrate of a segment.

## 3.2 Smart Cache Prefetching

Smart Cache Prefetching motivation is to mitigate the disadvantages of basic cache prefetching. Smart cache prefetching keeps the computation to the minimum and increase the prefetching accuracy. The trade-off between accuracy and computation is a controversial part of cache prefetching scheme designing. Samrt cache framework has two main components, DASH Request Handler (DRH) and Cache Manager (CM) (see Fig. 2).

### 3.2.1 DASH Request Handler (DRH)

DRH includes two modules, MPD Parser and Throughput Collector. DRH is directly connected to the DASH clients and serves the requests. The client requests the initial MPD file which will go through the cache server and from there to the origin server. In this framework, MPD file is a must-have for the cache server. DRH transfers the MPD file to the MPD Parser module in DRH, which parses it and store data in the Segment-Request Index module. Then similar to basic scheme, cache will send back the MPD file the client.

Every time that the client requests for a segment, it also includes some information in the request HTTP header. Client measures the moving average throughputs, $A_j$ , for $j-$th request, which is computed based on the previous $m$ segments:

$$A_j = \frac{\sum_{k=j-n}^{j-1} S_k/n}{\sum_{k=j-n}^{j-1} T_k/n} = \frac{\sum_{k=j-n}^{j-1} S_k}{\sum_{k=j-n}^{j-1} T_k}, \quad j \geq 2, \quad j \geq m. \tag{3.1}$$

Where $S_k$ is the segment size of the $k$-th segment, and $T_k$ is the total time taken for the client to receive $k$-th segment from cache. $j >= 2$, because for the first request, client and cache do not have any useful information for prefetching yet. If $j < m$, then Equation. 3.1 just calculates the mean.

DRH component receives the HTTP header information per client segment request, including the client smooth throughput value, by using Throughput Collector module. Throughput Collector stores the throughput values in the Segment Request Index along with the client ID, Session-ID, requested video ID, and its related list of bitrates from the MPD file for the later use.

### 3.2.2 Cache Manager (CM)

CM includes the Prefetch Manager (PM) and Replacement Manager (RM). The PM module in CM has two functions; first it uses the throughput stored in Segment-Request Index to forecast the client throughput for the next segment that will be requested by the client. Forecasting is based on an exponential smoothing (with trend) approach.

The forecasting method used by CM is based on exponential smoothing with trend on the moving average of $j-$th received throughput value from the client to estimate the $j+1$-th moving average throughput value of the client throughput at the time of cache prefetching. Using $A_j$ and the forecasted throughput value for the current segment, $FIT_j$ , the cache module forecasts the throughput for the next segment ($FIT_{j+1}$). Briefly, the exponential smoothing with trend is calculated as follows

$$\begin{aligned} F_{j+1} &= FIT_j + \alpha * (A_j - FIT_j) \\ \tau_{j+1} &= \tau_j + \delta * (F_{j+1} - FIT_j) \\ FIT_{j+1} &= F_{j+1} + \tau_{j+1}, \end{aligned} \tag{3.2}$$

where $\tau_j$ is the trend component for segment $j-$th, $F_j$ is the forecast value without trend for segment $j-$th, $A_j$ is the actual value measured for segment $j-$th, and $FIT_j$ is the forecast value including trend at time $j-$th. Here, $\alpha$ (0 $le\alpha$ $\leq$ 1) is the smoothing constant and $\delta$ (0 $\leq$ $\delta$ $\leq$ 1) is the trend smoothing constant.

Now discussing exponential smoothing without trend, which is similar to Equation. 3.2, but $\tau = 0$. Considering that $F_j + 1$ is the future throughput for the $(j+1)$-th segment given that $F_j$ is the currently forecasted throughput, and $A_j$ is the current actual throughput from the client for the $j$-th segment. For the smoothing parameter, $\alpha$ ($0 \leq \alpha \leq 1$), this relationship is given by

$$F_{j+1} = \alpha A_j + (1 - \alpha) F_j. \tag{3.3}$$

PM uses a rate-based ABR (similar to client ABR, without buffer consideration) to select the appropriate segment bitrate for prefetching. For the very first segment request, the cache serves this request with the lowest bitrate from the origin server, as the client would ask for the lowest bitrate representation with its ABR. When the client asks for the second segment, PM needs a throughput value in order for its ABR to prefetch the second segment. In this case, we consider the cache throughput value for the second segment to be the size of the segment divided by the time that it takes for the request to transfer from the client to the cache plus the time that it takes for the cache to fetch the segment from origin server. Therefore, the second segment estimated throughput value is calculated as follows:

$$\begin{aligned} &\text{Throughput\_forecast} = \frac{S_1}{\Delta t_1 + \Delta t_2}, \\ &\Delta t_1 = t_2 - t_1, \\ &\Delta t_2 = t_3 - t_2. \end{aligned} \tag{3.4}$$

Here, $t_1$ is the moment that the client sends the request, $t_2$ is the instant that the cache receives the request from the client, and $t_3$ is the instant that the cache sends the segment to the client. The first throughput measurement does not include the transmission time between cache server and the client. When client requests for the second segment, it has the throughput value from 3.1 which can be used in the cache with 3.2 or 3.3.

In the smart cache framework, we also designed a multi processing cache to handle multiple client connections simultaneously. Each connection is served as a separate process. As a process, client connection has multiple threads, such as prefetching thread (prefetches a segment from origin server) and current thread (calculates throughput and bitrate of the prefetching segment). As we mentioned, after the cache calculates the forecasted throughput of the next segment, the prefetching method in the cache uses the same ABR that is used by the client. Thus, PM second function is an ABR similar to the ABR in the clients to use the forcasted throughput as input and prefetch a bitrate for the next segment. The client uses a hybrid ABR that is throughput-based and buffer-based, with a moving average value for throughput. The cache entity only uses the throughput-based version of the client ABR. As we can see cache is acting as a DASH client itself, measuring throughput and using ABR for bitrate selection. And cache behavior is dependent on DASH client behavior.

The role of the last module, RM, in the smart cache framework is to determine the segments that are to be replaced in order to accommodate new segments, especially during contention from multiple clients. If the number of segments stored in the cache becomes greater than an initial number, $n$, then the cache makes the decision to replace segments using the RM. In our approach, we replace the Least Recently Used (LRU) segment.

**Algorithm 1** smart cache procedure

```
$R$: Request from client, $\alpha$ and $\delta$ smoothing constants

Function MAIN($R, \alpha, \delta, n, Q$)
  $bitrates[]$
  $URLs[]$
  $j \leftarrow 2$
  $forecast\_throughput \leftarrow 0$
while $R$ do
    if $R == MPD$ then
        $bitrates, URLs \leftarrow$ PARSE($R$)
        STORE($bitrates, URLs$)
    else if $R == S_1$ then
        $t_1 \leftarrow$ HEADER($R$)
        $t_2 \leftarrow$ MEASURE_TIME()
        RECIEVE_FROM _ORIGIN_SERVER($S_1$)
        $t_3 \leftarrow$ MEASURE_TIME()
        SEND_TO _CLIENT($S_1$)
        $\Delta t_1 \leftarrow t_2 - t_1$
        $\Delta t_2 \leftarrow t_3 - t_2$
        $forecast\_throughput \leftarrow$ SIZE($S_1$)/($\Delta t_1 + \Delta t_2$)
        $S_2 \leftarrow$ PREFETCH($forecast\_throughput$)
        ADD($S_2$)
        $T_2 \leftarrow 0$
        $FIT_2 \leftarrow 0$
        STORE($T_2, FIT_2$)
    else
        SEND_TO_CLIENT($S_j$)
        $A_j \leftarrow$ HEADER($R$)
        $forecast\_throughput \leftarrow$ FORECAST($A_j$)
        $S_{j+1} \leftarrow$ PREFETCH($forecast\_throughput$)
        ADD($S_{j+1}$)
        $j++$
    end if
end while
End Function
Function FORECAST($A_j$)
  RETRIEVE($FIT_j, T_j$)
  $F_{j+1} \leftarrow FIT_j + \alpha * (A_j - FIT_j)$
  $T_{j+1} \leftarrow T_j + \delta * (F_{j+1} - FIT_j)$
  $FIT_{j+1} \leftarrow F_{j+1} + T_{j+1}$
  STORE($FIT_{j+1}, T_{j+1}$)
  $return\ FIT_{j+1}$
End Function
Function PREFETCH ($forecast\_throughput$)
  Use ABR with all available bitrates from MPD
  $return\ S_{j+1}$
End Function
Function ADD ($S_{j+1}$)
if $Size(Q) == n$ then
    Pop(Q)
end if
  $Q \leftarrow S_{j+1}$
End Function
```

# CHAPTER 4

# SMART CACHE PREFETCHING EVALUATION

## 4.1 Environment

The smart cache framework is implemented on the Global Environment for Networking Innovations (GENI) research testbed [11]. The GENI platform allows us to regulate the link bandwidths. There are three entities in the implementation: client(s), cache server, and origin server. In the topology, client(s) is/are connected to a LAN switch. The in-network cache server is also connected to the same LAN switch. The cache server is directly connected to the origin video content server Fig. 3.

We used the Apache2 HTTP server as the origin content server. The DASH player client along with the basic/smart cache server are implemented in Python. The DASH player uses ABR [1] to determine the bitrate for the next video segment based on the throughput measurements of the link connected to the cache server and the playback

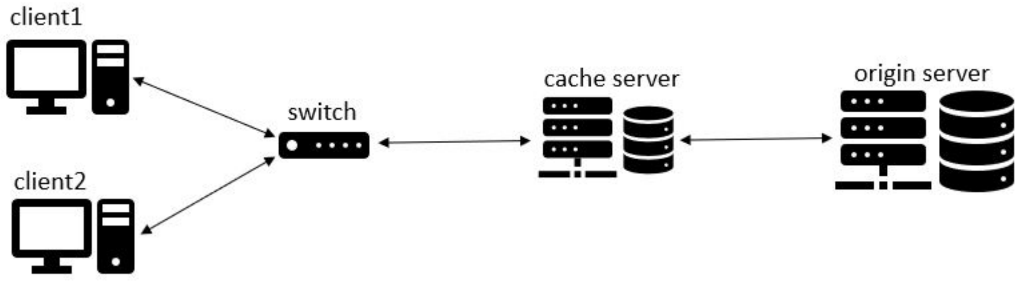


Figure 3: Network topology with clien(s), cache and origin server

buffer status of the client.

## 4.2 Traffic

This work was experimented with two different assumption, bottleneck on the link between client and cache server (client-cache link) and bottleneck on the link between cache server and origin server (cache-server link).

### 4.2.1 client-cache link

The link between the cache server and client is most susceptible to network fluctuations. In order to evaluate the performance of in-network cache, the following scenarios have been assumed between cache server and a single client:

- In the base scenario, a fixed bandwidth of 5 Mbps is considered on the link between the client and cache as well as the link between the cache and the video server with no background traffic. The only traffic that is going through the link is from the single client that requesting video segments from the cache server.
- In the short interval background traffic scenario, we fixed the bandwidth as in the base scenario but emulated a congested link between the cache server and the client. The congestion was generated with a background traffic of multiple short file downloads over TCP connection by other clients using the link between the video client and the cache. These short background spikes in congestion was created within multiple intervals over the duration of the video session.

- For the third scenario, we fixed the bandwidth as in the base scenario but generated long interval background traffic by emulating longer durations of congestion on the link between the cache server and the client. As in the case with short interval background traffic, we used multiple file downloads by other clients.

### 4.2.2 cache-server link

The backbone link between the cache and origin server is the most time consuming path, cache goes through this path either for prefetching or fetching a content for a client. In this case, we are onsidering two general scenarios, in the first scenario there is one client, and in the second scenario there are two clients. While the focus of this part is on scenarios with multiple clients, results for the single-client scenarios are included for comparative purposes. Within the first scenario, we tested once with no background traffic on the links (10Mbps for each link), and then with a UDP background traffic on the link between the cache and the origin server. We increased the size of the links from 5Mbps in the previous experimental assumption to 10Mbps in these set of experiments. Because, there are two client competing for the resources and the maximum size of the video segments is about 4MB.

The double-client scenario is studied only with bottleneck traffic between the cache and the origin server. The choice of the bottleneck link through background traffic is motivated by prior studies. For example, it has been reported that the cache-server link with limited bandwidth could cause a bottleneck in the network [33] and [27]. Based on [29], a simple cache solution is potentially beneficial when the bandwidth bottleneck is between the client and the cache. However, caching can be more effective if the bottleneck is between the cache and the origin server. A common issue that arises when the DASH technology applies caching technique is that the connectivity between the client and the cache server will have a higher rate (throughput) than that of the connectivity between the cache and the server [32]. This causes a cache to consume more time fetching un-cached segments from the original server [39]. Thus, the ABR in the client, incorrectly estimates network conditions, which in turn causes wrong quality decisions, ultimately resulting in a negative effect on QoE [42]. When there are multiple clients contending at the same cache, this issue is exacerbated. On the other hand, this issue does not happen in the first set of study when the bottle neck is the access link. Cache existence and its benefits is observable when there is bottleneck beyond the access network and prefetching the right contents will save time and resources for the overall framework. With these assumptions bottleneck on client-cache link may improve the cache performance, however, there would not be much evidence of QoE improvement with cache prefetching. Thus, in this work, we consider the case of a network bottleneck on a cache-server bandwidth backbone link.

The background UDP traffic was created by transferring a very large file (1.6 GB) using UDP socket python code on each of the cache and the origin server machines 40 sec after the first client began its requests. The 40-second interval is chosen based on our preliminary experiments since by this time, the throughput value has already surpassed the peak threshold for initial bitrate ramp-up phase. On the other hand, if we generate the traffic before 40 sec, then the client would stay on the requests with lower bitrate for a greater number of segments. Therefore, implementing background traffic has no significant effect, as the requests would have remained at the low bitrates for a longer period before the throughput hits the threshold. For all the experiments, we replicated each of the possible scenarios five times to avoid any artifact from the GENI testbed.

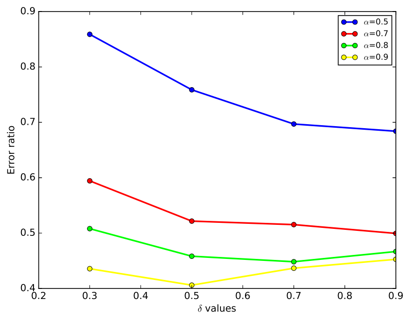
## 4.3 Dataset

For evaluation, we used the publicly available *Big Buck Bunny* and the *Elephants Dream* DASH videos from the ITEC dataset [31]. For the first study with bottleneck on the client-cache link, we considered *Big Buck Bunny* and the *Elephants Dream* 4sec palyback duration with 150 and 164 segments, respectively. Each video has 20 different bitrate codec representations. In the second study, 10 sec and 15 sec segment playback durations for *Big Buck Bunny* were selected; while the entire video has 60 segments and 40 segments, respectively (for a total video length of 600 sec). Both datasets have 20 different bitrates (resolutions) for each segment to choose from. The logic behind choosing longer playback duration was the observability of the results.

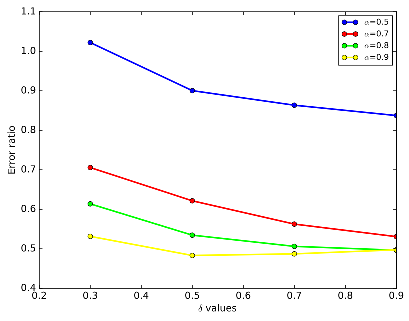
## 4.4 Initial Analysis and Adjustment

### 4.4.1 First Study

For the first set of experiments with bottleneck on the client-cache link, the accuracy of the forecasting method presented in Equation. 3.3 is dependent on the values used for $\alpha$ and $\delta$. To determine the best values for $\alpha$ and $\delta$, we evaluated the forecasting method under the mentioned network traffic scenarios using the values $\alpha$ = 0.5, 0.7,



(a) Big Buck Bunny

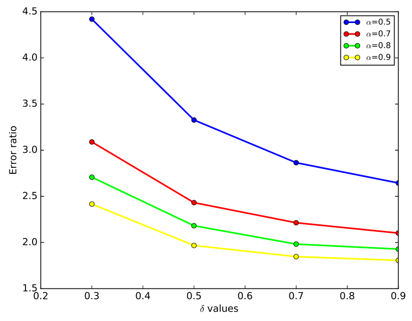


(b) Elephants Dream

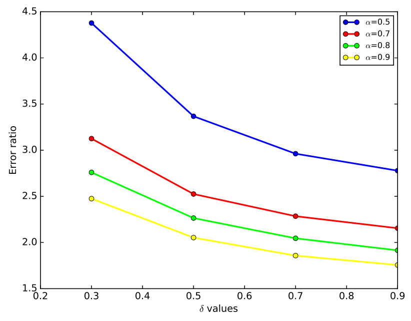
Figure 4: Forecasting error for Big Buck Bunny (a), Elephants Dream (b), No Traffic

0.8, 0.9 and $\delta = 0.3, 0.5, 0.7, 0.9$ for two datasets *Big Buck Bunny* and *Elephants Dream*. Higher $\alpha$ values indicate that $FIT_{j+1}$ is more dependent on previous $A_j$. The results from the evaluation of the forecasting methods under different parameters are presented in Fig. 4-6. Based on these evaluations, we see that the accuracy increases as we increase $\alpha$ for all scenarios.

However, for $\alpha$ , the accuracy in no background traffic scenario for both datasets increases up to 0.5 and then decreases as seen in Fig. 4a and Fig. 4b. For short Interval traffic shown in Fig. 5 and long Interval traffic shown in Fig. 6, the accuracy increases up to 0.9 that is the maximum $\delta$ value in the experiment. Considering all the scenarios, we found that $\alpha = 0.9$ and $\delta = 0.7$ give the best result for all different network traffic scenarios, which is used for the studies in the next section. For this study set,there is only one user involved, so including a replacement algorithm is unnecessary.

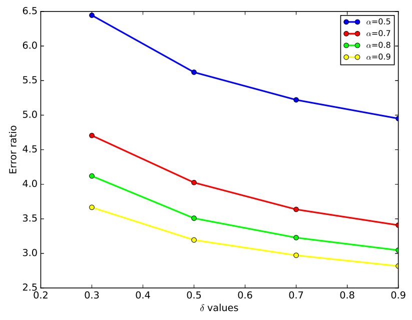


(a) Big Buck Bunny

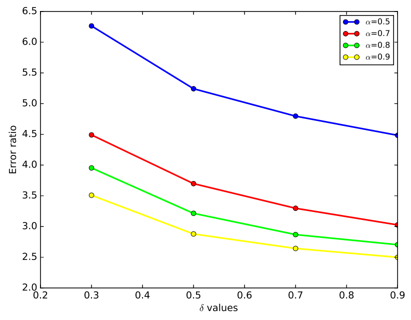


(b) Elephants Dream

Figure 5: Forecasting error for Big Buck Bunny (a), Elephants Dream (b), Short Interval Traffic



(a) Big Buck Bunny



(b) Elephants Dream

Figure 6: Forecasting error for Big Buck Bunny (a), Elephants Dream (b), Long Interval Traffic

Table 1: Exponential smoothing error ratio for double client scinatio with 10s and 15s

| client(playback duration) | $\alpha$=0.5 | $\alpha$=0.7 | $\alpha$=0.9 |
|---|---|---|---|
| client1(10s) | 13.42 | 10.74 | 9.35 |
| client2(10s) | 15.83 | 12.31 | 10.25 |
| client1(15s) | 20.36 | 15.79 | 14.22 |
| client2(15s) | 17.51 | 14.13 | 13.14 |

### 4.4.2 Second Study

For the second set of experiments, bottle neck on the cache-server link, the accuracy of the exponential smoothing forecasting method (Equation. 3.4) is dependent on the value used for $\alpha$. We notice that trend and $delta$ is not very effective in our forecasting accuracy. To determine the best values for $\alpha$, we evaluate the forecasting method under the most tense network background traffic, by applying UDP traffic on the cache-server link for the double-client scenario, with 10s and 15s segment playback duration, using $\alpha =$ 0.5, 0.7, 0.9. See Table. 1 where the error ratios are presented. The accuracy increases as we increase $\alpha$ for all clients in both the datasets. Based on this initial assessment, we chose $\alpha =$ 0.9 that has the lowest error ratio for the rest of our study.

It is important to note that the second client has higher error ratio than the first client. The cause for this difference is most likely since the second client experiences more traffic on the bottleneck link, which is caused by the presence of the first clients requests. The other observation is that the 15s segment duration video is found to have a higher error ratio compared to the 10s segment duration video.

we conducted our study based on the condition that when a second client initiates a connection, the cache is empty. Therefore, in the double-client scenario, the cache has to prefetch the segments for both clients, because the required segments from the second client that the first client also requests are all removed. To capture the performance penalty associated with the cache misses, it is necessary to consider a cache policy in which the cache is cleared for each new client [29].

We consider a 10-segment cache size in our study with the Least Recently Used (LRU) method in the Replacement Manager. In the double-client scenario, the second client starts with a short delay of 20 sec than the first client. We observed that when both clients run simultaneously, the second clients segment requests are mostly cache hits. This occurs because the second client requests the same segments as the first client, especially for the first few video segments. In this case, there is not much difference in the results of single-client and double-client scenarios. However, by injecting a delay for the second client prior to initiating its connection, we were able to see the effect of cache size on our final results. From our initial experimentations, we found that the first segment is deleted in less than 20 sec when the cache size is 10 segments.

We compared the smart cache with other works such as [33] and [29], which the duty of the cache is to prefetch the next segment of the same bitrate as the current segment [29]. Although the cache performs a basic function in these works, some may implement extra cache functions to improve the prefetching scheme, while still maintaining the same basic function. Overall, in this study we consider the basic concept that is common to most of the works mentioned in our related work, and we refer to it as a basic prefetching scheme. Thus, throughout our study, we compared our approach with a basic prefetching scheme. After showing the result of throughput accuracy with exponential smoothing in the above section, in the next section, we study the result of cache performance metrics such as cache hitrate and its effect on client throughput, the nlater one was considered for the second study with bottleneck on the core link. Finally, since various metrics are attributed to QoE [22], we did a QoE study considering the following metrics: average latency, average bitrate, and average of bitrate switches or oscillations. please notice that, this evaluation related to the double-client in the second study with the bottleneck on the cache-server link.

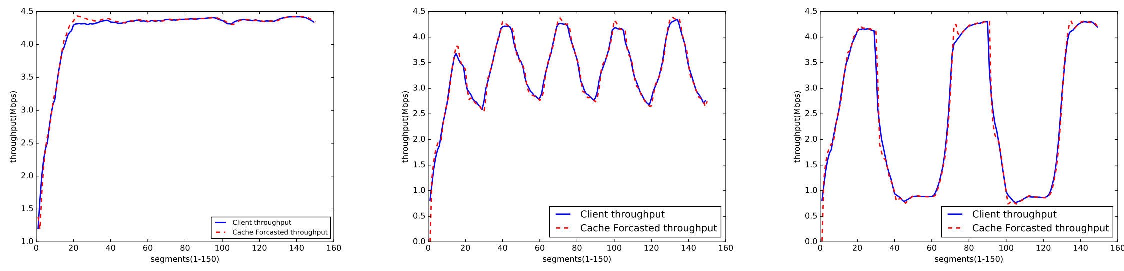
The ABR used at the cache and the client are different, the one implemented in the cache is throughput-based and the one implemented in the client is a hybrid/dynamic ABR. Client ABR determines the next bitrate based on the throughput observed for previous $k$ segments. In our evaluation, we used $k = 10$ as we found this value to be suitable after our initial analysis. This value of $k$ is also used by a previous work [50]. Cache ABR determines the next bitrate based on the throughput forecasted with Equation. 3.3 or Equation. 3.4.

## 4.5 Comparative Study

### 4.5.1 First Study

#### 4.5.1.1 Evaluating the Cache Throughput Forecasting

In order to predict the correct bitrate for the next segment at the smart cache, it is necessary to forecast the throughput for the next segment observed by the client. Accurate forecasting ensures that the cache is able to prefetch the next segment before it receives the request from the client. In the current section, we present our evaluation on the accuracy



(a) No Background Traffic (b) Short Interval Background Traffic (c) Long Interval Background Traffic

Figure 7: Throughput comparison smart cache vs. client

of our forecasting method for estimating client moving average throughput under the three different traffic scenarios. In an ideal case, both the values of the forecasted throughput and the actual next throughput value that client observes on the link need to be the same; however, in practice, it is not exactly the same as there is a ramp up phase and moreover it highly difficult due to the unpredictable short term fluctuations in the network conditions. Thus, we concentrate on reducing the error ratio defined as follows:

$$ERR = \frac{\sum_{k=j-n}^{j-1} |FIT_j - A_j|}{\sum_{k=j-n}^{j-1} |A_j|}. \tag{4.1}$$

In 7. a, we present the comparison of the actual measured throughput at the client and the throughput forecasted at the cache using our model for the first scenario with no background traffic for the *Big Buck Bunny* video. In the base scenario with steady throughput, we observed that the forecasted value is fairly close to the actual value and the error ratio given by equation 4.1 was found to be 0.44% across multiple runs. In the case of the *Elephants Dream* video, we found the deviation to be 0.49% (Table 2).

Table 2: Throughput error ratios comparison for smart cache (Bandwidth=5 Mbps)

| Video | No Background Traffic | Short Interval Background Traffic | Long Interval Background Traffic |
|---|---|---|---|
| BigBuckBunny | 0.44% | 1.80% | 2.97% |
| ElephantDream | 0.49% | 1.76% | 2.64% |

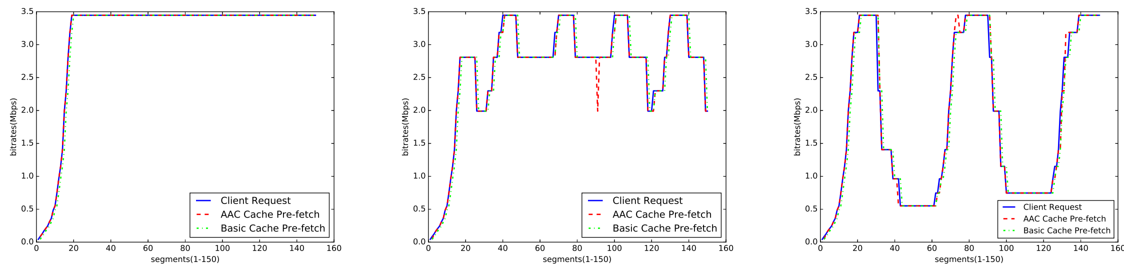
For the second scenario, short interval background traffic, we see that the throughput fluctuates between 2.5 Mbps and 4.25 Mbps in short intervals multiple times during the video session. Even in this case where the throughput fluctuates frequently, we were able to predict the throughput with high accuracy (see Fig. 7b). For the *Big Buck Bunny* video, the forecasted throughput follows the trend in the client throughput with an error ratio of 1.80%. Similarly, a 1.76% deviation for the *Elephants Dream* video was observed. In this scenario, the error ratio increase is not surprising compared to the base scenario with no background traffic due to the influence of the short duration background traffic. In case of long interval background traffic 7c, the throughput observed fluctuates between a much higher range as compared with the second scenario. In this case, the fluctuations are between 1 Mbps and 4.25 Mbps but are less frequent. The forecasting error ratio increased to 2.97% and 2.64% for *Big Buck Bunny* and *Elephants Dream*, respectively. This higher increase in error is concentrated to instances where there was a sudden change in the measured throughput, since the forecasted throughput value is based on the previous moving average throughput value. As we will demonstrate in the next section, this increase in the forecasting error does not affect the prediction of the next bitrate significantly. Thus, we see that even under varying network conditions and over multiple runs for different videos, we are able to forecast the throughput for the subsequent segments with high accuracy.

#### 4.5.1.2 Analysis of Cache Hits and Misses

Recall that our approach uses a client-driven prefetching method. We compare our approach with a basic bitrate-based prefetching scheme which is the base of the most previous works on cache prefetching. This basic prefetch scheme works as follows: the cache prefetches the next segment of the same bitrate as the current segment [29].

In this section, we analyze the performance of smart cache by considering the cache-hit rate for the prefetched segments in comparison with a basic prefetch scheme. We evaluate the cache-hit rate under different scenarios for both the *Big Buck Bunny* and *Elephants Dream* videos. When the client starts the video stream with the ABR, it begins with the lowest bitrate and gradually increase the bitrate to match the network conditions until it reaches the highest bitrate supported by the network condition. For the scenario with no background traffic (Fig. 8a), we can compare the bitrates being prefetched by the basic scheme and the smart smart scheme with the client requested bitrate. The prefetches made by the basic scheme lags the bitrate fetched by the client ABR during the initial ramp-up stage causing multiple cache-misses. Only when the playback reaches the steady state does the basic prefetching cache scheme start to prefetch successfully. The overall, success rate for the basic scheme in the current scenario across 10 runs was found to be 87.20% (Table. 3). With smart cache, we are able to leverage the forecasting model to predict the next bitrate for the prefetch even during the ramp-up phase, giving it a cache hitrate of 99.33% (Table. 3).

In the second scenario with short interval background traffic, with the congestion in the link between the client and the cache due to the competing TCP connections, the



(a) No Background Traffic (b) Short Interval Background Traffic (c) Long Interval Background Traffic

Figure 8: Bitrate request smart cache vs. basic cache vs. client

Table 3: Cache hitrates in smart and smart schemes (Bandwidth=5 Mbps)

| Video | No Background Traffic | | Short Interval Background Traffic | | Long Interval Background Traffic | |
|---|---|---|---|---|---|---|
| | basic | smart | basic | smart | basic | smart |
| Big Buck Bunny | 87.20% | 99.33% | 75.33% | 93.2% | 71.06% | 89.73% |
| Elephant Dream | 88.41% | 99.39% | 77.92% | 93.53% | 73.96% | 88.35% |

ABR observes reduced throughput as the video playback proceeds. In Fig. 8b, we see the client bitrate request pattern and the cache prefetch request rate for both basic and the smart cache schemes. In this case, the cache hitrate performance of smart cache is more than 15% better than the basic prefetching scheme. Finally, the last scenario has a highly variable throughput due to sudden intermittent background traffic spikes that lasts for few minutes during playback. Under this scenario, we see that the bitrate prefetched by the smart cache scheme matches the client bitrate except at the instant of sudden change (Fig. 8c). Even then, the smart cache scheme is able to outperform the basic cache scheme by a higher margin as compared to the previous scheme (see Table 3). Thus, by making better predictions, smart cache improves cache hitrate and reduces unnecessary prefetches that reduces the bandwidth use between the cache and the origin DASH servers. More importantly, it also reduces the segment fetch time for the client as most of the requested segment are served from cache server itself.

### 4.5.2 Second Study

#### 4.5.2.1 Cache Hitrates

For all the scenarios, the smart cache scheme hitrate is significantly higher than that of the basic hitrate (see Table 4). Furthermore, for the single-client and double-client scenarios, the difference in the hitrate for the smart cache compared to the basic cache scheme increases more for videos with 15s than it does for videos with 10s. On the other hand, the throughput accuracy error from the smart cache throughput accuracy section shows that 15s has more error than that of the 10s. The cause of this higher error ratio may be attributed to the different features of data, such as the fact that there are fewer segments with larger size and higher bitrates for the 15s than there are for the 10s. For the single-client scenario, we report results without and with background traffic to understand how background traffic affects performance. We observe the highest increase in smart cache hitrate as compared to the basic scheme in the single-client scenario when the traffic is added to the cache- server link (see Table 4). The smart cache double-client scenario with a 15s has a same hitrate compared to a 10s (see Table 5). Overall, the smart cache hitrate increases less when the second client is added to the video fetching process, as compared to smart cache with a single client. The reason could be that the cache takes care of two clients at the same time and there is more traffic with the presence of the second client.

Table 4: Cache hitrate ratio for all the scenarios

| playback duration | basic | smart | increase |
|---|---|---|---|
| 10s-notraffic | 66.67% | 97.33% | 45.99% |
| 15s-notraffic | 50.00% | 97.5% | 95.00% |
| 10s-traffic | 41.00% | 80.00% | 95.12% |
| 15s-traffic | 27.00% | 70.00% | 159.26% |
| 10s-Double-Client | 52.17% | 79.67% | 52.71% |
| 15sDouble-Client | 41.5% | 73.25% | 76.51% |

#### 4.5.2.2 Client Throughput Measurements

In order to understand the client throughput, it is important to understand that the increase in the number of misses in the cache server causes the cache to request the missed segments from the original server, thereby consuming more time and resources for the client to receive the requested segment. Since there is a small number of misses in the smart cache scheme, each time that the client requests a segment from the cache, it will be served by the cache itself. In other words, the cache does not have to go all the way to the original server to fetch the requested segment and returns it to the client. This effect will be more prevalent when there is a background traffic on the core link. We found that the throughput measured by the DASH player is higher with the smart cache scheme than with the basic scheme. In Table 5, the measured throughput for smart and basic cache for single-client with 10s are almost similar, because only the first few segments are cache misses, which have smaller sizes. In the double-client scenarios, the clients that request 15s segments experience less throughput increase when using smart cache than for 10s segments. The second client for 15s shows the least increase with smart cache compared to basic (see Table 6). Due to a higher number of misses in the cache, the difference in

Table 5: Client average throughput (Mbps) with a single client scenario

| playback duration | basic | smart | ratio-increase |
|---|---|---|---|
| 10s-notraffic | 4.17 | 4.74 | 13.67% |
| 15s-notraffic | 3.73 | 4.89 | 31.10% |
| 10s-traffic | 3.05 | 4.13 | 35.41% |
| 15s-traffic | 2.55 | 3.61 | 41.57% |

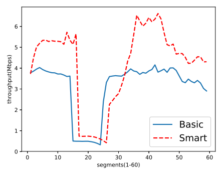
Table 6: Client average throughput (Mbps) with double clients scenario

| playback duration | basic | smart | ratio-increase |
|---|---|---|---|
| 10s-double client1 | 2.74 | 3.65 | 33.21% |
| 10s-double client2 | 2.54 | 3.62 | 43.70% |
| 15s-double client1 | 2.45 | 3.06 | 24.90% |
| 15s-double client2 | 2.47 | 2.78 | 12.55% |

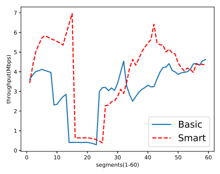
throughput increases for the double-client scenario is less when compared to the single-client scenario, specifically for the second client. Note that, in the basic scheme, the client must wait for the cache to fetch all of the missed segments from the origin server. The difference in measured throughput between the two schemes increases by adding traffic on the outgoing link of the cache. In Fig. 9, we present throughput behavior with time line shown based on the segment number in the x-axis. The first peak shows that the throughput measurement is increasing to a threshold while the two clients are requesting segments and background traffic is running; however, it suddenly drops after 40 sec and stays steady. After the bottleneck is taken away, the trend increases. However, the two clients still compete for resources that results in an unsteady behavior.

### 4.5.2.3 Analysis of Client QoE

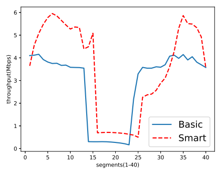
Finally, we focus on client QoE by considering the following metrics: average latency, average bitrate, and average of bitrate switches frequency or oscillations.



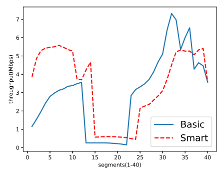
(a) First client 10s



(b) Second client 10s



(c) First client 15s



(d) Second client 15s

Figure 9: Throughput measurement for clients with smart vs. basic cache.

1) Latency: It has been reported that the time that users wait for the requested content to be downloaded from the server to local devices can significantly influence user experience [40]. The user may not always feel the latency in each segment delivery but the measurement of latency can be a good metric to evaluate the QoE. In all the scenarios, smart cache displays better QoE than the basic scheme in regards to average latency (see Table. 7). For the single-client 10s and 15s without traffic, the delay difference between basic and smart cache is not significant. But for the double-client scenarios, we observe that for each client the delay of 15s is more than that of the 10s. In addition, the difference between smart cache and basic for 15s is more than 10s. The second client in both datasets experiences less difference between basic and smart cache (see Table. 8). Thus, an important parameter in QoE experience in terms of average latency is the segment playback duration.

Fig. 10 presents trends on latency for 10s and 15s in the double-client scenarios. In the basic scheme, the moment that the traffic runs on the cache-server link occurs at nearly the same time that the client asks for a segment before segment number 15. In the smart cache scheme, the moment that the traffic runs on the cache-server link occurs at nearly the same time that the client asks for a segment after segment number 15. This

Table 7: Average latency and average bitrate for a single client in basic and smart cache schemes with and without traffic

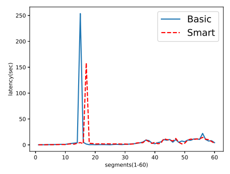
| | Average latency(ms) | | Average bitrate(Mbps) | |
|---|---|---|---|---|
| playback duration | basic | smart | basic | smart |
| 10s-notraffic | 5.43 | 5.20 | 2.79 | 2.81 |
| 15s-notraffic | 7.63 | 6.51 | 2.34 | 2.38 |
| 10s-traffic | 7.58 | 6.10 | 1.64 | 1.90 |
| 15s-traffic | 9.41 | 8.46 | 0.96 | 1.01 |

Table 8: Average latency and average bitrate for double clinets in basic and smart cache schemes with a 10s and 15s

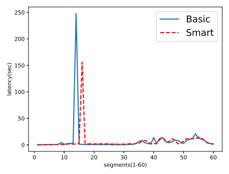
| | Average latency(ms) | | Average bitrate(Mbps) | |
|---|---|---|---|---|
| clients | basic | smart | basic | smart |
| client1-10s | 7.64 | 7.10 | 1.63 | 1.81 |
| client2-10s | 7.23 | 6.82 | 1.47 | 1.78 |
| client1-15s | 9.83 | 8.50 | 0.80 | 1.13 |
| client2-15s | 9.30 | 8.58 | 0.75 | 1.11 |

is since in the basic scheme the first few segments are cache missed. When after 40 sec the background traffic is activated on the link, smart cache is already a few segments ahead of the basic scheme. In the figures, we see that the peak value of latency for the basic scheme is almost 250 sec, while for smart cache, this is around 150 sec. This may be attributed to the smart cache having accurately prefetched more of the requested segments at that time. This latency can change depending on the size of the segment, which means that if the background traffic happens on higher segment numbers, then we will probably observe more latency and more difference between basic and smart cache schemes. If the traffic happens in the beginning of the requests, then the difference will not be significant because the sizes of the segments are smaller.

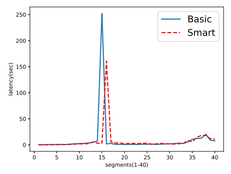
2) Average Bitrate Gain: The bitrate metric study shows that for all scenarios,



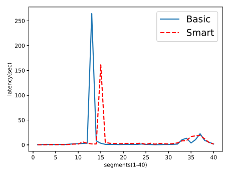
(a) First client 10s



(b) Second client 10s



(c) First client 15s

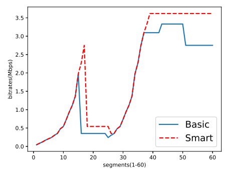


(d) Second client 15s

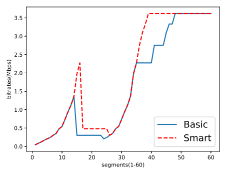
Figure 10: Latency: smart vs. basic.

smart cache shows higher gain than the basic scheme (see Table. 7 and Table. 8). Also, the 15s segment duration video shows lower bitrate gain than 10s. Fig. 11 shows additional details on the requested bitrate of those clients. The first peaks in each of the plots are related to the moment when the bitrate already hits the link throughput threshold for ramp-up phase. In a single-client with no background traffic, this peak would be higher than other scenarios, because there is no traffic to stop it and throttle the link throughput. We observe that there is a sudden drop after 40 sec, and the throughput measurements thereafter stay steady on the minimum bitrate. After the bottleneck is taken away, the bitrate values will have sharp upshifts and this trend functions more smoothly as the two clients are still running and competing for link resources. The blue line (basic) is behind or underneath the red line (smart). In 10s, the gap between two schemes is more moderate than that of 15s; this relationship is also displayed in numeric values in the tables. smart cache compared to basic in the double-client scenario with 15s displays more gain than 10s.

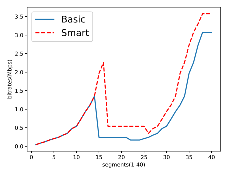
3) Bitrate Switching: In terms of bitrate switching frequency, we did not observe any changes in measurement for the single-client scenario without background traffic. For single-client scenarios with background traffic, the changes are also insignificant. For



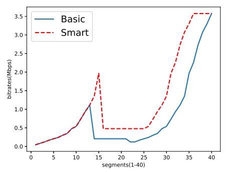
(a) First client 10s



(b) Second client 10s



(c) First client 15s



(d) Second client 15s

Figure 11: Bitrate Gain: samrt vs. basic.

Table 9: Average bitrate switching: double clients

| clients | basic | smart |
|---|---|---|
| Double-10s-client1 | 31.8 | 32.4 |
| Double-10s-client2 | 32.2 | 32.0 |
| Double-15s-client1 | 29.0 | 28.6 |
| Double-15s-client2 | 30.4 | 26.4 |

double-client scenarios, the values for bitrate switching is almost the same except for 15s, where smart cache has less bitrate switching compared to the basic scheme. These insignificant changes could be due to the fact that the changes we implement are in the cache itself, and not in the client ABR (see Table. 9). In cases when the smart cache has higher bitrate switching than the basic scheme, this is related to upshifts. The experience of users is more negatively affected by downshift switchings than it is by upshift switchings [15]. [18] shows that with larger segment playback duration, frequent bitrate switching is not significantly worse than videos with less bitrate switches.

Key Observations: We now summarize key observations with double-clients based on our study:

- Cache Hitrate: For 10s, smart cache increases the hitrate by 52% and for 15s smart cache increases the hitrate by 76% compared to the basic scheme.

- Client Throughput: With smart cache for 10s, the throughput measurement of client1 and client2 increase 33.21% and 43.7%, respectively, compared to the basic scheme. This value for 15s double-client for client1 and client2 increases to 24.9% and 12.55%, respectively.

- Delay: With smart cache for 10s, delay for client1 decreases by 7.07% while client2 delay decreases by 5.67% compared to the basic scheme. For 15s with smart cache, delay decreases for client1 and client2 by 13.53% and 7.74%, respectively.

- Bitrate Gain: With smart cache for 10s, client1 and client2 bitrate gain increase by 11.04% and 21.09%, respectively, over the basic scheme. For 15s, the gain increases by 41.25% and 48% for client1 and client2, respectively.

- Bitrate Switching: smart cache does not have a noticeable change in bitrate switching for the first and second client in 10s compared to the basic scheme.

- Findumentally, latency and bitrate gain have a direct relationship. If the bitrate gain increases, then the latency also increases. However, smart cache shows less latency increase with higher bitrate gain than the basic scheme.

# CHAPTER 5

# MOVING QOE MODELS

In this Chapter, we present our two moving QoE models: MQoE_RF (Rate Frequency) and MQoE_SD (Standard Deviation). Our first moving QoE model, MQoE_RF (see Fig. 12), considers two QoE metrics: bitrate gain and bitrate switching frequency. Bitrate gain reflects increase in quality to the end users, it has a positive impact on the QoE. However, if bitrate switching frequency happens frequently, the end user may be displeased; that is, this factor negatively impacts overall quality of experience. Thus, our model rewards the bitrate gain. On the other hand, our model penalizes bitrate switching frequency metric. To consider the effect of bitrate switching frequency (count of switches) as a separate metric on QoE, we should consider this metric in a non-linear model. Since our model focuses on a multi-client environment, both these metrics are considered in terms of aggregation among all the clients. Secondly, to address for in-flight QoE estimation, we take a window-based approach to provide QoE periodically during a video session.

To consider the nonlinear relationship between the two metrics over multiple clients and to modestly penalize for bitrate switching, our model considers adding the average bitrates received by all the clients, which is divided by the sum of the exponential smoothing values of the bitrate switching frequency of all clients in each window adjusted by a weight. The denominator in Equation. 5.1 also accounts for the situation if there are

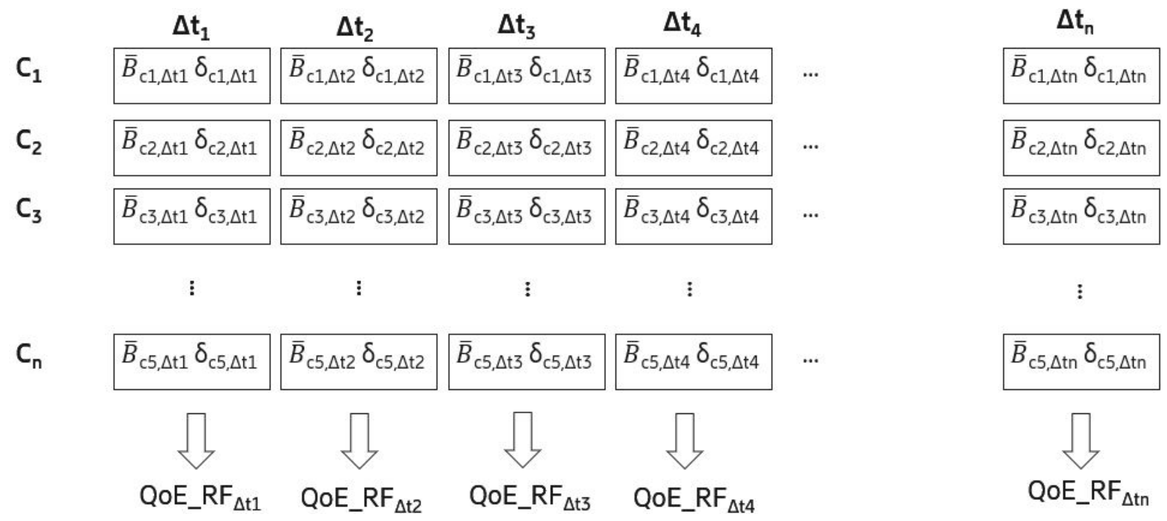

$\Delta t_1$
$\Delta t_2$
$\Delta t_3$
$\Delta t_4$
$\Delta t_n$
$C_1$
$C_2$
$C_3$
$C_n$
$\bar{B}_{c1,\Delta t1}\ \delta_{c1,\Delta t1}$
$\bar{B}_{c1,\Delta t2}\ \delta_{c1,\Delta t2}$
$\bar{B}_{c1,\Delta t3}\ \delta_{c1,\Delta t3}$
$\bar{B}_{c1,\Delta t4}\ \delta_{c1,\Delta t4}$
$\bar{B}_{c1,\Delta tn}\ \delta_{c1,\Delta tn}$
$\bar{B}_{c2,\Delta t1}\ \delta_{c2,\Delta t1}$
$\bar{B}_{c2,\Delta t2}\ \delta_{c2,\Delta t2}$
$\bar{B}_{c2,\Delta t3}\ \delta_{c2,\Delta t3}$
$\bar{B}_{c2,\Delta t4}\ \delta_{c2,\Delta t4}$
$\bar{B}_{c2,\Delta tn}\ \delta_{c2,\Delta tn}$
$\bar{B}_{c3,\Delta t1}\ \delta_{c3,\Delta t1}$
$\bar{B}_{c3,\Delta t2}\ \delta_{c3,\Delta t2}$
$\bar{B}_{c3,\Delta t3}\ \delta_{c3,\Delta t3}$
$\bar{B}_{c3,\Delta t4}\ \delta_{c3,\Delta t4}$
$\bar{B}_{c3,\Delta tn}\ \delta_{c3,\Delta tn}$
$\bar{B}_{c5,\Delta t1}\ \delta_{c5,\Delta t1}$
$\bar{B}_{c5,\Delta t2}\ \delta_{c5,\Delta t2}$
$\bar{B}_{c5,\Delta t3}\ \delta_{c5,\Delta t3}$
$\bar{B}_{c5,\Delta t4}\ \delta_{c5,\Delta t4}$
$\bar{B}_{c5,\Delta tn}\ \delta_{c5,\Delta tn}$
QoE_RF$_{\Delta t1}$
QoE_RF$_{\Delta t2}$
QoE_RF$_{\Delta t3}$
QoE_RF$_{\Delta t4}$
QoE_RF$_{\Delta tn}$

Figure 12: MQoE_RF for multiple clients.

no switching in a window, in which case the QoE model can still be computed based on the average bitrate.

If we set $\Delta t$ to be the window and the number of active clients to be $C$, then our MQoE_RF model can be written as (see Table. 10 for the complete list of notations):

$$MQoE\_RF_{C,\Delta t} = \frac{\frac{1}{C}\left(\sum_{c=1}^{C} \overline{B_{c,\Delta t}}\right)}{1 + \frac{\frac{1}{C}\left(\sum_{c=1}^{C} \delta_{c,\Delta t}\right)}{\gamma}}. \tag{5.1}$$

Here, $\overline{B_{c,\Delta t}}$ represents the average bitrate for client $c$ during the window $\Delta t$, and $\delta_{c,\Delta t}$ represents exponential moving average on bitrate switching frequency, given by:

$$\delta_{c,\Delta t} = (1-\nu) * \delta_{c,\Delta t-1} + \nu N_{c,\Delta t}. \tag{5.2}$$

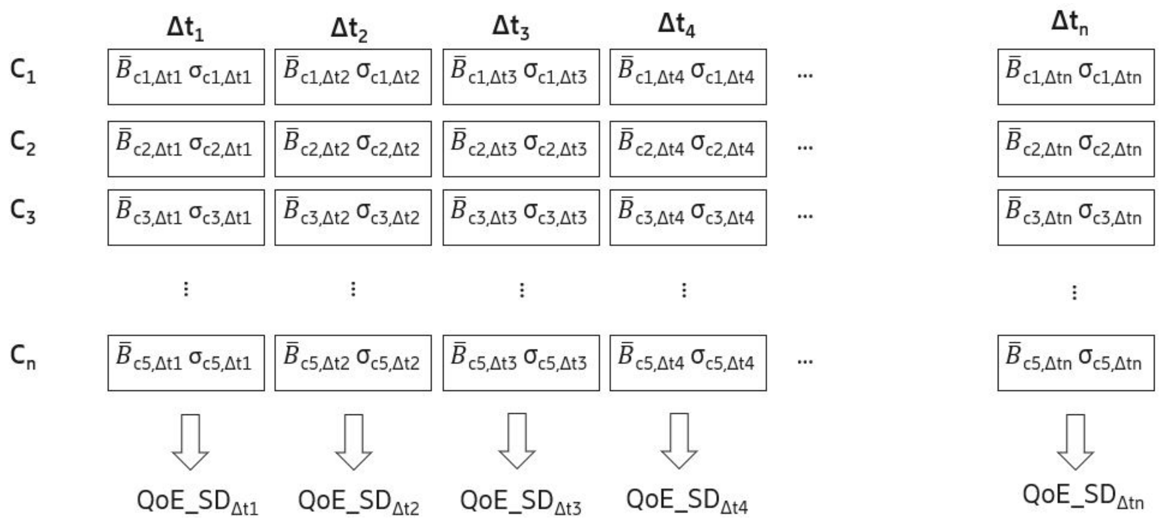
We now explain further the rationale behind our choices. Our assumption is that

| variable | description |
|---|---|
| $B$ | bitrate |
| $\overline{B}$ | average bitrate |
| $K$ | Number of segment |
| $C$ | Number of active clients |
| $\Delta t$ | time window |
| $\delta$ | exponential smoothing of the switching frequency |
| $N$ | number of switching frequency |
| $\sigma$ | standard deviation |
| $\nu$ | exponential smoothing weight ($0 \leq \nu \leq 1$) |
| $\alpha$, $\beta$, $\gamma$ | weights |

Table 10: Notations used in QoE models.

window $\Delta t$ is a reasonable time window for measurements (see Chapter 6 or further discussion). Then, $\overline{B_{c,\Delta t}}$ represents the average of all the DASH segments for client $c$ in $\Delta t$. The switching frequency factor in the denominator is exponentially smoothed to even out any large oscillatory behavior during window $\Delta t$. The parameter $\gamma$ acts as a scaling parameter on this bitrate switching frequency. Finally, to account for the possibility of no switching frequency, especially over multiple windows, that could lead to $\delta_{c,\Delta t}$ being nearly zero, we have added one in the denominator in Equation. 5.1 as the final stabilization factor in our MQoE_RF model.

Our second moving QoE model, MQoE_SD (see Fig. 13), differs from MQoE_RF (Equation. 12) in that it is a linear model that relates the bitrate magnitude with the change arising in bitrates. That is, the first term in MQoE_SD 5.3 is the same as the numerator in MQoE_RF 5.1, which reflects the average of bitrates among all the clients. The standard deviation of the bitrates of all the clients in a window can be an aggregated value to capture the switching magnitude during a window; this term is then submitted from the



$\Delta t_1$
$\Delta t_2$
$\Delta t_3$
$\Delta t_4$
$\Delta t_n$
$C_1$
$C_2$
$C_3$
$C_n$
$\bar{B}_{c1,\Delta t1}\ \sigma_{c1,\Delta t1}$
$\bar{B}_{c1,\Delta t2}\ \sigma_{c1,\Delta t2}$
$\bar{B}_{c1,\Delta t3}\ \sigma_{c1,\Delta t3}$
$\bar{B}_{c1,\Delta t4}\ \sigma_{c1,\Delta t4}$
$\bar{B}_{c1,\Delta tn}\ \sigma_{c1,\Delta tn}$
$\bar{B}_{c2,\Delta t1}\ \sigma_{c2,\Delta t1}$
$\bar{B}_{c2,\Delta t2}\ \sigma_{c2,\Delta t2}$
$\bar{B}_{c2,\Delta t3}\ \sigma_{c2,\Delta t3}$
$\bar{B}_{c2,\Delta t4}\ \sigma_{c2,\Delta t4}$
$\bar{B}_{c2,\Delta tn}\ \sigma_{c2,\Delta tn}$
$\bar{B}_{c3,\Delta t1}\ \sigma_{c3,\Delta t1}$
$\bar{B}_{c3,\Delta t2}\ \sigma_{c3,\Delta t2}$
$\bar{B}_{c3,\Delta t3}\ \sigma_{c3,\Delta t3}$
$\bar{B}_{c3,\Delta t4}\ \sigma_{c3,\Delta t4}$
$\bar{B}_{c3,\Delta tn}\ \sigma_{c3,\Delta tn}$
$\bar{B}_{c5,\Delta t1}\ \sigma_{c5,\Delta t1}$
$\bar{B}_{c5,\Delta t2}\ \sigma_{c5,\Delta t2}$
$\bar{B}_{c5,\Delta t3}\ \sigma_{c5,\Delta t3}$
$\bar{B}_{c5,\Delta t4}\ \sigma_{c5,\Delta t4}$
$\bar{B}_{c5,\Delta tn}\ \sigma_{c5,\Delta tn}$
QoE_SD$_{\Delta t1}$
QoE_SD$_{\Delta t2}$
QoE_SD$_{\Delta t3}$
QoE_SD$_{\Delta t4}$
QoE_SD$_{\Delta tn}$

Figure 13: MQoE_SD for multiple clients.

bitrate term based on a weight. Thus, our second model, MQoE_SD, can be written as:

$$MQoE\_SD_{C,\Delta t} = \frac{1}{C}\left(\sum_{c=1}^{C} \overline{B_{c,\Delta t}}\right) \quad - \alpha \cdot \frac{1}{C}\left(\sum_{c=1}^{C} \sigma(B_{c,\Delta t})\right), \tag{5.3}$$

where $\sigma(B_{c,\Delta t})$ represents the standard deviation on bitrates in window $\Delta t$ for client $c$. To contrast our moving QoE models, consider next a static QoE model such as the original MPC QoE model [58]:

$$MPC\_QoE = \sum_{k=1}^{K} B_k - \beta \cdot \sum_{k=1}^{K-1} |B_k - B_{k-1}|, \tag{5.4}$$

where $B_k$ is the bitrate of segment $k$ while $K$ is the total number of segments in a video. Note that such static QoE models consider all segments being delivered in their QoE calculation. Before we discuss how a static model could be adapted for moving QoE determination, we point out that we kept the two most dominant terms from [58]: bitrates of segments and differences in bitrate from one segment to the next from the original MPC QoE model. We do not consider two terms from the original MPC QoE model: the startup delay and rebuffering. The startup delay is an issue only at the beginning of a video session. Since we are considering a windowed scenario for moving QoE, this term is relevant only at the beginning in the initial window, but no longer in any other windows and it is, thus, irrelevant for moving QoE and it is ignored. The other term, rebuffering, was found to have very minimal effects in a recent extensive study with video streaming [56]; thus, we also do not consider this term (we will comment on rebuffering later in Chapter 6 based on our study and point out how rebuffering is indirectly captured by our MQoE models). Since Equation. 5.4 for observing QoE is for an entire video, we adapt it for use in each window by computing for the segments transferred in each window. Assume that in the window $\Delta t$, the number of segments is $K_{\Delta t}$, the MPC QoE model in the window $\Delta t$ for client $c$ can be rewritten as:

$$MPC\_QoE_{c,\Delta t} = \sum_{k=1}^{K_{\Delta t}} B_k^c - \beta \cdot \sum_{k=1}^{K_{\Delta t}-1} |B_k^c - B_{k-1}^c|. \qquad (5.5)$$

Thus, over the set of all clients in window $\Delta t$, we get the following moving MPC-based QoE model, which we refer to as MQoE_MO:

$$MQoE\_MO_{C,\Delta t} = \frac{1}{C} \sum_{c=1}^{C} \left( \sum_{k=1}^{K_{\Delta t}} B_k^c \qquad - \beta \cdot \sum_{k=1}^{K_{\Delta t}-1} |B_k^c - B_{k-1}^c| \right). \qquad (5.6)$$

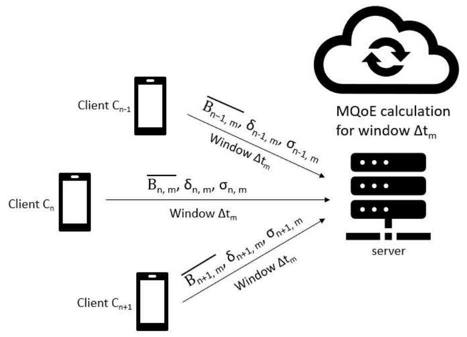

Client $C_{n-1}$
$\overline{B_{n-1,m}}, \delta_{n-1,m}, \sigma_{n-1,m}$
Window $\Delta t_m$
MQoE calculation for window $\Delta t_m$
Client $C_n$
$\overline{B_{n,m}}, \delta_{n,m}, \sigma_{n,m}$
Window $\Delta t_m$
server
Client $C_{n+1}$
$\overline{B_{n+1,m}}, \delta_{n+1,m}, \sigma_{n+1,m}$
Window $\Delta t_m$

Figure 14: MQoE Architecture with thee clients $n-1$, $n$, $n+1$ for window $m$.

## 5.1 Diagram Architecture for MQoE

MQoE is an aggregated value among multiple clients and it can be reported every $\Delta t$ seconds (See Fig. 14). During a video session, there are multiple MQoE values, which are calculated each time in a centralized point in the network. If an ISP or content provider wants to implement MQoE monitoring model, the DASH client protocol would not be modified for implementation. The users are blind to this monitoring process. The only task that each client suppose to do, independently of what DASH protocol with whatever configurations is using, is to send the information about the received bitrates in a window duration of $\Delta t$ ($\overline{B_{c,\Delta t}}$, $\sigma(B_{c,\Delta t})$, $\delta_{c,\Delta t}$) to a controller. This controller that is set up by the providers, receives all the information of each client in each window $\Delta t$. The controller is responsible to implement the MQoE model on the received data (it can be MQoE_RF, MQoE_SD, or MQoE_MO). In Fig. 14, the controller is on the server for ease of understanding. It shows that three clients $n-1$, $n$, $n+1$ send information at window $m$.

# CHAPTER 6

# MOVING QOE MODEL EVALUATION

## 6.1 Environment Study

To study the MQoE models, we implemented our study environment on the GENI testbed [3], in which clients access a video from a DASH video server. In this environment, we allow multiple clients to simultaneously access the same video to emulate watching the same live event by a number of users. The raw link bandwidth was set to 10 Mbps. DASH clients were implemented in Python and the video server was based on Apache HTTP server. For ABR scheme used by the clients is the dynamic/hybrid ABR algorithm that applies both throughput and buffer signal for bitrate selection based on [1].

For our study, we used *Big Buck Bunny* (BBB) and *Elephants Dream* (ED) [28], two well-known DASH video datasets, which consist of 150 and 164 segments, respectively. Each of these videos has 20 bitrate representations, ranging form 0.045 Mbps at the lowest resolution for both of the datasets to 3.936 Mbps and 4.066 Mbps at the highest resolution for *Big Buck Bunny* and *Elephants Dream*, respectively. The twenty representations and the gaps between the representations sequence in both datasets are

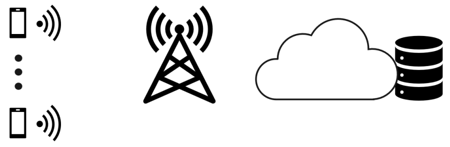


Figure 15: mobile network topology

very similar. Each segment was of 4 sec playback duration. Thus, the entire Big Buck Bunny video is 10 min long and the entire *Elephants Dream* video is 11 min. While the two datasets have very similar bitrate representations, the sizes in terms of bytes of each segment in a dataset with a specific representation is not the same as the other dataset. In Fig. 16, we show the segment sizes (in MB) of the highest bitrate codec representation for each dataset to illustrate this point. For example, for first few segments of *Big Buck Bunny* with highest bitrate of 3.936 Mbps has a significant larger size than the first few segments of *Elephants Dream* highest bitrate of 4.066 Mbps. We observe similar differences in the other segments multiple times. The main implication of this observation is that even if each segment received is from the highest representation, the bitrates are not related to the segment sizes in bytes. Consequently, MQoE observed for user watching Big Buck Bunny would be different than for *Elephants Dream* since the ABR algorithm depends on the throughput as a factor, assuming all other factors being equal.

To emulate the effect on the clients to experience mobility while traveling and connected to a wireless network environment as shown in Fig. 15, we used traces on path behavior dataset provided by Riser et al. [45] while traveling by a car, a train, or a ferry. We installed Wondershaper [4] on the server machine to throttle the link with these traffic traces. As noted in [45], with a car, there are frequent fluctuations on bandwidth availability and also dead time when the signal is not available. On the other hand, due to length of the videos we studied, our study faces the frequent fluctuation part, but never reached the dead time mentioned in [45]. With the train trace, there is a large drop in bandwidth availability at one point during our study, which captures essentially the dead

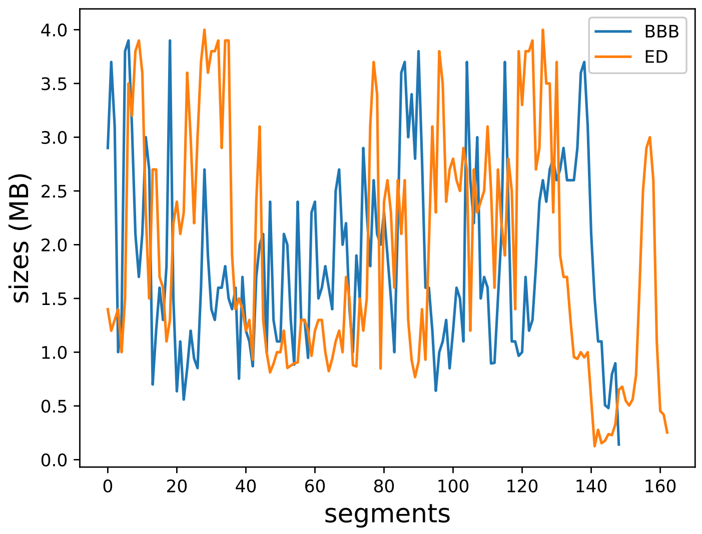


Figure 16: Segments in megabytes: BBB vs. ED for the highest representation

time. Finally, for the ferry trace, the ferry was going from one shore to another shore; thus, the bandwidth availability continued to gradually drop as the ferry moved away from the departing shore. Due to the duration of the videos we studied, the bandwidth availability was just about to start to ramp up due to signal strength improving from the other shore when our videos ended. Thus, the three traces gave us different perspectives on mobility patterns during the video streaming period and it is instructive to keep this in mind. We conducted our study in this setting with multiple clients ranging from three clients to ten clients. Finally, MQoE_MOdels for multiple clients are presented by normalizing based on the maximum value of the MQoE MOdel for a single client for the associated traffic trace.

## 6.2 Initial Analysis and Adjustment

Our initial analysis centered around determining window $\Delta t$, weights associated with our QoE models, and the limitation of MQoE_MO for use in moving QoE monitoring. Our first experimentation was to determine the window duration $\Delta t$ by varying the window size. We observed that with a small window size, there are many windows that have no switching. Sometimes with a higher traffic that causes latency in the client request, no segment might be transmitted if it is a very tiny window. Based on our initial trials, the window size of 60 sec was found to be a good window size that has a reasonable number of segments and switching in each window. The window size was kept at a constant time duration for the rest of our study. Then, in all our study, we consider the first window for $\Delta t$ to be the ramp-up window; thus, we will focus on results from window 2 to window 10 for both the videos. In addition, we can observe ramp-up of the frequency in the second window as we used exponential smoothing with previous value from the first window.

For use in Equation. 5.1, exponential smoothing in switching frequency was used as shown in Equation. 5.2. We found that setting $\nu = 0.75$, which gives more weight to the newest value of switching frequency, allows us a level of relative stability while capturing the changes. Thus, this value was used in the rest of our study. We considered a number of different values for both $\gamma$ and $\alpha$ used in 5.1 and 5.2, respectively. The combination of $\gamma$ and $\alpha$ are categorized into three sets: set0: $\gamma = 25$ and $\alpha = 0.5$; set1: $\gamma = 10$ and $\alpha = 1$; set2: $\gamma = 5$ and $\alpha = 1.5$.This is summarized in Table 11.

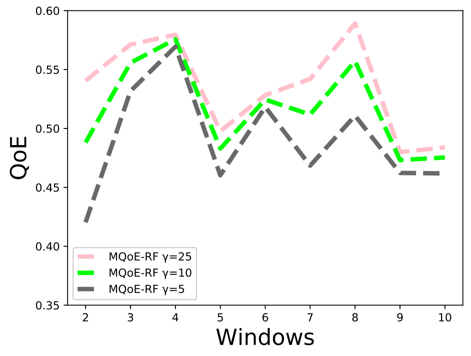
We found that when $\gamma = 25$ (from set0), the MQoE_RF behavior is similar to the

| sets | $\gamma$ | $\alpha$ |
|---|---|---|
| set0 | 25 | 0.5 |
| set1 | 10 | 1.0 |
| set2 | 5 | 1.5 |

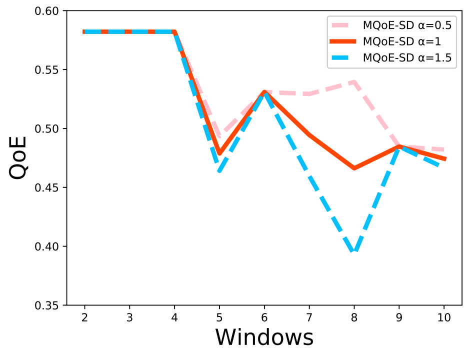
Table 11: Sets of weight parameters for MQoE models

bitrates received by the clients; i.e., the bitrate term in the numerator is the dominant term in 5.1 for MQoE_RF at this value of $\gamma$. As we reduced $\gamma$ from 25 to 5, we noticed that the MQoE behavior changes to the point of being more fluctuating, while giving higher weights to the denominator in Equation. 5.1; Thus, we show the graphs for three values of $\gamma$ at 25, 10, and 5 in Fig. 17a with three clients in car. Similarly, we found that when $\alpha = 0.5$ (from set0), then the MQoE_SD behavior in Equation. 5.3 is similar to the bitrates received by the clients. When $\alpha$ increases to 1.5, the MQoE behavior changes to the point where QoE has higher fluctuation because it is giving more weights to the second term in Equation. 5.3. Thus, for $\alpha$, three values 0.5, 1.0, 1.5 are shown in Fig. 17b for three clients in car. From this discussion, it is clear that set0, i.e., $\alpha = 0.5$ and $\gamma = 25$ essentially reflects the same behavior as bitrates. The parameters, $\gamma$ and $\alpha$, are self-learned parameters. We note that $\gamma$ between 5 and 25 and $\alpha$ between 0.5 and 1.5 are the useful ranges for these parameters to account for the associated term. Below or above these ranges, we observe asymptotic behavior that would not give us any new information.

Consider next the MQoE_MO model 5.6. We found that the straightforward extension of the MPC QoE model 5.4 to moving QoE model 5.6 is problematic at times. Consider again Fig. 18, which also includes MQoE_MO on the graph. From window 8 to 10, the MQoE_MO value dropped by 57.0% while the bitrate drop was only 20.03%



(a) MQoE_RF



(b) MQoE_SD

Figure 17: Three clients (car) with different values of $\alpha$ and $\gamma$ (for BBB)

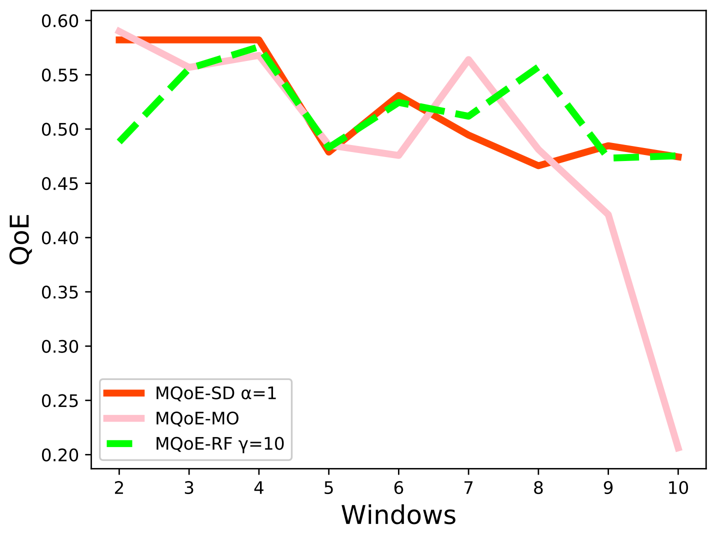


Figure 18: Three clients/Car, Moving QoE models MQoE_RF, MQoE_SD and MQoE_MO (for BBB)

(see Fig. 19a). On further investigation, we found that since the number of segments transmitted during a window can vary (depending on the network condition), it may also be possible that this number can be quite low as it so happened in window 10. With 5.6, a large drop is possible in the moving QoE in a particular window. This result also illustrates that a static QoE model is not readily usable as a moving QoE model. Thus, for moving QoE, new models as we proposed here are necessary. In the rest of the paper, we simply focus on our MQoE models 5.1 and 5.3, and for two sets of $\alpha$ and $\gamma$ parameters values: set1 and set2.

## 6.3 Comparative Study

By considering three mobility traces based on car, train and ferry, our study focuses on three dimensions: 1) to establish the behavior of the MQoE models as the number of clients is varied, 2) to understand how the models are impacted as we consider two different videos, and 3) perceived QoE by clients in a multi-client scenario. As noted earlier, the environment for this study mimics as if the clients are watching a live event. Of the two videos, most of our discussions from our study centers around scenarios for the *Big Buck Bunny* video. We also discuss results for *Elephants Dream*, in certain cases with cars, to show the difference in behavior that is a manifestation of the difference between the two videos in terms of bitrates and segment sizes that we discussed earlier in section Environment Study. For each scenario, we show a set of figures that represents the average bitrate, the bitrate exponential switching frequency, the standard deviation of the bitrate switching magnitude, and the MQoE values for both our models as $\Delta t$ changes.

For some of the scenarios, we present the mean segment size of each window along with maximum and minimum values of each window during the session for all the clients. This will show the main reason of difference in bitrate representation selection for the two datasets and finally the MQoE values while having similar configurations. Then we provide segment-based size and bitrate comparison for the first client of a scenario for the two datasets. Next the results are discussed for each of the three mobility trace scenarios: car, train, and ferry, with more detailed discussions for car. For all trace scenarios, we studied three situations in terms of the number of simultaneous clients: three, five and ten clients.

### 6.3.1 Mobility Trace: Car

#### 6.3.1.1 Three Clients

Consider first three simultaneous clients watching when the *Big Buck Bunny* video is streamed. From Fig. 19a, we see that as we go from window 6 to window 7, the bitrate increases. Along with that, the bitrate switching frequency and bitrate switching magnitude also increase (see Fig. 19b and Fig. 19c), which affect the MQoE models (see Fig. 19d) in different ways.

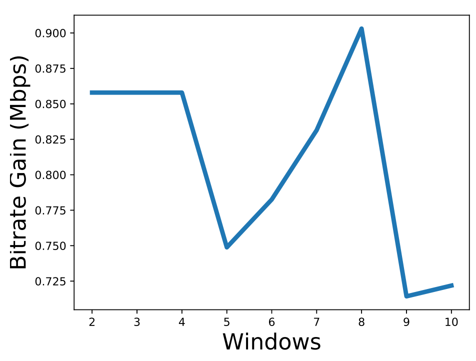
With set1, the bitrate switching frequency and bitrate switching magnitude impact on both models which cause QoE value drop by 2.43% for MQoE_RF and with a larger drop by 6.88% for MQoE_SD. The impact of the magnitude on the linear model is larger than the impact of frequency on the non-linear model for window 7. These changes become more severe when the value of $\alpha$ increase and $\gamma$ decrease where MQoE_RF and

MQoE_SD drop by 9.6% and 13.44% for set2, respectively.

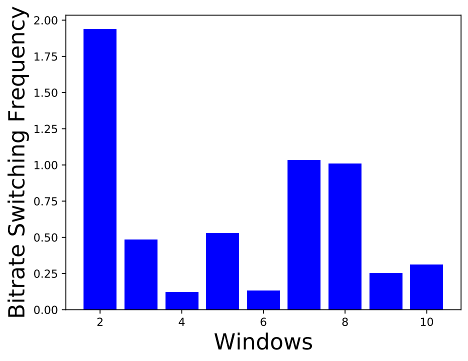
From window 7 to window 8 MQoE_RF with either value of $\gamma$ shows an increasing trend when the bitrate reaches the peak value. It means that for that value of bitrate, the bitrate switching frequency cannot be much significant for the model to take a different trend than bitrate. The value of bitrate switching frequency Fig. 19b did not change notably. However, MQoE_SD decreases and with larger values of $\alpha$ from set2, a larger drop occurs, which is caused by the high value of the bitrate switching magnitude with a higher weight in 5.3.

Note that in the plots on the standard deviation of the bitrate switching magnitude (see Fig. 19c and also in later figures), some windows have values equal to zero. This happens when in a specific window, no bitrate switching occurs. This may also happen if the latency due to congestion on a link is too high that the number of segments in these windows is one or less.

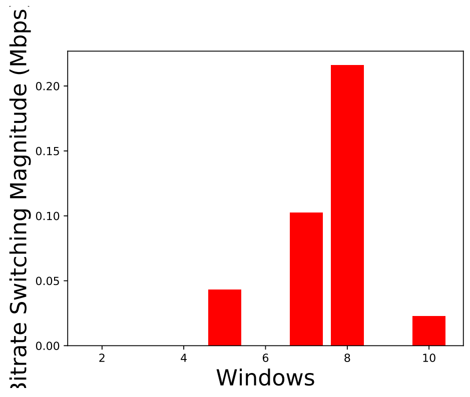
Now if we look at the results for the *Elephants Dream* video (Fig. 20), the metrics, value and behavior on each window were found to be different than with the *Big Buck Bunny* video. Bitrate (see Fig. 20a), unlike the *Big Buck Bunny* (see Fig. 19a), in the first three windows has an increasing trend as the standard deviation on bitrate switching shows some values on those windows. From window 8 to widow 9, while the bitrate decreases, MQoE_SD shows an increasing trend when the standard deviation for bitrate switching decreases. MQoE_RF shows a closer trend to the bitrate as the bitrate switching frequency has almost the same value for these two windows. From window 9 to window 10, the bitrate has an increasing trend; however, MQoE_SD decreases due to the increase



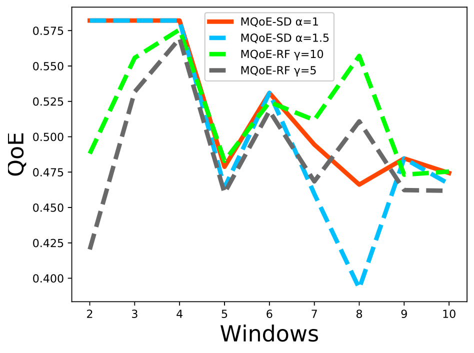
(a) Bitrate



(b) Exponential smoothing values of switching frequency

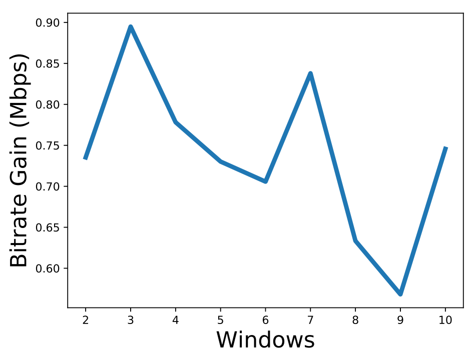


(c) Standard deviation of switching

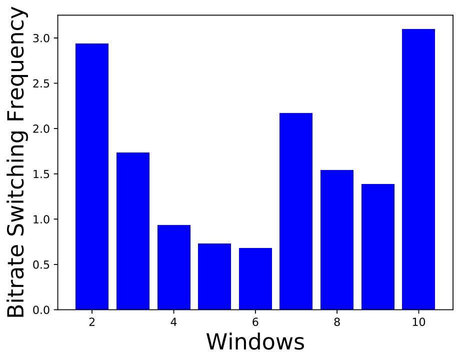


(d) MQoE

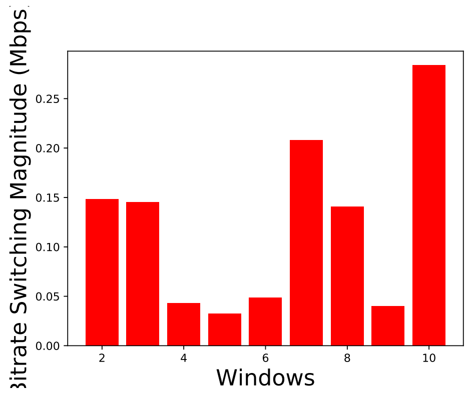
Figure 19: Three clients: car (BBB)



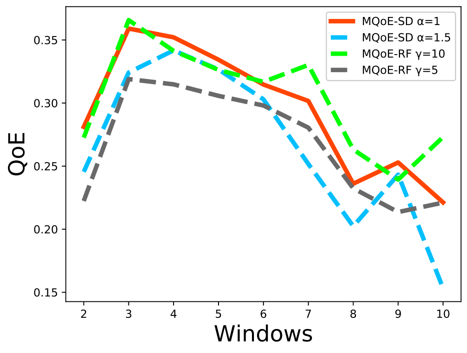
(a) Bitrate



(b) Exponential smoothing values of switching frequency



(c) Standard deviation of switching



(d) MQoE

Figure 20: Three clients: car (ED)

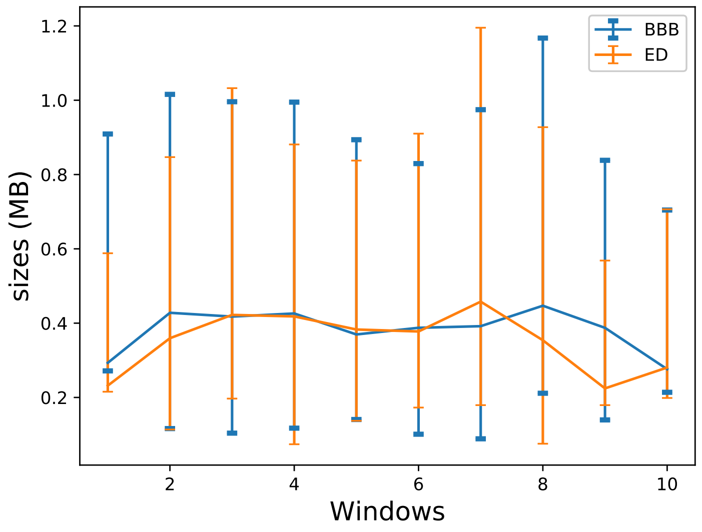


Figure 21: Three clients/Car: size in each window for BBB and ED

in bitrate switching standard deviation by almost six times. The MQoE_RF still shows rise for both sets, although the jump is lower than bitrate jump, as the bitrate switching frequency has been doubled.

We next take a comparative view of the observations from the two different videos. As we mentioned in the previous section, the bitrate representation of the two videos are similar; however, segments at the same level of the bitrate representations vary in sizes (in terms of MB). In other words, similar representations do not mean similar sizes in bytes. The average, maximum and minimum on the sizes in MB for segments transmitted in each window for all clients are shown in Fig. 21. For *Big Buck Bunny*, the average sizes (in MB) from window 3 to window 6 are close to *Elephants Dream*. On the other hand, from the window 1 to window 3 and from the window 7 to window 10, the average sizes for *Big Buck Bunny* is higher than *Elephants Dream*. Table. 12 shows the average values

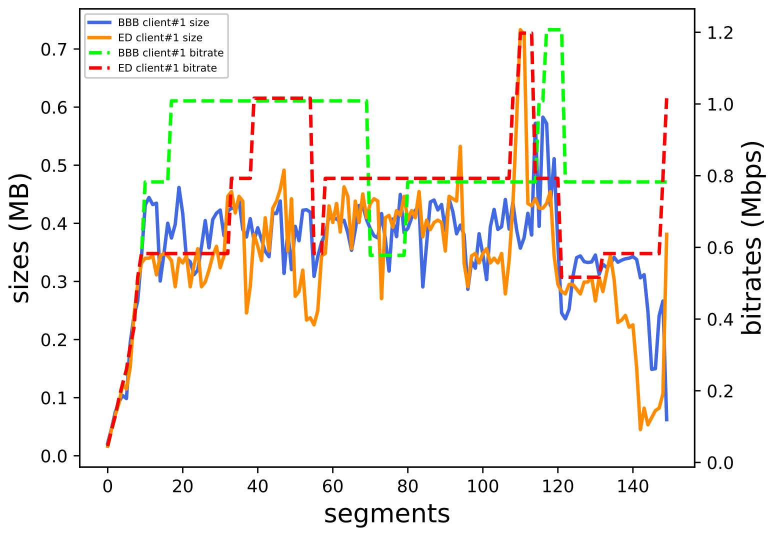


Figure 22: Three clients (car): first client's segments in bytes: BBB and ED

for all the studied cases. This presents the average size of the 10 window for each dataset wherein *Big Buck Bunny* is found to be greater than that for *Elephants Dream* in this for three clients (in car). It is instructive to compare bitrates for each video, shown here for the first client along with segment byte sizes chosen; see Fig. 22. This is to illustrate that the measured MQoE for each video can be noticeably different, which is possible since the ABR algorithm uses throughput (that depends on the bytes transferred) as a factor in deciding the bitrate to choose for the next segment.

#### 6.3.1.2 Five Clients

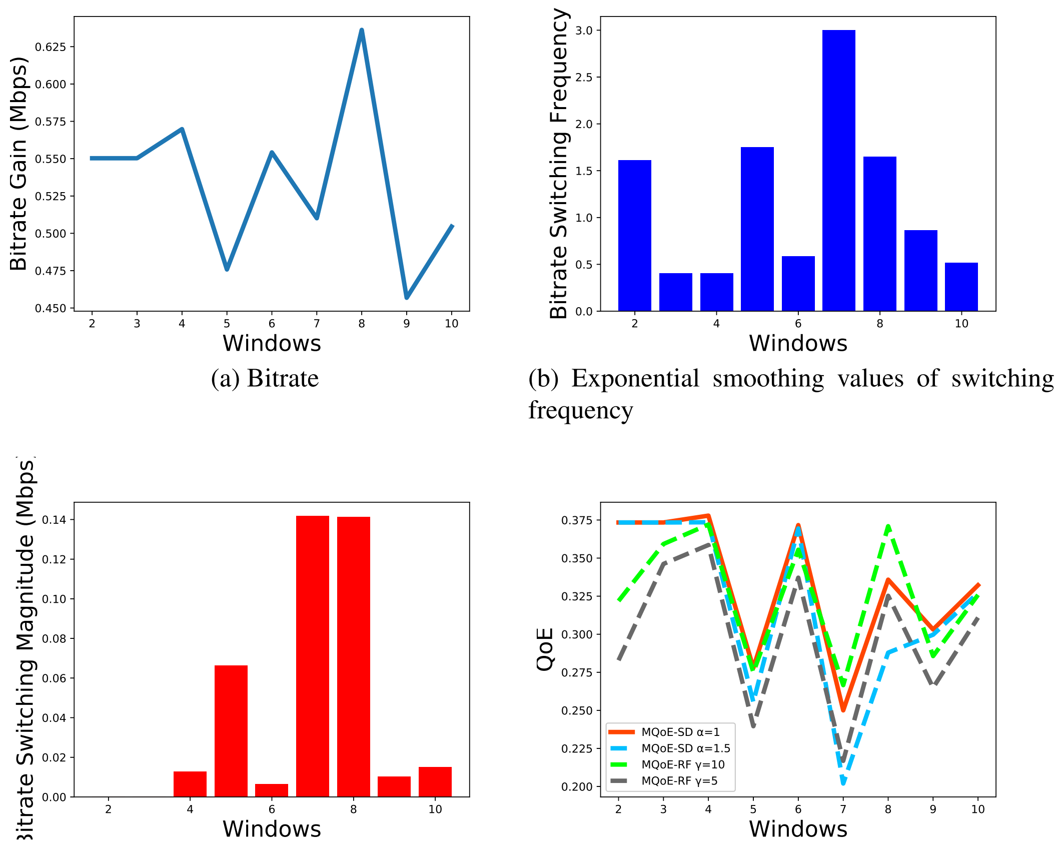
Going from three clients to five clients (see Fig. 23) for *Big Buck Bunny*, we see that bitrates have more swings (compare Fig. 23a to Fig. 19a. Both models follow the pattern of bitrate changes with set1. Same trend can be observed for set2 with MQoE_RF

| Scenario | Average Value BBB | Average Value ED |
|---|---|---|
| Car (3 clients) | 0.38 | 0.35 |
| Car (5 clients) | 0.26 | 0.21 |
| Car (10 clients) | 0.13 | 0.11 |
| Train (3 clients) | 0.27 | 0.34 |
| Train (5 clients) | 0.17 | 0.19 |
| Train (10 clients) | 0.12 | 0.10 |
| Ferry (3 clients) | 0.33 | 0.34 |
| Ferry (5 clients) | 0.24 | 0.22 |
| Ferry (10 clients) | 0.13 | 0.11 |

Table 12: Average values of segment sizes (in MB) for all the case studies.

(where $\gamma = 5$), but from window 8 to 9 this trend changes temporally for MQoE_SD (where $\alpha = 1.5$). The MQoE_RF has higher rise compared to MQoE_SD for both sets of weight parameters as the effect of bitrate switching frequency is smaller than the bitrate switching magnitude. With a larger $\alpha$, we see a shorter rise for MQoE_SD as the effect of bitrate switching magnitude is too high that MQoE_SD does not show a similar trend as bitrate for these windows.

We note that the bitrate value on the window 8 spikes for *Big Buck Bunny* (see Fig. 23a) while for *Elephants Dream*, it shows a drop (see Fig. 24a). From window 9 to window 10, there is rise in the bitate and bitrate switching frequency and magnitude; however MQoE_SD decreased with both sets of parameters. The reason is that from window 9 to window 10 there is a very large rise of bitrate switching magnitude. The QoE value (see Fig.24d) shows smaller drop and rise compared to *Big Buck Bunny* as the value of bitrate switching frequency and standard deviation for *Elephants Dream* change less for sequential windows than the same metrics in *Big Buck Bunny*. The average segment size per window for *Elephants Dream* is less than that for *Big Buck Bunny* (see Fig. 25



(a) Bitrate

(b) Exponential smoothing values of switching frequency

(c) Standard deviation of switching

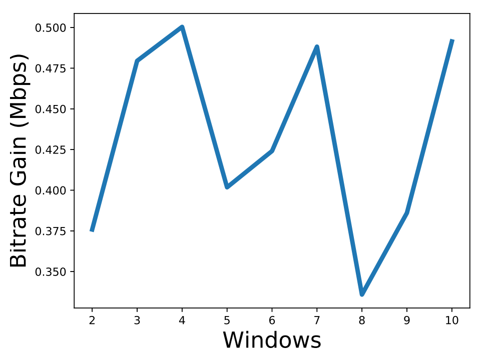
(d) MQoE

Figure 23: Five clients: car (BBB)

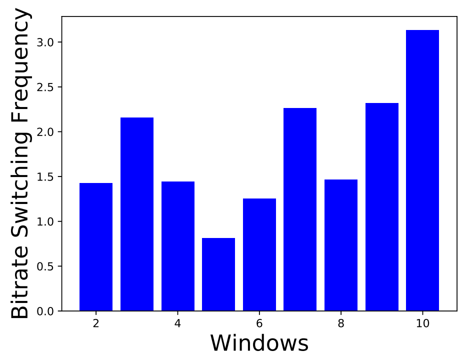
and Table. 12). Fig. 26 shows first client segments in bytes for the two videos when there are five clients.

#### 6.3.1.3 Ten Clients

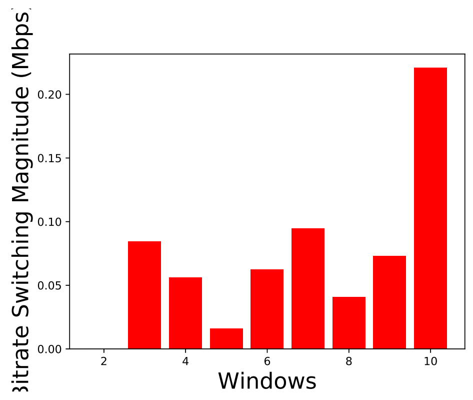
When we go from five clients to ten clients for *Big Buck Bunny* (see Fig. 27), the shape of the bitrate swings is notably different while the range of values are smaller due to higher number of clients. There is a 12.32% rise for MQoE_RF from window 5



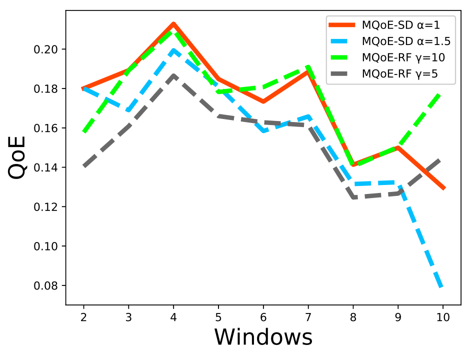
(a) Bitrate



(b) Exponential smoothing values of switching frequency



(c) Standard deviation of switching



(d) MQoE

Figure 24: Five clients: car (ED)

to window 7 with set1. On the other hand, from window 5 to 6, the MQoE_RF with set2 and MQoE_SD with set1 and set2, there is an increase in the MQoE value by 14.17%, 3.69%, 3.55%, respectively; then, from window 6 to window 7, these models decrease by 8.91% 2.75%, 20.76%, respectively. The reason for the drop in MQoE_SD is that the bitrate switching magnitude is much larger in this window compared to previous windows and this value has a significant effect on the linear model 5.3. At window 7, the bitrate switching frequency is also large. However, its effect on the nonlinear model 5.1 is not

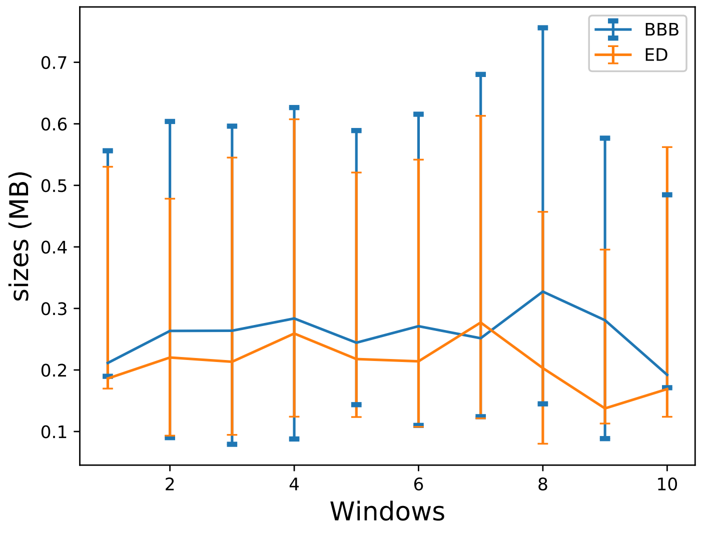


Figure 25: Five clients/Car: window bytes BBB and ED

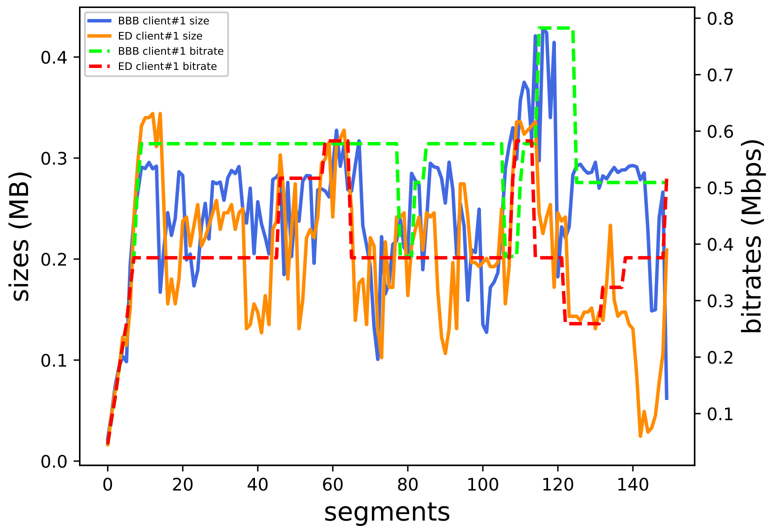
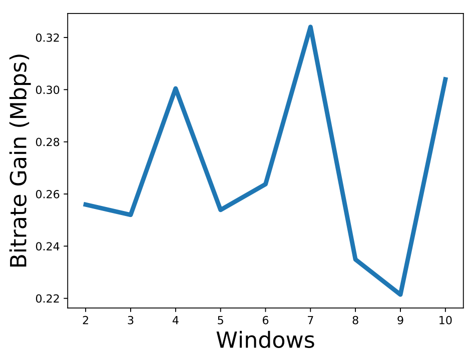
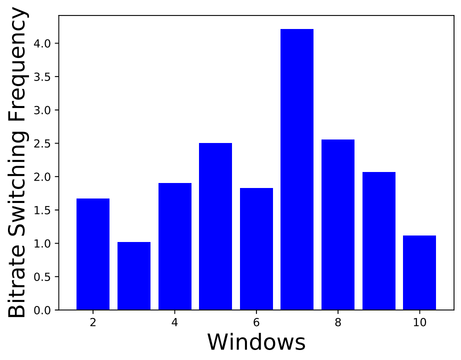


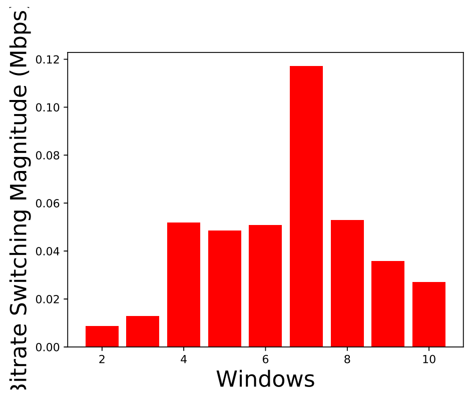
Figure 26: Five clients/Car: first client’s segments in bytes for BBB and ED



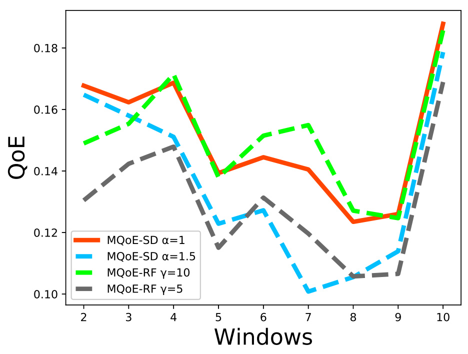
(a) Bitrate



(b) Exponential smoothing values of switching frequency



(c) Standard deviation of switching

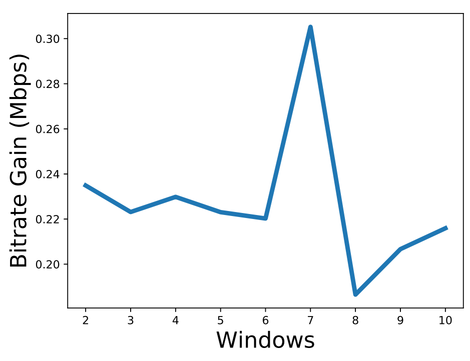


(d) MQoE

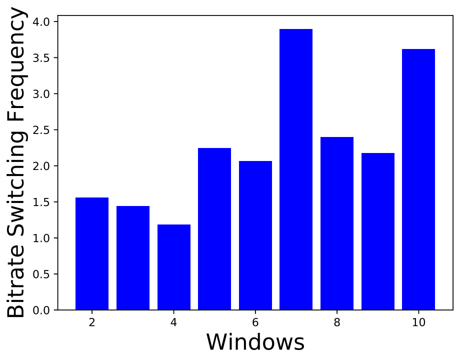
Figure 27: Ten clients: car (BBB)

as significant as the linear model 5.3. This is an illustration of how our QoE models are amenable to capturing the sensitivity due to frequency and the magnitude of the switching.

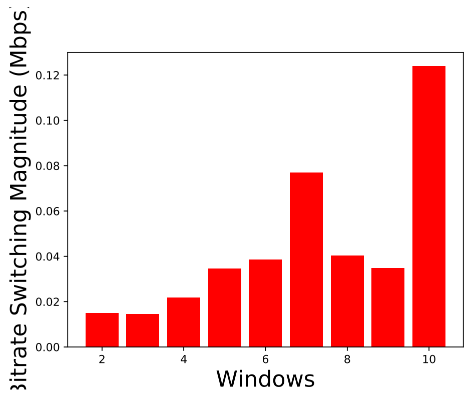
For *Elephants Dream* video, the bitrate has a large rise on window 7 compared to other windows. The two models (see Fig. 28d), within any set of wight parameters, show a very similar behavior while on window 10 they all drop when the bitrate shows a rise. This drop is moderate for MQoE_RF and severe for MQoE_SD. The increase in bitrate switching frequency on window 10 is almost twice that in the window 9 and the



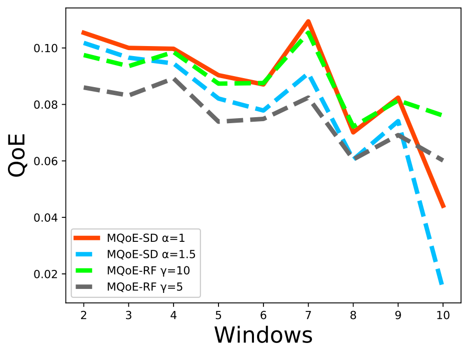
(a) Bitrate



(b) Exponential smoothing values of switching frequency



(c) Standard deviation of switching



(d) MQoE

Figure 28: Ten clients: car (ED)

increase of bitrate switching standard deviation is tripled the value of window 9. Fig. 29 shows that *Big Buck Bunny* has larger size segments in each window than for *Elephants Dream*. In general, the 10-client situation leads to a congested environment, and thus, the differences are minimized due to competing for link resources by all clients.

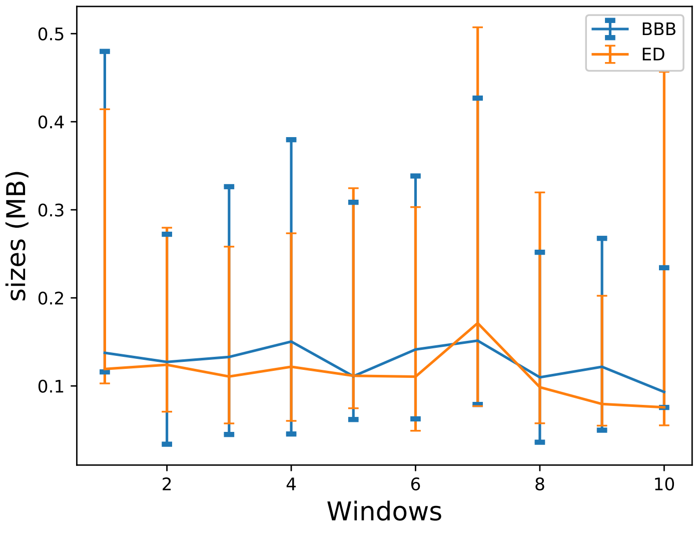


Figure 29: Ten clients (car): window bytes BBB and ED

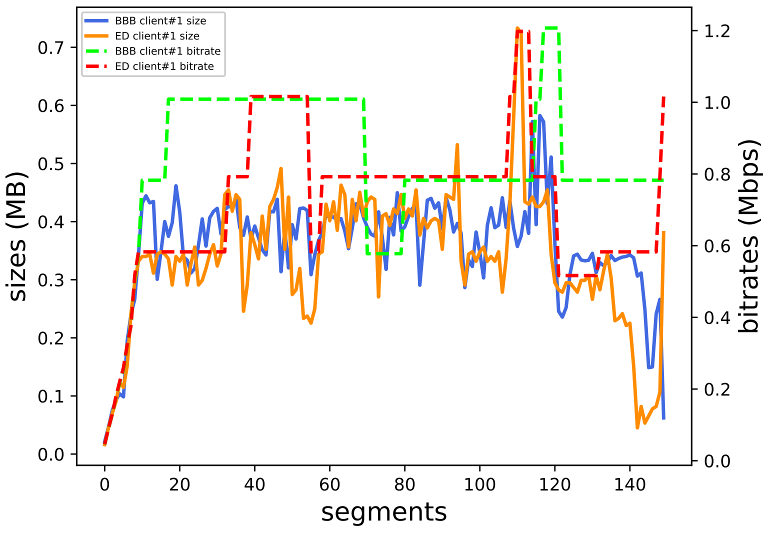


Figure 30: Ten clients (car): first client segment byte BBB and ED

### 6.3.2 Mobility Trace: Train

#### 6.3.2.1 Three Clients

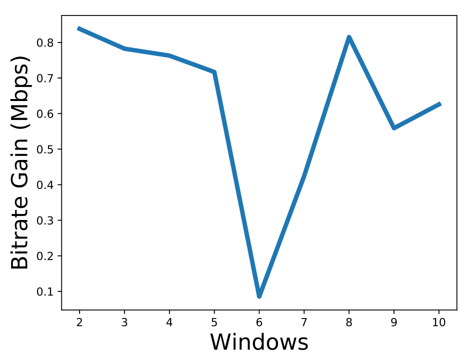
We next consider our study for mobility being from a train with three clients for *Big Buck Bunny*. For this mobility scenario, a large drop in bitrate occurs in the middle due to a drop in bandwidth availability (see Fig. 31a). More specifically, in window 6, the operable bitrate drops significantly; during this window, there is no switching to improve the bitrate. Naturally, QoE for each model also drops in this window. Going from window 6 to window 7, there is a spike in the bitrate, which increases the values for both MQoE moels with set1. More specifically, we see a larger increase in MQoE_RF than MQoE_SD.

When $\alpha$ increases with set2, MQoE_SD takes a downward trend. In window 7, both bitrate switching frequency and bitrate switching magnitude have large value. But, by increasing $\alpha$ for set2, the effect of bitrate switching magnitude on the MQoE_SD model is higher. MQoE_RF and MQoE_SD, were both able to capture the penalty of the bitrate switching frequency and the bitrate switching magnitude.

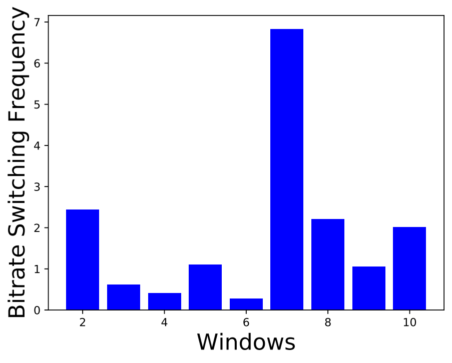
#### 6.3.2.2 Five and Ten Clients

For five clients (Fig. 32 for *Big Buck Bunny*), the overall behavior is similar to that of three clients in most windows. However, with ten clients (Fig. 33), we observe a different shape than three and five clients. From window 6 to window 7, for both set1 and set2, there is a rise for both models along with the rise represented for the bitrate. This increase is larger for MQoE_RF.

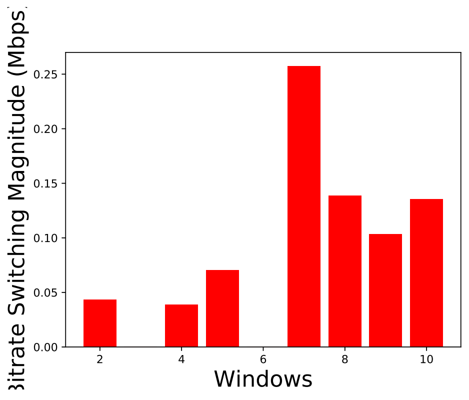
From window 7 to window 8, the bitrate is decreasing slightly. For the model



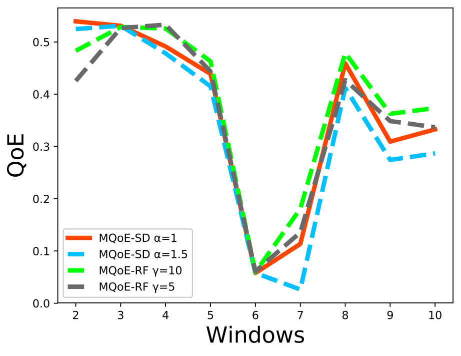
(a) Bitrate



(b) Exponential smoothing values of switching frequency



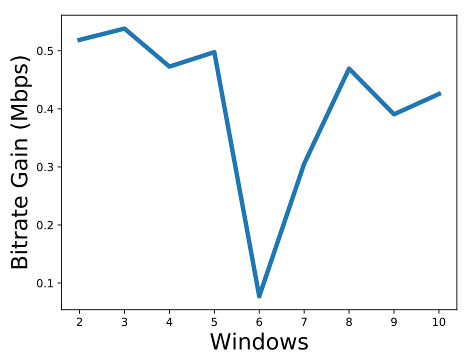
(c) Standard deviation of switching



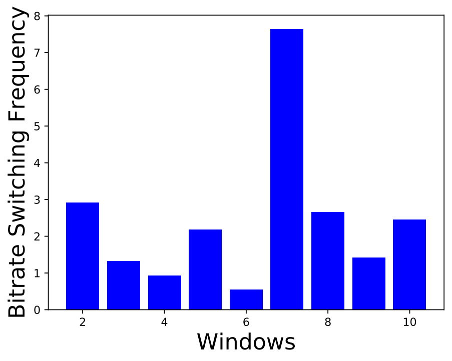
(d) MQoE

Figure 31: Three clients: train (BBB)

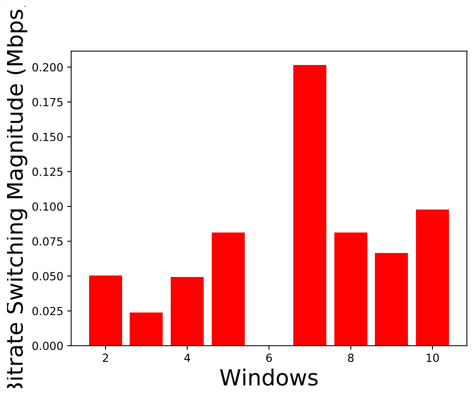
MQoE_RF with set1, we can see that the QoE value decreases while the bitrate frequency switching decreases from window 7 to window 8. The reason is that the model receives the impact of bitrate trend more than the bitrate switching frequency. However, for set2, the MQoE_RF is almost flat as the model receive more impact by bitrate switching frequency. On the other hand, MQoE_SD behaves differently than bitrate. For both sets of the weight parameters, MQoE_SD increases, unlike the bitrate and MQoE_RF. This rise means that the magnitude size is not significant compared to the previous window.



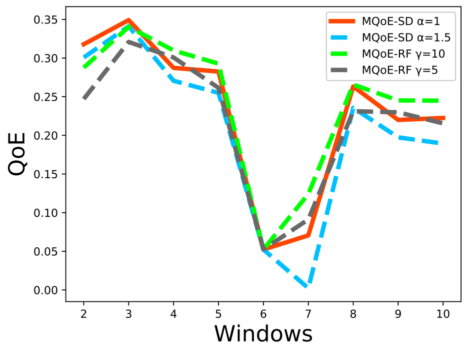
(a) Bitrate



(b) Exponential smoothing values of switching frequency

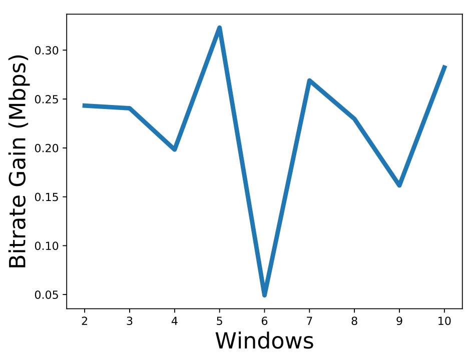


(c) Standard deviation of switching

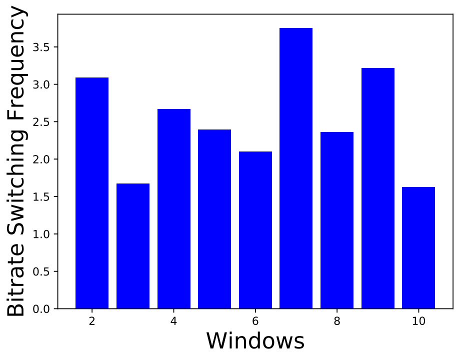


(d) MQoE

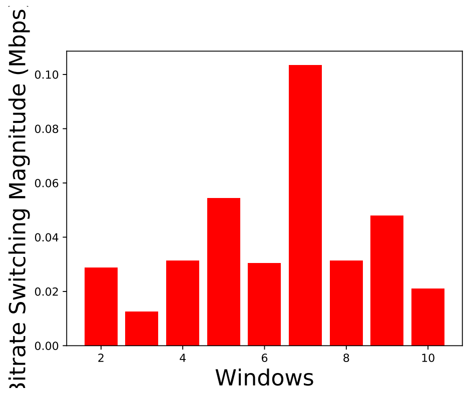
Figure 32: Five clients: train (BBB)



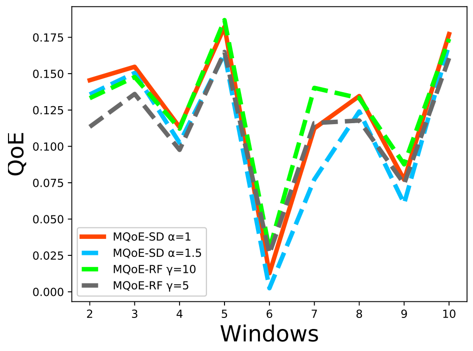
(a) Bitrate



(b) Exponential smoothing values of switching frequency



(c) Standard deviation of switching



(d) MQoE

Figure 33: Ten clients: train (BBB)

#### 6.3.2.3 Comparison between Two Videos

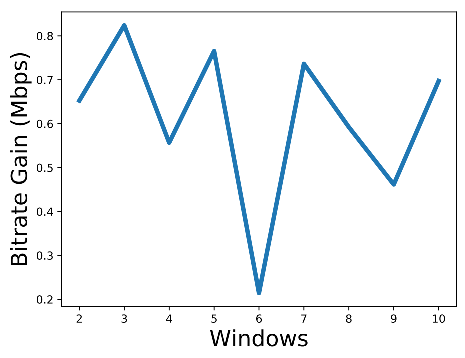
For three, five and ten clients on a train watching the *Elephants Dream* video, MQoE models generally show somewhat similar patterns (see Fig. 34, Fig. 35, Fig. 36). On the other hand, for *Elephants Dream* compared to *Big Buck Bunny*, there is more fluctuation of bitrates and, thus, for MQoEs. Based on Fig. 37, Fig. 38 and Fig. 39 that show sizes on windows, for three clients train the area under the curve for *Elephants Dream* is larger than that for *Big Buck Bunny*. For five clients, the two plots of *Elephants Dream* and *Big Buck Bunny* are going through many changes, however, the overall sizes are not significantly different.

For ten clients train the area under the curve for *Big Buck Bunny* is slightly larger than that for *Elephants Dream* (see Table. 12 for average values).

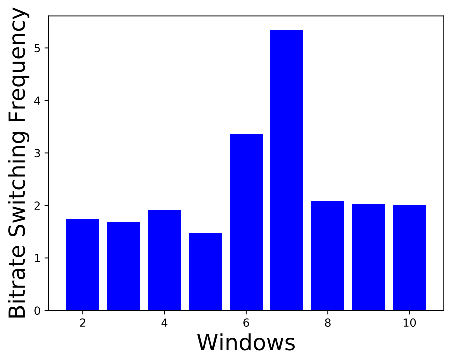
### 6.3.3 Mobility Trace: Ferry

For the case of traveling in a ferry, the signal drops gradually as the ferry moves away from the departing shore, which impacts the available bandwidth. Thus, with all ferry scenarios with three, five and ten clients, we see the QoE value gradually drops for both MQoE models along with the bitrate (see Fig. 40-42) for *Big Buck Bunny*. We do note a small difference in window 10 for all the scenarios as the bitrate shows a small increase from window 9 to 10.

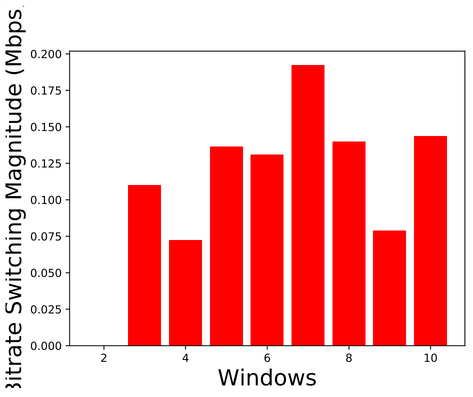
With three clients, there is a small difference in MQoE_RF compared to MQoE_SD on each $\Delta t$ (MQoE_RF is higher than MQoE_SD). With MQoE_RF when $\gamma$ decreases in set2, it does not reflect the small peaks of bitrate significantly. However, MQoE_SD is



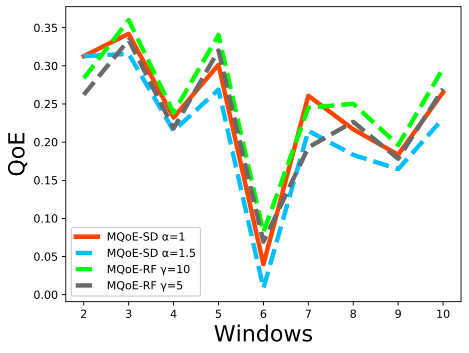
(a) Bitrate



(b) Exponential smoothing values of switching frequency

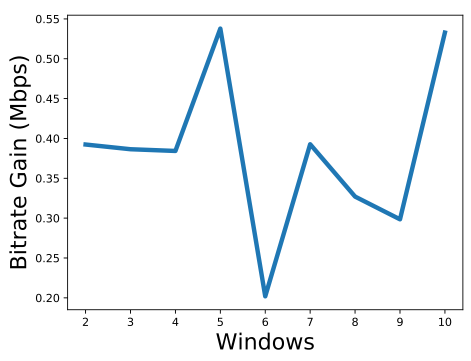


(c) Standard deviation of switching

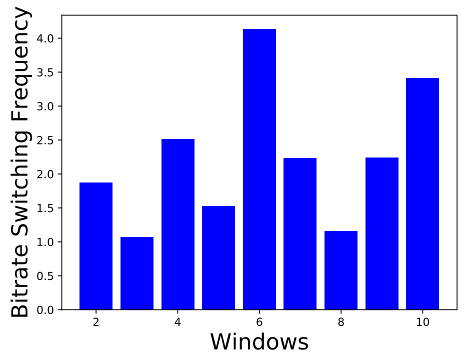


(d) MQoE

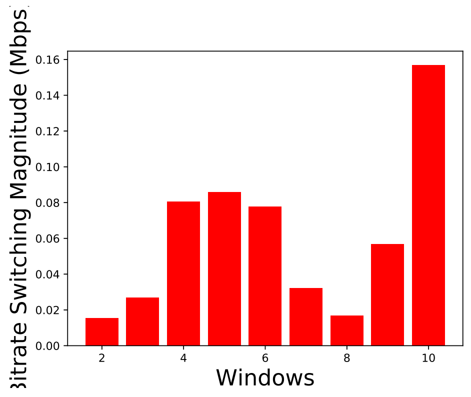
Figure 34: Three clients: train (ED)



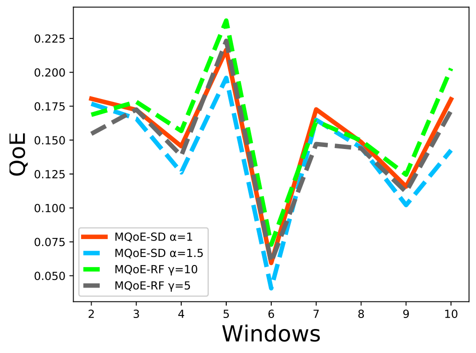
(a) Bitrate



(b) Exponential smoothing values of switching frequency

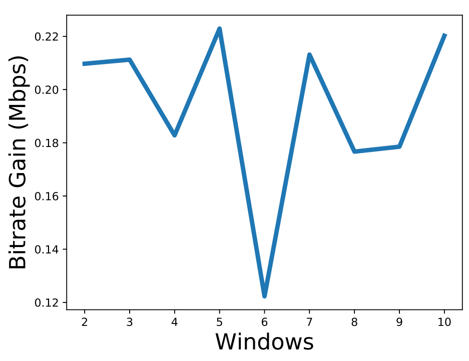


(c) Standard deviation of switching

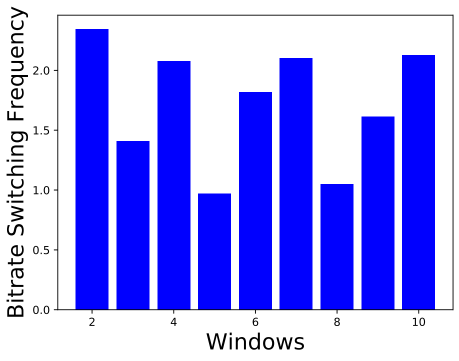


(d) MQoE

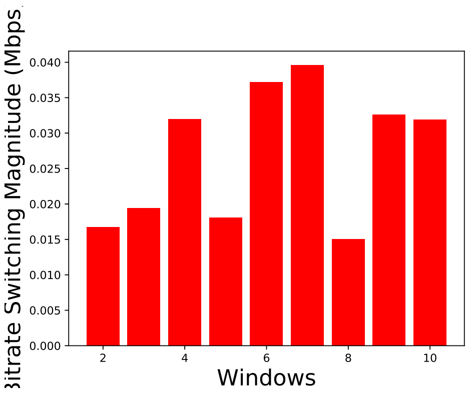
Figure 35: five clients: train (ED)



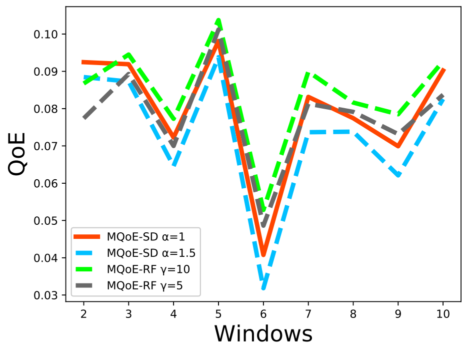
(a) Bitrate



(b) Exponential smoothing values of switching frequency



(c) Standard deviation of switching



(d) MQoE

Figure 36: Ten clients: train (ED)

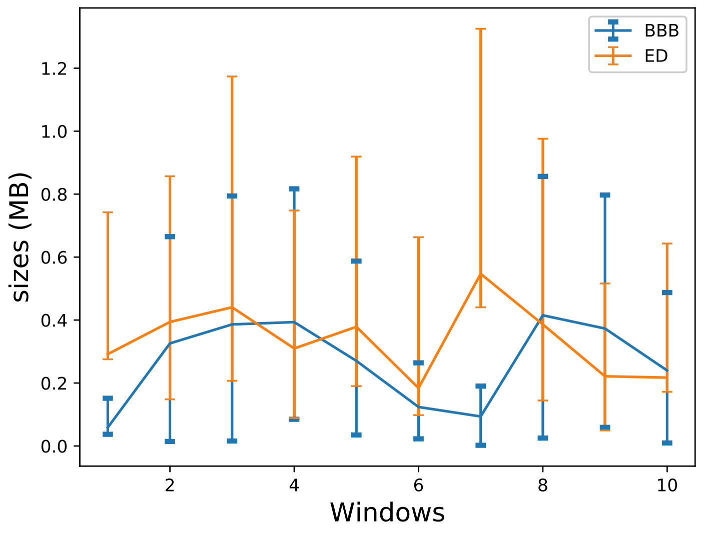


Figure 37: Three clients (train): window bytes BBB and ED

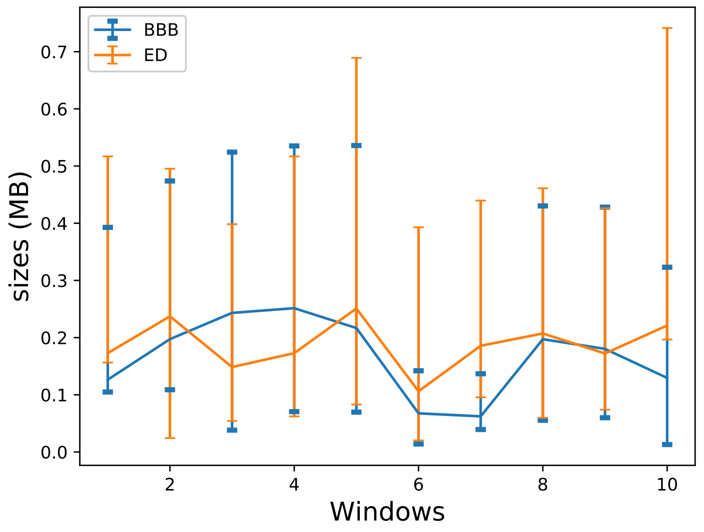


Figure 38: Five clients (train): window bytes BBB and ED

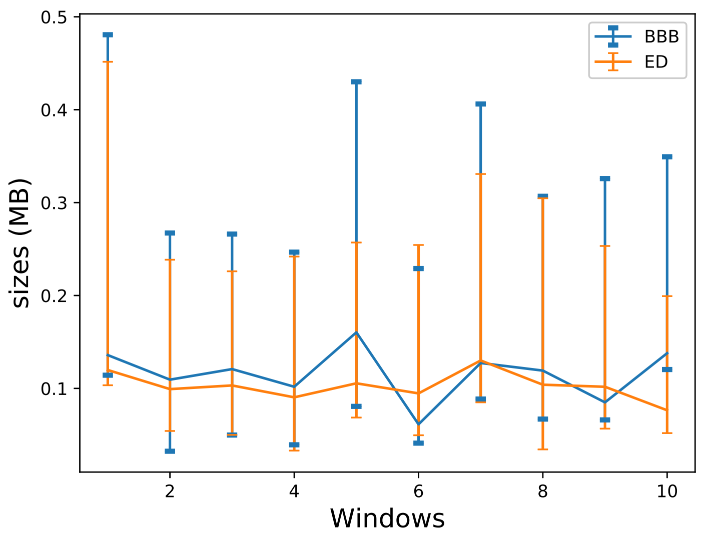
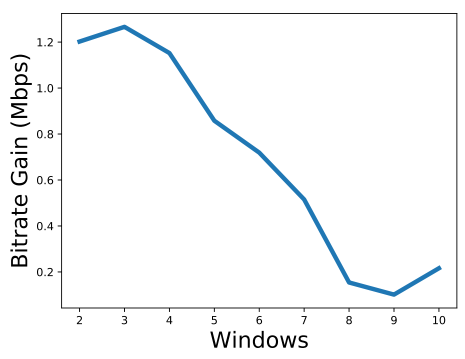


Figure 39: Ten clients (train): window bytes BBB and ED

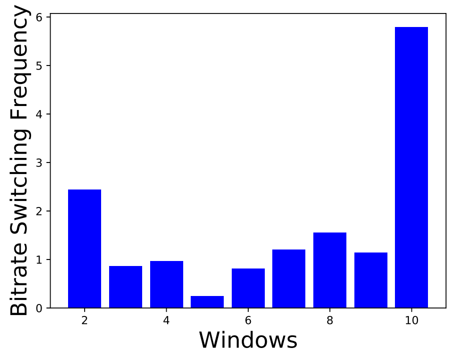
more sensitive on the magnitude of bitrate switching and shows bumps and dents even more than the bitrate.

For the five and ten clients also MQoE_RF is higher than MQoE_SD. For MQoE_RF, the jumps on window 4 and window 7 are significantly higher than MQoE_SD. This is mostly affected by increase of the bitrate on those windows when there are more number of clients to compete and the value of bitrate switching frequency decreases. There are high values of bitrate switching magnitude in these windows, which does not let MQoE_SD to rise as much as MQoE_RF.

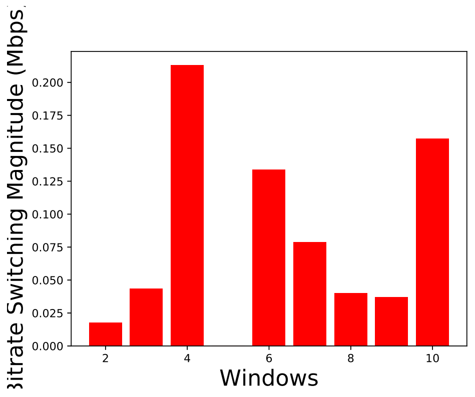
For ten clients, the peaks get smaller when the $\alpha$ increases as the bitrate switching magnitude is large for these windows; by increasing $\alpha$ (set2), its impact on the linear model MQoE_SD is noticeable. The trend in the case of *Elephants Dream* is very similar



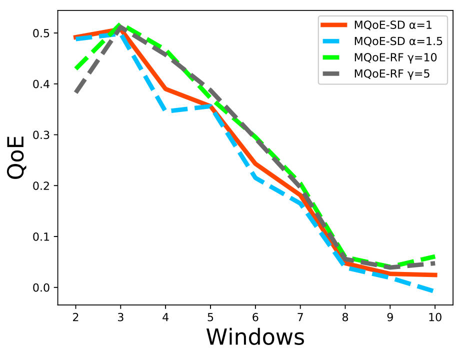
(a) Bitrate



(b) Exponential smoothing values of switching frequency



(c) Standard deviation of switching

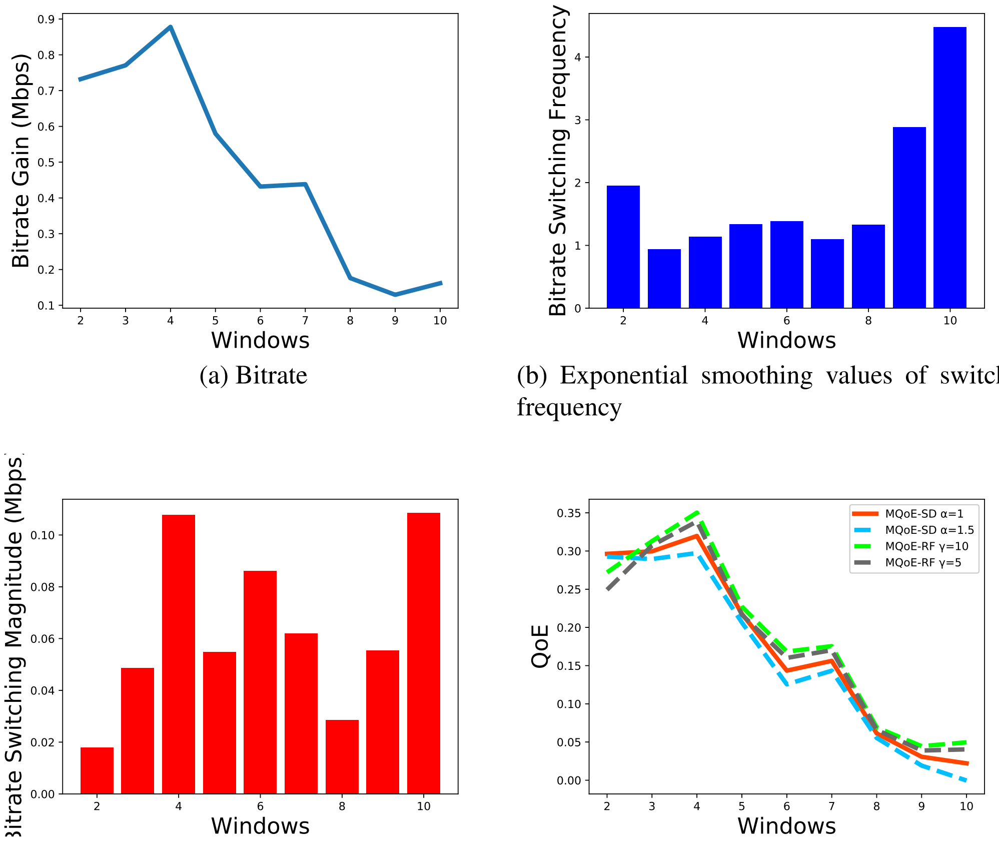


(d) MQoE

Figure 40: Three clients: ferry (BBB)

to *Big Buck Bunny* and, thus, is not shown here. Recall that our model does not explicitly consider rebuffering. We found that only in the case of a ferry, we observed some rebuffering, which occurred as it reached the lowest point of the wireless signal.

Certainly, rebuffering is a factor when there is a dead time. On the other hand, our QoE models capture this drop indirectly by reporting a lower QoE value and assigning a zero if there are no segments transmitted at all in a window. Thus, as we postulated early on, rebuffering does not need to be explicitly captures in the MQoE models.



(a) Bitrate

(b) Exponential smoothing values of switching frequency

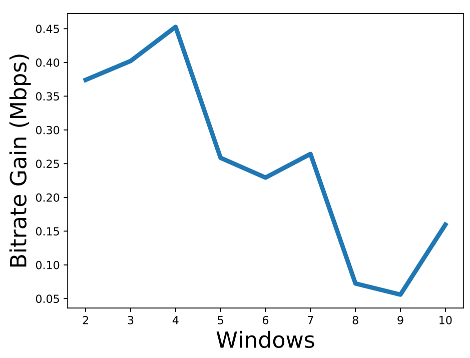
(c) Standard deviation of switching

(d) MQoE

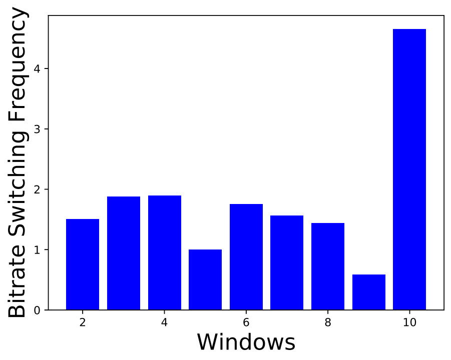
Figure 41: Five clients: ferry (BBB)

### 6.3.4 MQoE Comparison with Multi-Client and Fairness

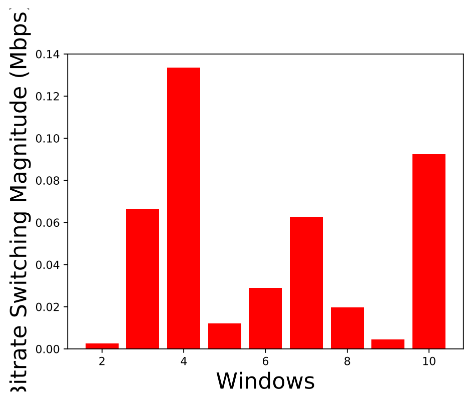
We now discuss how increasing the number of clients impacts MQoE. With a higher number of the clients, the competition for the resources increases, which causes each client to get a smaller share of the resources. Smaller shares also causes a QoE degradation (see Fig. 43, shown for *Big Buck Bunny*). In the case of cars, MQoE_RF with $gamma = 10$ (set1), three clients scenario has higher MQoE value than for five and ten clients. With five clients, the QoE decreases about 36.86% on average compared to three



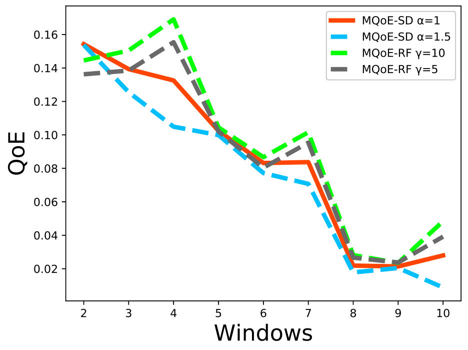
(a) Bitrate



(b) Exponential smoothing values of switching frequency



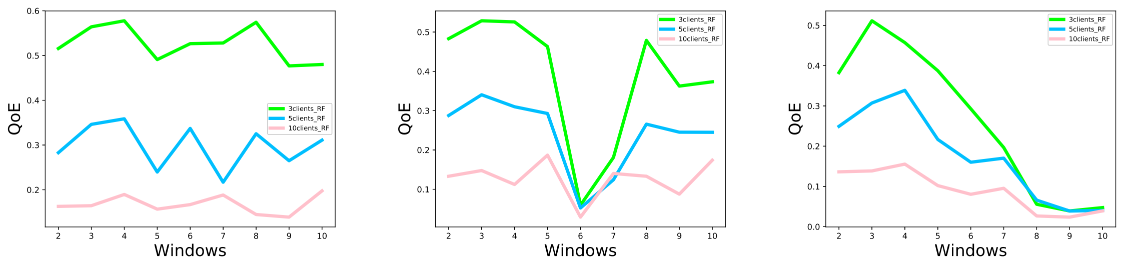
(c) Standard deviation of switching



(d) MQoE

Figure 42: Ten clients: ferry (BBB)

clients. With ten clients, the QoE shows a decrease of 53.71% on average compared to five clients. In the case of the train, regardless of the number of clients, the QoE drops to the lowest point at window 6. For the rest of the windows, QoE behavior with the train is showing almost the same pattern as with the car. On average, with five clients, the QoE decreases by 37% compared to three clients. With ten clients, the QoE shows a decrease of 47% compared to five clients. With the ferry, we see the decreasing trend from the starting window to the final window. But the QoE drop for the case with three clients is

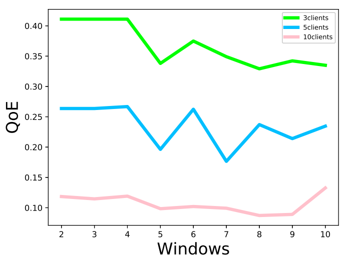


(a) car (b) train (c) ferry

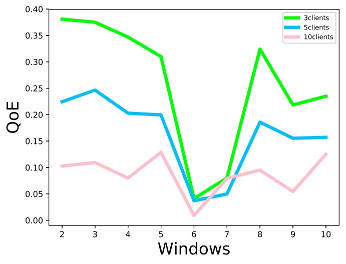
Figure 43: mult-client: MQoE_RS, $\gamma = 10$

much more significant since the QoE was higher to start at the beginning compared to five and ten clients scenarios.

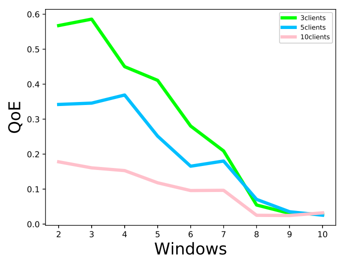
We show the results for three, five and ten client scenarios for MQoE_SD with $\alpha = 1$ in Fig. 44. We observed a similar trend as with MQoE_RF. We also analyze how fairly each client is treated in terms of MQoE. In the case of three clients, for MQoE_RF of each client with $\gamma = 10$, the standard deviation was 6.89% compared to the average among all the clients. With the same weight parameter, for ten clients this value decreased to 2.52%. The lower deviation indicates that clients are getting fairer share in a congested environment, although none are getting very high QoE. Recall that all clients watched the same video in our study. This shows that the ABR does not treat each client equally fairly except for in a congested environment.



(a) car



(b) train



(c) ferry

Figure 44: mult-client: MQoE_SD, $\alpha = 1$

# CHAPTER 7

# CONCLUSION

In this Dissertation, we provide solutions for QoE improvemnt with in-network cache prefetcing and QoE monitoring from ISP and content providers perspective.

In the first work, we propose a smart in-network cache prefetching scheme for video streaming to prefetch a segment bitrate. Smart cache is a multi-threaded limited-sized cache server that uses an ABR algorithm for segment bitrate selection in prefetching. This selection is based on forecasted throughput values calculated with exponential smoothing in the cache server itself. The forecasted throughput values are based on the previous moving average throughput values from the clients. This idea has been studied in two sets of experiments, once with a single client and bottleneck on the access link. Then, with double clients and bottleneck on the core link along with a cache replacement algorithm. We compared our scheme with an original basic scheme. The results show that smart cache increases hitrate in the cache, thereby improving throughput and QoE metrics of the clients such as average bitrate gain, latency, and bitrate switching frequency.

In the second work, we present two moving QoE models that can report ensemble QoE in review windows on a periodic basis for multiple clients streaming. To our knowledge, we are the first to propose Moving QoE (MQoE) models for video streaming performance monitoring that can be used by content service providers. Secondly, the multi-client scenario has rarely been studied before, which is important to consider for content providers. Our study shows that such models can be used to understand the QoE behavior of multiple clients during streaming, especially for a video transmission such as for a live event. We also found out that a static QoE model such as MQoE_MO is not suitable for moving QoE. Our nonlinear model MQoE_RF is preferable when a content provider wants to capture the bitrate switching frequency in the QoE model. Our linear model MQoE_SD, in general, is useful at capturing the standard deviation of bitrate switching. Based on our observation, For weight parameters, $\gamma = 10$ with MQoE_RF and $\alpha = 1$ with MQoE_SD (i.e., set1) were found to be the best values to use to adequately capture the bitrate switching frequency and bitrate switching magnitude impact while keeping quality due to bitrates. Our work is expected to be useful to content providers to observe variations for different conditions and fairness on QoE received by different clients so that they can take appropriate actions.

# REFERENCE LIST

[1] Dash.js reference client implementation. https://github.com/Dash-Industry-Forum/dash.js.

[2] Delivering live YouTube content via DASH. https://developers.google.com/youtube/v3/live/guides/encoding-with-dash.

[3] GENI testbed. https://portal.geni.net/.

[4] WonderShaper – A tool to limit network bandwidth in Linux. https://www.tecmint.com/wondershaper-limit-network-bandwidth-in-linux/.

[5] International Telecommunication Union: P. 10: Vocabulary for performance and quality of service, Amendment 2: New definitions for inclusion in recommendation ITU-T P. 10/G. 100. *Int. Telecomm. Union, Geneva* (2008).

[6] ISO/IEC 23009-1: 2019: Information technology-Dynamic adaptive streaming over HTTP (DASH)-Part 1: Media presentation description and segment formats, 2019. http://web.archive.org/web/20210226024535/https://www.iso.org/standard/75485.html.

[7] Alberti, C., Renzi, D., Timmerer, C., Mueller, C., Lederer, S., Battista, S., and Mattavelli, M. Automated QoE evaluation of dynamic adaptive streaming over HTTP. In *2013 Fifth International Workshop on Quality of Multimedia Experience (QoMEX)* (2013), IEEE, pp. 58–63.

[8] Balachandran, A., Sekar, V., Akella, A., Seshan, S., Stoica, I., and Zhang, H. Developing a predictive model of quality of experience for internet video. *ACM SIGCOMM Computer Communication Review 43*, 4 (2013), 339–350.

[9] Barman, N., and Martini, M. G. QoE modeling for HTTP adaptive video streaming–a survey and open challenges. *IEEE Access 7* (2019), 30831–30859.

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

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

[12] Brunnström, K., Beker, S. A., De Moor, K., Dooms, A., Egger, S., Garcia, M.-N., Hossfeld, T., Jumisko-Pyykkö, S., Keimel, C., Larabi, M.-C., et al. Qualinet white paper on definitions of quality of experience, 2013. http://web.archive.org/web/20181104090406/https://hal.archives-ouvertes.fr/hal-00977812/document.

[13] Cisco Visual Networking. Forecast and methodology, 2016–2021. *White Paper* (2017). http://web.archive.org/web/20200214054830/https://www.reinvention.be/webhdfs/v1/docs/complete-white-paper-c11-481360.pdf.

[14] Coverdale, P., Moller, S., Raake, A., and Takahashi, A. Multimedia quality assessment standards in ITU-T SG12. *IEEE Signal Processing Magazine 28*, 6 (2011), 91–97.

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

[16] De Vriendt, J., De Vleeschauwer, D., and Robinson, D. Model for estimating QoE of video delivered using HTTP adaptive streaming. In *2013 IFIP/IEEE International Symposium on Integrated Network Management (IM 2013)* (2013), IEEE, pp. 1288–1293.

[17] Dobrian, F., Sekar, V., Awan, A., Stoica, I., Joseph, D., Ganjam, A., Zhan, J., and Zhang, H. Understanding the impact of video quality on user engagement. *ACM SIGCOMM Computer Communication Review 41*, 4 (2011), 362–373.

[18] Egger, S., Gardlo, B., Seufert, M., and Schatz, R. The impact of adaptation strategies on perceived quality of HTTP adaptive streaming. In *Proceedings of the 2014 Workshop on Design, Quality and Deployment of Adaptive Video Streaming* (2014), ACM, pp. 31–36.

[19] Guo, Z., Wang, Y., and Zhu, X. Assessing the visual effect of non-periodic temporal variation of quantization stepsize in compressed video. In *2015 IEEE International Conference on Image Processing (ICIP)* (2015), IEEE, pp. 3121–3125.

[20] Hartshorn, C., Al-Shaikhli, W., Kiani Mehr, S., Cummins, D., Tamarapalli, V., and Medhi, D. Developing models from measurements in a noisy environment: lessons from an indoor Wi-Fi measurement study on application performance. In *The 30th International Conference on Computer Communications and Networks (IC-CCN 2021)* (2021), IEEE.

[21] Hoßfeld, T., Seufert, M., Sieber, C., and Zinner, T. Assessing effect sizes of influence factors towards a QoE model for HTTP adaptive streaming. In *2014 sixth international workshop on quality of multimedia experience (qomex)* (2014), IEEE, pp. 111–116.

[22] Juluri, P., Tamarapalli, V., and Medhi, D. Measurement of quality of experience of video-on-demand services: A survey. *IEEE Communications Surveys & Tutorials 18*, 1 (2015), 401–418.

[23] Khirman, S., and Henriksen, P. Relationship between quality-of-service and quality-of-experience for public internet service. In *Proc. of the 3rd Workshop on Passive and Active Measurement* (2002), vol. 1.

[24] Kiani Mehr, S., Jogalekar, P., and Medhi, D. Moving QoE for monitoring DASH video streaming: models and a study of multiple mobile clients. *Journal of Internet Services and Applications 12*, 1 (2021), 1–26.

[25] Kiani Mehr, S., Juluri, P., Maddumala, M., and Medhi, D. An adaptation aware hybrid client-cache approach for video delivery with dynamic adaptive streaming over HTTP. In *2018 IEEE/IFIP Network Operations and Management Symposium (NOMS)* (2018), IEEE, pp. 1–5.

[26] Kiani Mehr, S., and Medhi, D. QoE performance for DASH videos in a smart cache environment. In *2019 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)* (2019), IEEE, pp. 388–394.

[27] Kim, J.-Y., Cho, K.-W., and Koh, K. A proxy server structure and its cache consistency mechanism at the network bottleneck. In *Computer Software and Applications Conference, 1999. COMPSAC'99. Proceedings. The Twenty-Third Annual International* (1999), IEEE, pp. 278–283.

[28] Kreuzberger, C., Posch, D., and Hellwagner, H. A scalable video coding dataset and toolchain for dynamic adaptive streaming over HTTP. In *Proceedings of the 6th ACM Multimedia Systems Conference* (2015), pp. 213–218.

[29] Krishnamoorthi, V., Carlsson, N., Eager, D., Mahanti, A., and Shahrmehr, N. Helping hand or hidden hurdle: Proxy-assisted http-based adaptive streaming performance. In *Modeling, Analysis & Simulation of Computer and Telecommunication Systems (MASCOTS), 2013 IEEE 21st International Symposium on* (2013), IEEE, pp. 182–191.

[30] Krishnan, S. S., and Sitaraman, R. K. Video stream quality impacts viewer behavior: inferring causality using quasi-experimental designs. *IEEE/ACM Transactions on Networking 21*, 6 (2013), 2001–2014.

[31] Lederer, S., Müller, C., and Timmerer, C. Dynamic Adaptive Streaming over HTTP Dataset, 2012. http://www-itec.uni-klu.ac.at/ftp/datasets/DASHDataset2014/.

[32] Lee, D. H., Dovrolis, C., and Begen, A. C. Caching in HTTP adaptive streaming: Friend or foe? In *Proceedings of Network and Operating System Support on Digital Audio and Video Workshop* (2014), ACM, p. 31.

[33] Liang, K., Hao, J., Zimmermann, R., and Yau, D. K. Integrated prefetching and caching for adaptive video streaming over HTTP: an online approach. In *Proceedings of the 6th ACM Multimedia Systems Conference* (2015), pp. 142–152.

[34] Liu, C., Hannuksela, M. M., and Gabbouj, M. Client-driven joint cache management and rate adaptation for dynamic adaptive streaming over HTTP. *International Journal of Digital Multimedia Broadcasting 2013* (2013), 1–16.

[35] Lohmar, T., Einarsson, T., Fröjdh, P., Gabin, F., and Kampmann, M. Dynamic adaptive HTTP streaming of live content. In *2011 IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks* (2011), IEEE, pp. 1–8.

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

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

[38] Moldovan, C., Skorin-Kapov, L., Heegaard, P. E., and Hoßfeld, T. Optimal fairness and quality in video streaming with multiple users. In *2018 30th International Teletraffic Congress (ITC 30)* (2018), vol. 1, IEEE, pp. 73–78.

[39] Mueller, C., Lederer, S., and Timmerer, C. A proxy effect analysis and fair adaptation algorithm for multiple competing dynamic adaptive streaming over http clients.

In *Visual Communications and Image Processing (VCIP), 2012 IEEE* (2012), IEEE, pp. 1–6.

[40] Pang, Z., Sun, L., Wang, Z., Xie, Y., and Yang, S. Understanding performance of edge prefetching. In *International Conference on Multimedia Modeling* (2017), Springer, pp. 527–539.

[41] Pham, T.-D., Vo, P. L., and Thang, T. C. Improving DASH performance in a network with caching. In *Proceedings of the Eighth International Symposium on Information and Communication Technology* (2017), ACM, pp. 255–261.

[42] Poliakov, V., Sassatelli, L., and Saucez, D. Case for caching and Model Predictive Control quality decision algorithm for HTTP Adaptive Streaming: is cache-awareness actually needed? In *Globecom Workshops (GC Wkshps), 2016 IEEE* (2016), IEEE, pp. 1–6.

[43] Poliakov, V., Sassatelli, L., and Saucez, D. Impact of caching on HTTP adaptive streaming decisions: towards an optimal. In *Computer Communications Workshops (INFOCOM WKSHPS), 2016 IEEE Conference on* (2016), IEEE, pp. 1053–1054.

[44] Rejaie, R., Yu, H., Handley, M., and Estrin, D. Multimedia proxy caching mechanism for quality adaptive streaming applications in the Internet. In *INFOCOM 2000. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE* (2000), vol. 2, IEEE, pp. 980–989.

[45] Riiser, H., Vigmostad, P., Griwodz, C., and Halvorsen, P. Commute path bandwidth traces from 3G networks: analysis and applications. In *Proceedings of the 4th ACM Multimedia Systems Conference* (2013), pp. 114–118.

[46] Seufert, M., Egger, S., Slanina, M., Zinner, T., Hoßfeld, T., and Tran-Gia, P. A survey on quality of experience of HTTP adaptive streaming. *IEEE Communications Surveys & Tutorials 17*, 1 (2014), 469–492.

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

[48] Shao, B., Renzi, D., Amon, P., Xilouris, G., Zotos, N., Battista, S., Kourtis, A., and Mattavelli, M. An adaptive system for real-time scalable video streaming with end-to-end QoS control. In *11th International Workshop on Image Analysis for Multimedia Interactive Services WIAMIS 10* (2010), IEEE, pp. 1–4.

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

[50] Tian, G., and Liu, Y. Towards agile and smooth video adaptation in dynamic HTTP streaming. In *Proceedings of the 8th international conference on Emerging networking experiments and technologies* (2012), ACM, pp. 109–120.

[51] Timmerer, C., Weinberger, D., Smole, M., Grandl, R., Müller, C., and Lederer, S. Live transcoding and streaming-as-a-service with MPEG-DASH. In *2015 IEEE International Conference on Multimedia & Expo Workshops (ICMEW)* (2015), pp. 1–4.

[52] Tran, H. T., Vu, T., Ngoc, N. P., and Thang, T. C. A novel quality model for HTTP Adaptive Streaming. In *2016 IEEE Sixth International Conference on Communications and Electronics (ICCE)* (2016), IEEE, pp. 423–428.

[53] Wang, C., Bhat, D., Rizk, A., and Zink, M. Design and analysis of QoE-aware quality adaptation for DASH: A spectrum-based approach. *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM) 13*, 3s (2017), 1–24.

[54] Wei, S., and Swaminathan, V. Low latency live video streaming over HTTP 2.0. In *Proceedings of Network and Operating System Support on Digital Audio and Video Workshop* (2014), pp. 37–42.

[55] Xue, J., Zhang, D.-Q., Yu, H., and Chen, C. W. Assessing quality of experience for adaptive HTTP video streaming. In *2014 IEEE International Conference on Multimedia and Expo Workshops (ICMEW)* (2014), IEEE, pp. 1–6.

[56] Yan, F. Y., Ayers, H., Zhu, C., Fouladi, S., Hong, J., Zhang, K., Levis, P., and Winstein, K. Learning in situ: a randomized experiment in video streaming. In *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI) 20* (2020), pp. 495–511.

[57] Yarnagula, H. K., Juluri, P., Kiani Mehr, S., Tamarapalli, V., and Medhi, D. QoE for mobile clients with segment-aware rate adaptation algorithm (SARA) for DASH video streaming. *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM) 15*, 2 (2019), 1–23.

[58] Yin, X., Jindal, A., Sekar, V., and Sinopoli, B. A control-theoretic approach for dynamic adaptive video streaming over HTTP. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication* (2015), pp. 325–338.

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

# VITA

Sheyda Kiani Mehr received her BS.c in Computer Engineering-Software from Shomal University of Amol, Iran in 2011. She started her graduate program journey at University of Missouri-Kansas City in 2014 in i-PhD program, Telecommunications and Computer Networking as major and Computer Science as minor. She got her MS.c in Computer Science in 2017 from University of Missouri-Kansas City.

Ms. Kiani Mehr worked as an Intern at Openwave Mobility, Redwood City, CA in 2018. Currently, Ms. Kiani Mehr is an intern in Ericsson, Santa Clara, CA since 2019.