MIMIC: Using Passive Network Measurements to Estimate HTTP-based Adaptive Video QoE Metrics

Tarun Mangla*, Emir Halepovic†, Mostafa Ammar*, Ellen Zegura*
*Georgia Institute of Technology †AT&T Labs Research
{tmangla3, ammar, ewz}@cc.gatech.edu emir@research.att.com

***Abstract*—HTTP-based Adaptive Streaming (HAS) has seen a major growth in the cellular networks. As a key application and network demand driver, user-perceived Quality of Experience (QoE) of video streaming contributes to the overall user satisfaction. Therefore, it becomes critical for the cellular network operators to understand the QoE of video streams. It can help with long-term network planning and provisioning and QoE-aware traffic management. However, tracking QoE is challenging as network operators do not have direct access to the video streaming apps, user devices or servers. In this paper, we provide a methodology that uses passive network measurements of unencrypted HAS video streams to estimate three key video QoE metrics - average bitrate, re-buffering ratio and bitrate switches. Our approach relies on the semantics of HAS to model a video session on the client. We first develop and validate our methodology through controlled experiments in the lab. Then, we conduct a large-scale validation of our approach using network data from a major cellular operator and ground truth QoE metrics from a large video service. We accurately predict the value of average bitrate within a relative error of 10% for 70%-90% of video sessions and re-buffering ratio within 1 percentage point for 65 -90% of sessions. We further quantify the network overhead due to video chunk replacement and observe that a significant number of sessions have a high overhead of 20% or more. Finally, we highlight several challenges with video QoE metrics estimation in a large-scale monitoring system.**

I. Introduction

The amount of mobile video traffic has grown tremendously and is further expected to increase 9-fold within the next 5 years [1]. Mobile network operators (MNOs) are increasingly expected to provide a high-quality video streaming experience, as video services assume mobile access. This growing scale and customer expectations require all operators, including MNOs to adequately provision their networks and support video delivery. Clearly, this network planning and provisioning can be better-informed if MNOs have an in-depth understanding of end user QoE and its relationship to network performance. Specifically, it can help network operators assess the impact of changing network configurations on video QoE and also allow long-term QoE-aware network provisioning.

However, estimating video QoE is challenging for a network operator, since they typically do not have access to the streaming app on user device, the device itself or the server. Hence, unlike content providers, they cannot use in-app plug-ins [2], [3] for measuring the streaming QoE. Recent work [4] proposes an alternative networking paradigm in which network operators and content providers collaborate, allowing content providers to share QoE information with network operators. However, this approach requires significant effort and is not immediately realizable. MNOs are, therefore, constrained to use data derived from within their network to estimate application and service quality, including video QoE.

Most of the existing QoE estimation approaches attempt to correlate QoS metrics derived from network measurements with video QoE metrics. OneClick [5] and HostView [6] develop regression models to detect the QoE of multiple applications including video streaming. Prometheus uses machine learning to estimate video re-buffering from flow-level measurements [7]. Similarly, Dimopoulos et al. [8] inspect QoE metrics sent by the player to the content provider to correlate them with network-level metrics. A general drawback with these statistical approaches is that they require ground truth QoE metrics for initial training of models. Such information is not easily available to the network operators. This strong dependence on ground truth QoE metrics for building the correlation makes it challenging for any network operator to implement such approaches in practice.

In addition, major content providers have switched to HTTP-based Adaptive Streaming (HAS) for delivering video. In HAS [9], the video player at the client dynamically changes the bitrate quality of requested video to adapt to the network conditions, also known as Adaptive BitRate (ABR) streaming. This application-layer bitrate adaptation, makes it more challenging to translate network QoS metrics to application layer video QoE metrics. Furthermore, different native players and even some content providers have proprietary implementations of the bitrate adaptation algorithms. Thus, models learnt for one video service might not yield the same accuracy if used for other services as observed in [8].

In this paper, we propose a methodology called MIMIC[1], that uses semantics of HAS to estimate video QoE metrics from passive network measurements for unencrypted HAS videos. MIMIC is based on the observation that the HTTP logs of HAS videos can give significant information to model a video session on the client. We use HTTP logs to estimate three video quality metrics: *average bitrate, re-buffering ratio*, and *bitrate switches*. In addition, we also monitor network overhead due to chunk replacement which is an important

[1]MIMIC stands for **M**easur**i**ng **M**ultimedia QoE **i**n **C**ellular network. The acronym is suggestive of our approach, *i.e.* we try to mimic the client video session playback using network traces

metric for network operators. MIMIC gives a fine-granular view of video QoE metrics *per session* by estimating their exact values as opposed to the categorical estimates made by earlier methods. An MNO can either use each of these metrics individually or feed the estimated values into an appropriate QoE prediction model, such as [10], to get a single score reflecting the QoE for the video session. MIMIC is similar in spirit to [11] and [12] that use TCP-layer metrics and HTTP logs respectively to model a video session at client. However, these methods were designed for HTTP progressive streaming and would not work for HAS video. MIMIC is developed in a controlled environment and then validated on a large-scale using network data from a major cellular provider and ground truth QoE metrics from a major content provider.

Our contributions can be summarized as follows:

i) We develop MIMIC*, a methodology for estimating video QoE metrics from passive measurements of HTTP logs;*

ii) MIMIC *can accurately predict the average bitrate within a relative error of 10% for 70%-90% of video sessions;*

iii) MIMIC *is able to predict re-buffering ratio within an absolute error of 1% for 65%-90% of video sessions, and*

iv) We quantify the network overhead due to video chunk replacement and observe that a significant number of sessions can incur a high overhead of 20% or more.

Finally, we highlight some limitations of our approach and challenges encountered in a large-scale QoE monitoring system. The rest of the paper is organized a follows: Section II provides a brief background of HAS technology and the associated QoE metrics. Section III describes methodology to estimate video QoE metrics in detail. In section IV, we discuss the results from controlled experiments as well as cellular network data, followed by a discussion of challenges and future work in Section V. We conclude the paper in Section VI.

II. BACKGROUND

In HAS, the video is split into chunks of equal duration with each chunk encoded at multiple bitrates chosen from a set of pre-defined bitrates. The video player on the client decides the quality of chunks to stream based on some adaptation logic. The use of HTTP makes this approach middle-box friendly and enables content providers to use commodity Content Distribution Network (CDN) servers. The bitrate adaptation at the client allows a diverse set of clients to perform well under a variety of network conditions. As a result, an overwhelming majority of content providers now rely on HAS-based technologies such as HLS [13] and MPEG-DASH [14] to stream videos to users.

The network traffic in an HAS video session consist of a sequence of HTTP GET requests and responses. When the client opens the video, the player first downloads a *manifest* file by sending an HTTP GET request to the server. This *manifest* file has the information about media segments such as bitrate levels, resolution and request Uniform Resource Identifier (URI). The player then sends an HTTP GET request for the first chunk. The quality of the requested chunk is usually specified in the chunk URI. Once the video chunk has been fully downloaded, it is decoded and played on the screen. Meanwhile, the player sends the request for next chunk, whose quality is decided based on the past chunk throughput and current buffer occupancy, and this process continues. Note that players typically do not pipeline chunk requests, i.e., request for the next chunk is sent only after the current chunk has been downloaded. *Our QoE metrics estimation approach exploits this strong serial request-response pattern for modeling a video session using HTTP logs.*

QoE metrics: In HAS, QoE is usually characterized by multiple quality metrics such as *average bitrate, re-buffering ratio*, and *bitrate switches* [10]. Average bitrate reflects the streamed video quality, based on the requirement for more bits to encode a higher quality image. Re-buffering ratio captures the extent to which video stalled because of buffer underrun. Bitrate switches represents the bitrate and user-perceived image quality variations in the session.

In addition, the content providers also care about video startup time (VST) which is the time it takes for the video to begin playing since the user requested the video. Our methodology currently does not estimate the VST and we consider it as a part of our future work.

III. METHODOLOGY

Conceptually, we utilize HTTP logs collected from the network to model a video session on the client. We illustrate MIMIC with a mobile app, referred to as *VideoApp* (anonymized for confidentiality), from a large mobile video service. We first describe measurements collected passively and then discuss how these measurements can be used to estimate different video QoE metrics. We then briefly describe the benchmark data that we use to evaluate the accuracy of our approach.

A. Passive Measurements

Our passive measurements include HTTP logs for *VideoApp* recorded by deploying a web proxy in the network. We identify the logs as belonging to the *VideoApp* by inspecting the request URIs. In addition to the request URI, the web proxy also provides the HTTP response completion timestamp and content size. Note that deep packet inspection techniques could also be used for this purpose, but at higher processing cost. The log data used for this study is anonymized and does not contain any user-identifiable information. We only consider logs that correspond to video chunk requests.

Figure 1 shows a sample chunk requst URI from the *VideoApp* session. We use the session identifier field in the request URI to group the request logs into video sessions. From the URIs of these chunk requests, we then extract the chunk identifier and bitrate or quality. We also collect some meta-data about each session such as device OS, OS

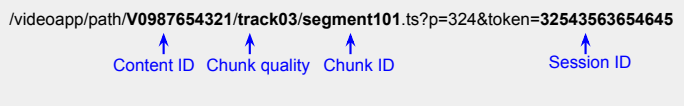


Fig. 1: Chunk URI template and the extracted information

version, content type and content identifier from the request URI and headers. Note that chunk bitrate (quality) and content identifiers in the URI are typically represented by arbitrary or randomized service-specific unique identifiers rather than human-readable values.

Chunk replacement: It is common to observe multiple chunk requests with the same chunk identifier, but different bitrate in the logs. This essentially means that the video client replaced an already requested chunk. This behavior has been alluded to in a previous study [15] but has not been quantified at large scale. Chunk replacement can happen due to several reasons, including (i) player trying to improve user experience when network conditions allow, and (ii) recovering from various errors or overly aggressive but aborted attempts for high-quality chunks. We handle chunk replacement by using only the most recently downloaded bitrates of each chunk to estimate QoE metrics, while carefully accounting for all replaced chunks. The replaced chunks are important from the perspective of both MNO (represent wasted network resources) and end user (represent wastage of limited data plan). We call the total amount of data due to replaced chunks as *chunk replacement (CR) overhead*.

B. Estimating QoE metrics

From the passive measurements, we get request completion time (T_i), chunk quality (Q_i) and chunk size (S_i) for every chunk request i in the video session V. The chunk duration (L) in seconds is obtained by investigating the *manifest* file for few videos in out-of-band experiments. We observed that *VideoApp* uses different chunk duration for Live and Video on Demand (VoD) content. The total number of chunks downloaded in a session after accounting for chunk replacement are denoted by N. We use this information to estimate different video quality metrics in the following manner:

- *Average bitrate*: We estimate the average bitrate by taking the time average of the collected chunk size of the session.

$$\hat{br} = \frac{\sum_{i=1}^{N} S_i}{N \times L} \tag{1}$$

- *Re-buffering ratio*: Intuitively, re-buffering time is estimated by keeping an account of video chunks that have been downloaded and the part of that video content that should have been played so far. Let B_i denote the total re-buffering since the beginning of the play until the time chunk i has been downloaded i.e. T_i. Clearly, B_1 is zero by definition as the initial re-buffering is termed as the video startup time. The re-buffering time between two consecutive chunk download times T_i and T_{i-1} is denoted by b_i. Then, B_i can be simply written as $\sum_{k=1}^{i} b_k$ and each of the b_i can be calculated as follows:

$$b_i = \max(T_i - T_1 - B_{i-1} - (i-1) \times L, 0) \quad \forall i \geq 2 \tag{2}$$

  Here, $T_i - T_1 - B_{i-1}$ represents the total video that should have been played since the beginning of the session and $(i-1) \times L$ represents the video content that has been downloaded. The re-buffering ratio can then simply be written as

$$\hat{rr} = \frac{B_N}{N \times L + B_N} \tag{3}$$

- *Number of bitrate switches*: Number of bitrate switches can be calculated simply by calculating the number of times the chunk quality changed between consecutive chunks. Here I is the indicator function and has a value of 1, if the consecutive chunks do not have same quality, zero otherwise.

$$\hat{br\_switch} = \sum_{i=2}^{N} I(Q_i \neq Q_{i-1}) \tag{4}$$

C. Ground Truth

To validate the accuracy of our approach we require the actual video QoE metrics. For this purpose, we use session-level QoE metrics collected by a mainstream 3rd party video analytics SDK built into the *VideoApp*. As noted before, this is the typical approach to QoE monitoring of commercial video services. It is our understanding that SDKs use API calls to the native player to register events such as playing, stopped, buffering at the application layer. The detailed logic is proprietary to each vendor. The QoE metrics provided by in-app SDK consist of average bitrate, re-buffering ratio, and video startup time. We assume that metrics from the SDK accurately capture user-perceived QoE. The in-app SDK does not report bitrate switches and chunk replacement. The ground truth logs are anonymized for privacy by the SDK, so that users or end devices could not be identified.

IV. Evaluation

The evaluation involves measuring the QoE metrics of *VideoApp* sessions using the network data and comparing them to the QoE metrics reported by the in-app SDK, referred to as ground truth. We first conduct experiments in a controlled environment, followed by validation using the real network data. In our analysis, we split the *VideoApp* sessions by content type *i.e.* Live or VoD, and Operating System (OS). This is because the video system parameters and the exact HAS implementation may depend on the content type and OS. We consider the two popular mobile OS's and refer to them as OS1 and OS2.

A. Evaluation using controlled experiments

Experimental Setup: The experimental setup consists of a smartphone with the *VideoApp* installed and a Linux box acting as a WiFi hotspot. Squid proxy is deployed on the Linux box to log all HTTP traffic from the smartphone. We use Linux tc to control the downlink bandwidth to the smartphone. A single test run constitutes of streaming a specific video in *VideoApp* for a duration of 5 minutes and under a specific bandwidth profile. The collected proxy logs from these test runs correspond to the logs we can get from the real network and we use them to calculate the video QoE metrics and then compare them to the ground truth. Creating a special test user allows us to positively match each test run with the *VideoApp* ground truth logs.

TABLE I: Bandwidth profiles used for controlled experiments.

BW1	constant bandwidth of 10000 kbps
BW2	2000 kbps with 20 kbps from t=180 s to t=240 s
BW3	bandwidth alternating between 2000 and 20 kbps every 30 s
BW4	bandwidth changes every 10 seconds, range=[20,10000] kbps, mean=2951 kbps, stddev=3932 kbps

TABLE II: QoE metrics estimation results from controlled experiments: VoD content, OS1

Bandwidth profile	Average bitrate (kbps)			Re-buffering ratio (%)	
	M.V.	P.V.	G.T.	M.V.	G.T.
BW1	3213	3680	3690	0.61%	0.09%
BW2	1220	1442	1440	3.48%	3.41%
BW3	855	1003	1030	32.08%	32.7%
BW4	1983	2276	2330	11.41%	12.0%

We run experiments with one representative video from Live and VoD content served by *VideoApp*. Each video was continuously streamed under four different bandwidth profiles described in Table I. The goal of these experiments is to do an initial rather than exhaustive validation of our methodology.

Results: Table II shows the comparison between the measured value (M.V.) and ground truth (G.T.) value of QoE metrics under different bandwidth profiles for VoD on OS1. Our methodology appears to consistently underestimate the average bitrate. Upon a closer examination, we find that the difference in our measurements can be attributed to the difference between the way we compute average bitrate and the way the in-app SDK calculates it. The in-app SDK uses the declared bitrate in the manifest whereas we consider the actual size of the chunks on the network while estimating the bitrate value. The declared bitrate represents the peak chunk bitrate over the entire video, as required by HLS design, thus leading to higher estimate of average bitrate by the in-app SDK. To verify this, we also calculate the predicted value (P.V.) using the declared bitrates in the *manifest* file. As shown in Table II the average bitrate predicted using this method is very close to the ground truth.

Our methodology can predict re-buffering ratio quite accurately with an absolute error less than 1% for VoD. The results are similar for experiments with Live video as shown in Table III. In addition, we also calculate the number of bitrate switches and CR overhead, not shown here as we do not have the ground truth for these. The controlled experiments show that we can very accurately predict the video QoE metrics using the network data.

B. Evaluation using large-scale real network data

Dataset: We deployed a web proxy in the cellular network of a major operator in the U.S. to obtain HTTP logs for *VideoApp* sessions. The logs were collected for a period of 12 days in the year 2017, in a part of the network covering a fraction of packet gateways. We estimate the QoE metrics for the logged sessions using MIMIC. Note that we are not able to store the raw HTTP logs due to large space overhead but only process the logs in a streaming fashion and retain the session-level QoE metrics and associated meta-data.

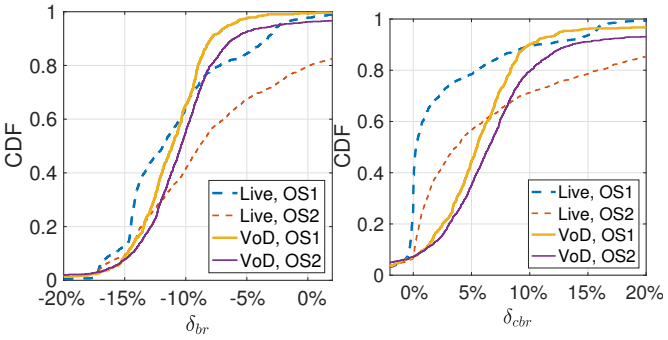
TABLE III: QoE metrics estimation results from controlled experiments: Live content, OS1

Bandwidth profile	Average bitrate (kbps)			Re-buffering ratio (%)	
	M.V	P.V	G.T.	M.V.	G.T.
BW1	3012	3242	3270	0.00%	0.14%
BW2	1146	1344	1260	8.87%	10.3%
BW3	769	917	951	12.69%	14.3%
BW4	994	1092	1190	11.70%	13.31%

Ground Truth: We again use the QoE metrics collected by the in-app SDK. However, unlike the active experiments, it is not trivial to match the sessions collected on the network with their corresponding in-app SDK logs. Due to the anonymization of both data sets, we have to resort to the following matching logic. We use the session content identifier, session start time and duration, OS kind and version, and CDN to match sessions. We further filter out sessions less than 1 minute since it is challenging to match the network session duration with the ground truth playtime for such short sessions. In the case of a single match between data sets, we use it for comparison, while in the case that a single *VideoApp* session matched with more than one proxy sessions, we discard all of them.

For the time period under study, we were able to match 70,214 sessions, which is a small fraction of all *VideoApp* sessions. Both Live and VoD are well-represented in this set. Although the efficiency of our highly simplistic matching logic is low, we still get a large number of sessions to validate our methodology.

Average bitrate: For *average bitrate*, we calculate the signed relative error (δ_{br}) in the estimated average bitrate ($\hat{br}$) and the ground truth average bitrate (br), defined as $\frac{\hat{br}-br}{br}$. Figure 2a shows the CDF of the relative error for sessions split by OS kind and content type. As expected, we underestimate the the average bitrate when compared to the in-app SDK reported bitrate since we are using the chunk size to estimate the bitrate as opposed to the bitrate values from the manifest by the in-app SDK. Note that from the point of view of MNO, the bitrate calculated using chunk size gives a more accurate indication of network load. We also estimate average bitrate using the declared values from the manifest in the controlled experiments. Figure 2b shows the CDF of the relative error (δ_{cbr}). We can predict the average bitrate within a relative error of 10% for at least 70% (90% for OS1) of sessions.



(a) Using chunk size (b) Using declared bitrates

Fig. 2: CDF of relative error in average bitrate estimation

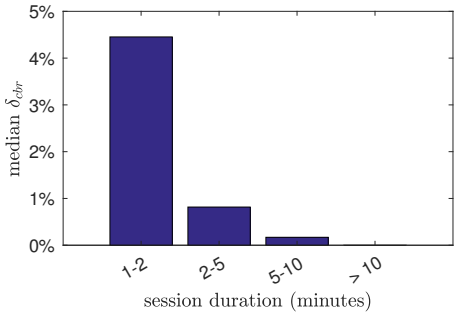


Fig. 3: Median relative error in average bitrate estimation vs. session duration

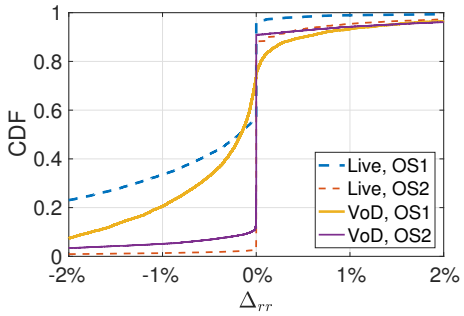


Fig. 4: CDF of error in re-buffering ratio estimation

We also note some challenges in bitrate estimation. We consistently overestimate average bitrate, although by a small value, when using declared values from the manifest. One reason for this overestimation is that we use all of the downloaded chunks during the session for average bitrate calculation unlike the in-app SDK which only considers chunks that have been played. From experiments, we observed that *VideoApp* sessions typically start with a low chunk quality and the quality increases later on as the playback progresses. Hence, if chunks in the playback buffer are also considered, it leads to overestimation of average bitrate. To validate this hypothesis, we classify the sessions into bins according to their duration and compute the median of the relative error in bitrate estimation for each bin. As shown in figure 3, the median relative error decreases as the session duration increase. This is because the relative contribution of the fixed-size buffer in the average bitrate decreases as the session playtime increases. Tracking the buffer occupancy in bitrate calculation can help make our estimation more accurate.

We also observed that the errors in bitrate estimation are higher in general for sessions on OS2 as compared to OS1. We speculate that it could either be because of a different chunk replacement policy in OS2 or a different methodology used by in-app for calculating bitrate. We plan to investigate this in detail in our future work.

Re-buffering ratio: We calculate the signed difference (Δ_{rr}) between the estimated re-buffering ratio ($\hat{rr}$) and the ground-truth re-buffering ratio (rr), defined as $\hat{rr} - rr$. Figure 4 shows the CDF of Δ_{rr} split by OS and content type. We can accurately predict the re-buffering ratio within an absolute error of 1% for 90% of session on OS2 and for 65% of sessions on OS1. Our methodology appears to underestimate the re-buffering ratio.

To understand this further, we categorize the re-buffering ratio into low ($rr < 1\%$), medium ($1\% < rr < 10\%$) and high ($rr > 10\%$) and compare the categorical predictions with ground truth. Table IV and V show the confusion matrices of these categorical predictions for VoD sessions on OS1 and OS2 respectively. For both OS types, our methodology can predict low and high re-buffering sessions with reasonably high accuracy. However, many sessions with medium re-buffering are classified low re-buffering. From a network operator's perspective, it is more important to identify sessions with higher re-buffering as compared to medium re-buffering.

One possible reason for re-buffering underestimation could be because of "trick play". Fast-forwarding or rewinding content in the video usually leads to resetting of buffer occupancy. Our estimation methodology does not detect and take into account trick play and would end up underestimating the re-buffering ratio in these cases. We also observed higher number of sessions with medium re-buffering on OS1 as compared to OS2. We speculate that this could be because of difference in HAS implementation or in-app SDK's reporting methodology. Identifying the root cause of this behavior is a part of our future work.

C. Additional metrics estimated from network data

Here we show the distribution of two additional metrics for which we do not have the ground truth. However, they demonstrate that we can obtain additional insights not otherwise available from in-app measurements.

Chunk replacement (CR) overhead: CR overhead is defined as the % of data in a video session transmitted due to replaced chunks. Table VI shows the CR overhead for *VideoApp* sessions split by OS and content type. A large number of sessions, as high as 90%, have non-zero chunk replacement. This can be attributed to the fact that most of the *VideoApp* sessions always start by downloading multiple lower quality chunks to quickly fill the playback buffer. The player replaces these chunks with higher quality chunks if it infers there is enough network bandwidth. This behavior is due to application or underlying OS native player design. CR accounts for 2.8% - 6.2% of the total network data for the *VideoApp* service. Furthermore, up to 35% of (Live, OS2) sessions have a CR overhead greater than 20%, which is a non-trivial overhead.

TABLE IV: Re-buffering ratio confusion matrix, VoD OS1

Ground Truth	Predicted re-buffering ratio		
	low rr	medium rr	high rr
low rr	90.6%	7.7%	1.6%
medium rr	49.9%	47.0%	3.1%
high rr	1.7%	15.5%	82.8%

TABLE V: Re-buffering ratio confusion matrix, VoD OS2

Ground Truth	Predicted re-buffering ratio		
	low rr	medium rr	high rr
low rr	94.4%	4.6%	1.0%
medium rr	56.5%	32.2%	11.3%
high rr	10.7%	9.7%	79.6%

TABLE VI: Network overhead due to chunk replacement

Content type	OS type	% sessions w/ non-zero CR	mean CR overhead (% bytes)	% sessions w/ CR overhead $\geq$ 20%	CR overhead (% total data)
Live	OS1	52.8%	7.5%	5.2%	2.8%
Live	OS2	89.2%	18.3%	35.8%	6.2%
VoD	OS1	91.9%	7.0%	10.9%	5.2%
VoD	OS2	92.6%	6.5%	9.4%	2.8%

This points to the need of looking into optimizing the trade-off between improved user experience and network overhead. However, the adaptation logic and CR behavior can be hidden within the native player design and not accessible to video content providers.

Bitrate switching: We compute bitrate switches per minute for each session and plot the CDF of its distribution in Figure 5. A large number of sessions have non-zero bitrate switches. This can be attributed to the player behavior during the startup phase of the video and later adaptation. The *VideoApp* player typically starts with a low bitrate level and switches up the bitrate as the playback progresses. We also observe high number of bitrate switches for Live sessions on OS2 and VoD sessions on OS1. We speculate this could be because of the differences in the player adaptation logic on these platforms.

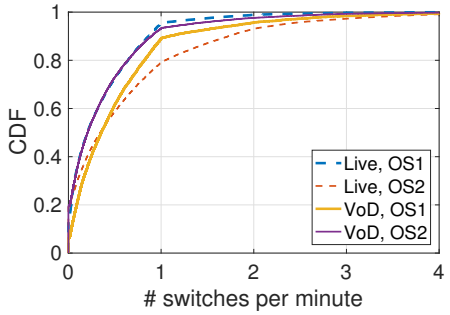


Fig. 5: CDF of number of switches per minute

V. Discussion and Future Work

Encrypted Traffic: Our QoE metrics estimation methodology relies on extracting information from video chunk URI and hence is limited to un-encrypted traffic. Specifically, we extract chunk quality and identifier from the URI. Chunk quality can potentially be estimated from the chunk size, especially for CBR videos. However, chunk identifier is not available from encrypted traffic. We plan to explore alternative ways to infer QoE metrics for encrypted video in the future, by exploiting insights and understanding of HAS traffic offered by Mimic.

Advertisements: We observed that for some of the VoD videos in *VideoApp*, ads are inserted into the playback. These ads are not part of the HAS stream, but downloaded as single files. Furthermore, the request URI for these advertisements did not have any information that would enable us to associate it to its corresponding session. Note that even other video streaming apps serve ads in a manner that is different from the way original content is served. This is because ads are usually served by a third-party content provider. Detecting these ads in a stateless manner and then accounting for them in estimating the QoE metrics remains a challenge.

User interaction: We currently do not track "trick-play" (pause, fast-forward or rewind) by the user during the session. Mimic needs to be modified if we are also to accurately estimate the QoE metrics of these sessions. User interaction makes it harder to model video sessions on the network, as tracking these events though network data can be challenging. We are working on developing techniques for trick-play detection.

Startup time: Mimic currently does not estimate video startup time (VST). VST depends both on the time to download the minimum number of chunks required to start playing as well as other service-specific interactions that take place inside the app. VST estimation is also part of the future work.

VI. Conclusion

We present Mimic, a methodology to estimate video QoE metrics from passive network measurements. The results from large-scale validation show that Mimic can provide a very accurate view of key QoE metrics, namely average bitrate and re-buffering ratio, to a network operator. We do a large-scale quantification of network overhead due to chunk replacement. Our future research will use insights from this study to further improve the video QoE metrics estimation methodology.

Acknowledgments

We thank our shepherd, David Hayes, and the anonymous reviewers for their comments and feedback. This work is funded in part by NSF grant NETS 1409589.

References

[1] "Cisco study," 2017. [Online]. Available: https://goo.gl/uz3SCN
[2] "Conviva: in-app QoE monitoring." [Online]. Available: https://goo.gl/kcLC9D
[3] New Relic, "Mobile APM features." [Online]. Available: https://goo.gl/Z75dTl
[4] J. Jiang, X. Liu, V. Sekar, I. Stoica, and H. Zhang, "EONA: Experience-oriented network architecture," in *Proc. ACM HotNets*, 2014.
[5] K. T. Chen, C. C. Tu, and W. C. Xiao, "OneClick: A framework for measuring network quality of experience," in *Proc. IEEE INFOCOM*, 2009.
[6] D. Joumblatt, J. Chandrashekar, B. Kveton, N. Taft, and R. Teixeira, "Predicting user dissatisfaction with Internet application performance at end-hosts," in *Proc. IEEE INFOCOM*, 2013.
[7] V. Aggarwal, E. Halepovic, J. Pang, S. Venkataraman, and H. Yan, "Prometheus: Toward quality-of-experience estimation for mobile apps from passive network measurements," in *Proc. ACM HotMobile*, 2014.
[8] G. Dimopoulos, I. Leontiadis, P. Barlet-Ros, and K. Papagiannaki, "Measuring video QoE from encrypted traffic," in *Proc. IMC*, 2016.
[9] S. Akhshabi, A. C. Begen, and C. Dovrolis., "An experimental evaluation of rate-adaptation algorithms in adaptive streaming over HTTP," in *Proc. ACM MMSys*, 2011.
[10] Y. Liu, S. Dey, F. Ulupinar, M. Luby, and Y. Mao, "Deriving and validating user experience model for DASH video streaming," *IEEE Transactions on Broadcasting*, 2015.
[11] R. Schatz, T. Hofeld, and P. Casas, "Passive YouTube QoE monitoring for ISPs," in *Proc. IMIS*, 2012.
[12] G. Dimopoulos, P. Barlet-Ros, and J. Sanjus-Cuxart, "Analysis of YouTube user experience from passive measurements," in *Proc. CNSM*, 2013.
[13] Apple, "HLS." [Online]. Available: https://developer.apple.com/streaming
[14] "DASH." [Online]. Available: http://dashif.org/mpeg-dash/
[15] A. Mansy, M. Ammar, J. Chandrashekar, and A. Sheth, "Characterizing client behavior of commercial mobile video streaming services," in *Proc. ACM MoVID*, 2013.