YouTube QoE Estimation from Encrypted Traffic: Comparison of Test Methodologies and Machine Learning Based Models

Irena Orsolic, Mirko Suznjevic and Lea Skorin-Kapov
University of Zagreb, Faculty of Electrical Engineering and Computing, 10000 Zagreb, Croatia
Email: {irena.orsolic, mirko.suznjevic, lea.skorin-kapov}@fer.hr

Abstract—Over the last few years, different client-side QoE monitoring apps have been developed that benchmark the performance of popular video streaming services. Such tools also provide the means for collecting ground truth data when developing models to estimate or classify QoE and various KPIs from encrypted network traffic. We present a client-side YouTube QoE monitoring tool named ViQMon, which extracts YouTube performance data from the official app's Stats for Nerds window, and is applicable on various devices and platforms (Android, iOS). We compare ViQMon to approaches relying on YouTube's APIs, and show relevant differences in buffering and application behavior in cases when videos are embedded and when videos are played in the official YouTube app. We further use ViQMon together with the collection of network measurements in both a laboratory and commercial mobile network to collect a large dataset of almost 500 YouTube videos streamed under different network conditions. The dataset is used to build machine learning based models for estimating QoE and various application-layer KPIs solely from IP-level network traffic features. As such, the approach is applicable in the context of both TLS and QUIC traffic. The paper further compares and analyses the performance of the built models.

Index Terms—QoE estimation, client-side monitoring, YouTube, machine learning, encrypted traffic

I. INTRODUCTION

With the steady rise in global mobile network traffic, to a large extent due to the proliferation of popular video streaming services (e.g., YouTube, Netflix, Facebook video), network operators are faced with the challenge of managing their networks in such a way as to keep their customers satisfied, while efficiently using available resources. A prerequisite for QoE-aware traffic management is the deployment of a QoE monitoring architecture in the network.

From an ISP's point of view, monitoring has become a very challenging task [1], as existing solutions that rely on deep packet inspection are for the most part no longer viable, due to traffic encryption. Assuming limited or no information exchange between ISPs and OTT service providers, ISPs must rely on the monitoring of traffic patterns and statistical traffic characteristics for QoE estimation and root cause analysis of potential quality problems. Such approaches are also dependent on the collection of application level ground truth data for building models that estimate or predict QoE from network traffic metrics. A number of recently published studies rely on machine learning (ML) techniques to find patterns in network-level data that correlate with certain application-level degradations [2]–[5]. Although the general objectives are similar, these QoE estimation solutions can also vary in terms of what is predicted with the models (e.g., quality class [2], [4], [5], specific app-KPIs [3]), and on what time-scale (e.g., per flow [2], per video-session [3]–[5]). What also varies is the model complexity, and resources needed to calculate features from network traffic and run the model on extracted data. As stalling was identified to be the most degrading QoE influence factor [6], there are research efforts focused only on buffer state prediction, with the aim to detect low buffer levels and take a management action that prevents stalling [7], [8]. However, all of the approaches listed above use information from TCP headers, and, in their current state, are not applicable for QUIC (Quick UDP Internet Connections) traffic, which is now predominant in YouTube. Regarding other video streaming services, we note that Facebook often uses its own protocol called ZERO for delivering video, while Netflix still uses TCP/TLS (Transport Layer Security).

While there are differences on the protocol level, most of today's popular video services implement the HTTP Adaptive Streaming (HAS) paradigm, and their performance on the application level can be monitored taking into account well-known QoE influence factors, such as initial delay, stalling, and audiovisual quality level [10], [11]. The mapping of these factors to QoE values has been studied in [22], [23], and a standardized model has been published in [24]. To a large extent, research on the QoE monitoring of such services so far has focused on YouTube. A number of studies have analysed YouTube's client behavior under various network conditions, aimed at providing an understanding of the underlying service behavior and adaptation mechanisms [4], [12]–[14].

Over the past years, different client-side apps that monitor YouTube’s performance have been developed, both commercial [15], [16], and non-commercial [4], [17], [19]. The purpose of such apps is multifold: besides obtaining ground-truth information for QoE estimation or prediction, they can be used for benchmarking network performance, analysing service behavior, etc.

For YouTube, client-side monitoring approaches found in literature can be divided into three groups. The first group are apps that embed YouTube videos using the IFrame API (e.g., YoMoApp [17], YouQ_IF [4]). Although the IFrame API
offers a lot of information about playback quality parameters, the question is to what extent do such client-side apps behave in the same way (referring to video streaming and adaptation logic) as the official YouTube apps that are most commonly used by YouTube viewers on smartphones. Another group of YouTube performance monitoring apps are apps that embed YouTube videos using the native YouTube API for Android (e.g., YouQ-AA [20]). However, a native API is only available for Android, and the number of performance metrics that can be retrieved through the API is limited, as compared to the IFrame API. The third approach is to perform monitoring using the official YouTube application. An example of such an approach can be found in [21], where the authors captured the screen while playing YouTube videos and used image processing techniques to detect the spinner that indicates stalling or initial buffering.

In this paper we present our Video Quality Monitor (ViQMon) methodology, developed in cooperation with Ericsson Nikola Tesla, that enables us to use the official YouTube app for measuring application-level KPIs. The approach relies on recording the screen while using the YouTube app, with the Stats for Nerds option enabled to display meta-data regarding video playback. The video recording is processed using Optical Character Recognition (OCR) techniques to derive QoE-relevant KPIs. The approach is platform-independent and enables monitoring on different operating systems.

As a first comparison, we compare the ViQMon approach to methodologies that rely on available APIs (namely YouQ IF [4] and YouQ-AA [20]) to obtain application-level performance data. The approaches are compared with regards to application behaviour and traffic characteristics. The second contribution is the evaluation of a set of machine learning models that estimate QoE and a variety of application-level KPIs based solely on IP-level traffic characteristics. For ground-truth QoE calculation we rely on the multidimensional QoE model for HAS published in ITU-T Recomm. P.1203 [24].

The paper is organized as follows: In Section II, we describe the ViQMon monitoring methodology, the process of data collection, and the process of ML-based model development and testing. Section III presents results, first comparing different client-side monitoring approaches, and then presenting the performance results of developed ML-based QoE and KPI estimation models. Conclusions are given in Section IV.

II. METHODOLOGY

A. ViQMon app-KPI monitoring approach

As opposed to QoE monitoring apps relying on YouTube’s APIs (YouQ [4], YoMoApp [17]), with ViQMon we obtain performance information directly from the official YouTube app. The approach is based on collecting screen recordings of videos played using the YouTube app with the Stats for Nerds option enabled and displayed. This option displays video meta-data as an overlay over the video. Information displayed includes video dimension, video resolution, unique ID, framerate, buffer state, etc. Recordings are then analysed using Tesseract OCR engine to read data necessary to derive application-layer KPIs per video viewing session, such as initial delay, average stalling count and duration, and percentage of time played on a given video quality level (Figure 1).

The main advantage of the approach is monitoring of the official YouTube app, as opposed to approaches that rely on embedding YouTube videos. In this way, collected data is realistic and reliable. Another advantage is that data analysis is done offline, on a desktop computer running ViQMon, while on mobile devices we only capture the screen recording, using any screen recording app. For this reason, ViQMon is applicable for various devices, OS versions, etc. On the other hand, the main challenges concerning the approach are related to collected data size. To reduce the usage of storage space on a mobile device, in our experiments we record the screen with a framerate set to the lowest value available in the screen recording app, bearing in mind that Stats for Nerds data is refreshed once per second. Also, lowering screen capturing resolution could reduce ViQMon processing time, but could also result in lowered OCR precision.

B. Datasets collection

Dataset D1: Comparison of monitoring apps: To gain insight into the differences in YouTube’s behaviour when different client-side QoE monitoring applications are used, we conducted measurements in a lab environment using YouQ_IF, YouQ_AA, and ViQMon. In all three cases, the same client device was used (Samsung S6), the same playlists were played in landscape mode, and the same network conditions were emulated. The client device is connected to the Internet using a WiFi connection, and various bandwidth limitations were emulated using the freely available tool IMUNES\(^2\), ranging from 0.5 Mbps to 5 Mbps, with a 0.5 Mbps step. At each bandwidth level, 10 videos were played differing in duration (63 to 435 seconds), popularity (number of likes), and type (music, sports, comedy). At the same time, we captured network traffic passing through a designated router. The final dataset consists of both application- and network-layer data corresponding to 300 played videos (10 videos streamed over 10 different bandwidth conditions, and repeated for 3 different measurement apps).

Dataset D2: ViQMon data collection (lab/WiFi): Dataset D2 was collected to build ML models that estimate application-level performance from network-level metrics. For

\[^1\]https://github.com/tesseract-ocr
\[^2\]http://imunes.net/
obtaining ground truth data, we used the ViQMon approach. Important to consider is the need to observe different quality degradations that might occur at the client, so as to enable ML algorithms to learn from examples, and finally to enable built models to recognize such events when the model is deployed in a network. To emulate conditions that resemble those found in a real network, we scripted the IMUNES emulator to use the 4G/LTE bandwidth logs\(^3\) published in [25]. The logs cover 5 hours of active monitoring, and were collected in different mobility scenarios: on foot, bicycle, tram, train, and car. An excerpt of the logs is depicted in Figure 2.

We created playlists on YouTube containing a total of 100 videos, with video duration distribution depicted in Figure 4a, and played them on Samsung S6 while connected to the network via a WiFi access point and with the aforementioned bandwidth envelopes imposed. As bandwidth observed in the logs is generally high and short outages would not induce quality degradations (due to content being buffered on the client), we repeated the playing of 100 videos three more times, while dividing the original bandwidth by factors of 10, 20, and 30. In this way we obtained data from video streaming sessions with quality degradations, while keeping virtually realistic bandwidth fluctuations. As 2 of the 100 videos were deleted from YouTube sometime during the experiments, Dataset D2 contains data corresponding to 98 played videos.

**Dataset D3: ViQMon data collection (mobile network):** Dataset D3 was collected in September 2017 in the network of a large European ISP, with traffic traces obtained from a probe in the mobile network, and videos viewed on a Sony Xperia X device. The dataset contains application- and network-layer data corresponding to the streaming of 105 YouTube videos, with app-layer KPIs derived using ViQMon. As the network coverage was very good in the city area covered by the probe used to capture network traffic, we limited network type in certain scenarios to 2G or 3G on the smartphone so as to invoke degradations. The final obtained Dataset D3 was used to test the ML models built using Dataset D2.

**C. QoE estimation from network-level data**

To build ML models that estimate YouTube performance, we first calculate a series of metrics (predictors) from network traffic, related to the streaming of each video, and label these metrics with ground truth application-level data (target). The dataset is then provided to ML algorithms to build models that are able to estimate the target variable using only the predictors. We assess ML models built to estimate different application-level parameters using different types of models. Target application-level parameters are listed as follows:

- MOS, as calculated by using the ITU-T Recommm. P.1203 [24], and the implementation published online\(^4\) [26],
- 2 MOS classes, based on the output of P.1203 (“high”, “low”), where instances with MOS ≥ 3.5 are labelled as “high”, and “low” otherwise,
- 3 MOS classes, based on the output of P.1203 (“high”, “medium”, “low”), where instances with MOS ≥ 4.0 are labelled as “high”, MOS ≥ 3.0 and < 4.0 as “medium”, and MOS < 3.0 as “low”,
- 2 video resolution classes (“hd”, “sd”), where “hd” means that the video was played in 720p or a higher resolution for most of its duration, “sd” otherwise, and
- 2 stalling occurrence classes (“yes”, “no”), which denote if there were any stalling events (excluding initial delay).

We note that the model from ITU-T Recomm. P.1203 has been validated for H.264 video, while for VP9 (which we often observed in YouTube) we used the Non-Standard Codec Extension for ITU-T P.1203\(^5\).

We list the 24 metrics calculated from network traffic, based on which the prediction is performed:

- \(\text{averageThroughput}, \text{avgPacketSize}, [\text{mean}, \text{hMean}, \text{median}, \text{lowMedian}, \text{highMedian}, \text{groupedMedian}, \text{min}, \text{max}, \text{stDev}, \text{pStDev}, \text{pVar}] \text{SizeIn5sIntervals} / \text{SizeIn1sIntervals}\).

Most of the features are based on calculating mathematical statistics on lists that store the amount of data downloaded by the client in 1- or 5-second periods. Although most of the features’ names are straightforward, we further clarify the following: in our feature set “hMean” stands for harmonic mean, “pStDev” for population standard deviation, and “pVar” for population variance.

The data used for ML model development (ML dataset\(^6\)) was prepared from Dataset D2 and contains 394 rows, corresponding to each played video. Each row in the file consists of metadata about the experiment and specific video, calculated network traffic features, and application-layer target variables. We analysed this dataset using the Weka tool\(^7\) and used it to build ML models that estimate MOS, MOS classes, longest played resolution, and stalling occurrence, using different algorithms available in Weka. We based our classification models on rule- and tree-based algorithms, as these were proven to work well for this type of problem [3], [4]. For building regression models that predict MOS value, we tried various algorithms: linear regression, k-nearest neighbors, REPtree, sequential minimal optimization (SMO), and multilayer perceptron. Before building a model, features were subset, for each of the used algorithms using Wrapper method. Model performance was first assessed using 10-fold

---

3http://users.ugent.be/~jvdrhoof/dataset-4g/
4https://github.com/itu-p1203/itu-p1203
5https://github.com/Telecommunication-Telemedia-Assessment/itu-p1203-codecextension
6https://www.fer.unizg.hr/qmanic/data_sets
7http://www.cs.waikato.ac.nz/ml/weka/
The AA app does not log initial delay distribution. If we can only assume that quality level data could not be observed that the amount of data is approximately the same in all of the experiments in which we used YouQ apps, (2) YouQ_AA app does not log information about played video resolution, as it is not provided by the YouTube Android API, (3) when using YouQ_IF, we observe TCP as the underlying transport protocol, whereas in experiments with YouQ_AA and ViQMon QUIC is used, and (4) all ViQMon data is collected with a precision of 1 s, as this is the rate at which Stats for Nerds data is refreshed.

Results are summarized in Figures 3a and 3b. Figure 3a shows the average per-video amount of data downloaded by the client application for each experiment (one experiment corresponds to 10 videos played at a given available bandwidth level, with 10 experiments conducted per client app). It can be observed that the amount of data is approximately the same in all of the experiments in which we used YouQ_IF (except on 0.5Mbps). From our logs, it is visible that the highest observed resolution played by the IFrame-based app was 480p, although higher video resolutions were available. On the other hand, the amount of data downloaded in the cases of YouQ_AA and ViQMon was significantly higher, with ViQMon log data showing higher resolutions being played as compared to YouQ_IF. However, as quality level data could not be collected using YouQ_AA, we can only assume that quality levels roughly correspond to those observed by ViQMon.

With Figure 3b we depict the differences in initial delay achieved with the three apps. It is clear that apps embedding YouTube videos introduce additional delay, as compared to the ViQMon case, where the official YouTube app is used. We note that with ViQMon, data is collected with 1s precision, but even if calculated average initial delays shown in Figure 3b are increased by one second (worst case scenario), the delay is still shorter than in the embedded video case. As bandwidth was kept constant and stable in a single experiment, we observed only 11 stalling events in total, during the playing of 300 videos. All of the stalling events happened in experiments with ViQMon, and when bandwidth was lower than 2.5Mbps.

While more detailed comparison results are omitted due to space limitations, we conclude that obtained results indicate significant differences in streaming behavior, and consequently traffic characteristics, across different monitoring approaches. Researchers and practitioners interested in monitoring application-level data should be aware of these differences, in particular when using this “ground truth” data for benchmarking network performance, or building QoE estimation models relying on encrypted traffic features. Given the widespread use of the official YouTube app on smartphones (Android and iOS), we further base our measurements and results on this case.

B. YouTube performance estimation using ML models

As different application-level metrics, measured at different granularities, may be relevant for use in QoE management strategies, the idea of this study was to assess to what extent different application-level KPIs and overall MOS can be estimated from IP-level data. Although the number of models we wanted to build was initially higher, after inspecting the collected dataset, it became clear that building some models is redundant. For example, initial delays observed in the dataset were short enough not to impact QoE (according to existing QoE models from literature) in almost all of the video streaming instances (Figure 4b), and thus classifying videos with 2s initial delay into a “long” initial delay class would not make sense. Finally, we decided to build models based on different algorithms that estimate MOS, classify MOS into 2 classes, and classify stalling in 2 classes. Using Dataset D2 we built 17 different ML models, 12 of which are classification, and 5 are regression models.

In line with our expectations, the performance of regression models was not very good. With RMSE at 0.6919 when predicting overall MOS, the best performing regression model was built using the SMO algorithm. On the other hand, for ISPs, it may likely be sufficient to classify MOS into a number of classes (such as “high”, “medium”, “low”), without the need for more precise estimations. Also, it may be more relevant to classify MOS and factors that influence MOS, to understand
<table>
<thead>
<tr>
<th>Target variable</th>
<th>Algorithm</th>
<th>Selected attributes (predictors)</th>
<th>Accuracy [%]</th>
<th>Precision</th>
<th>Recall</th>
</tr>
</thead>
<tbody>
<tr>
<td>MOS {high, low}</td>
<td>OneR</td>
<td>stDevSizeIn1sIntervals</td>
<td>80.203</td>
<td>h: 0.811</td>
<td>l: 0.790</td>
</tr>
<tr>
<td></td>
<td>J48</td>
<td>stDevSizeIn1sIntervals, avgPacketSize</td>
<td>82.741</td>
<td>h: 0.830</td>
<td>l: 0.824</td>
</tr>
<tr>
<td></td>
<td>RF</td>
<td>avgSizeIn5sIntervals, hMeanSizeIn5Intervals, minSizeIn5Intervals, stDevSizeIn1sIntervals, hMedianSizeIn1sIntervals</td>
<td>81.2183</td>
<td>h: 0.828</td>
<td>l: 0.792</td>
</tr>
<tr>
<td>MOS {high, medium:m, low:l}</td>
<td>OneR</td>
<td>maxSizeIn1sIntervals</td>
<td>70.3046</td>
<td>h: 0.826</td>
<td>m: 0.489</td>
</tr>
<tr>
<td></td>
<td>J48</td>
<td>avgPacketSize, pStDevSizeIn5sIntervals, stDevSizeIn1sIntervals</td>
<td>70.8122</td>
<td>h: 0.847</td>
<td>m: 0.494</td>
</tr>
<tr>
<td></td>
<td>RF</td>
<td>avgSizeIn5sIntervals, minSizeIn5Intervals, maxSizeIn5sIntervals, mediumSizeIn1sIntervals, maxMedianSizeIn1sIntervals</td>
<td>71.8274</td>
<td>m: 0.568</td>
<td>l: 0.625</td>
</tr>
<tr>
<td>Longest resolution {hd, sd}</td>
<td>OneR</td>
<td>maxSizeIn1sIntervals</td>
<td>81.4721</td>
<td>hd: 0.863</td>
<td>sd: 0.758</td>
</tr>
<tr>
<td></td>
<td>J48</td>
<td>maxSizeIn1sIntervals, medianSizeIn1sIntervals</td>
<td>83.5025</td>
<td>hd: 0.849</td>
<td>sd: 0.815</td>
</tr>
<tr>
<td></td>
<td>RF</td>
<td>averageThroughput, avgSizeIn5sIntervals, lowMedianSizeIn5sIntervals, pStDevSizeIn5sIntervals</td>
<td>82.2335</td>
<td>hd: 0.855</td>
<td>sd: 0.780</td>
</tr>
<tr>
<td>Stalling occurrence {yes, no}</td>
<td>OneR</td>
<td>medianSizeIn5sIntervals</td>
<td>69.2982</td>
<td>y: 0.625</td>
<td>n: 0.853</td>
</tr>
<tr>
<td></td>
<td>J48</td>
<td>highMedianSizeIn1sIntervals</td>
<td>70.1754</td>
<td>y: 0.640</td>
<td>n: 0.821</td>
</tr>
<tr>
<td></td>
<td>RF</td>
<td>hMeanSizeIn1sIntervals, highMedianSizeIn1sIntervals, minSizeIn1sIntervals, stDevSizeIn1sIntervals</td>
<td>69.2982</td>
<td>y: 0.672</td>
<td>n: 0.714</td>
</tr>
</tbody>
</table>

The reasons behind degradations, than to use computational resources on complex regression models. We therefore further focus our results analysis on classification models only.

The ML dataset used for building classification models is presented in Figure 5. Figure 5a shows the distribution of MOS values in the dataset, as calculated using the ITU-T P.1203 model. In Figure 5b we show the number of instances that belong to the classes of each type of classification model. Given the distribution of stalling classes, we subset the dataset to balance the number of instances in each class. For each of the target variables, we built models using three algorithms: OneR, J48, and Random forest. The results in terms of accuracy, precision, and recall are listed in Table I.

From the Table, it is visible that classifying MOS and video resolution into 2 classes achieves high accuracy (up to 83%). When classifying MOS into 3 classes, classification errors usually happen between classes “medium” and “low”, while instances from class “high” seem easily distinguished by the model. Regarding the algorithms used to build models, we observe that more complex models introduce only a slight increase in performance, as compared to the OneR rule-based model. Complex models require more resources when utilized in a real network, and it seems questionable if it is worth using more resources for only a slight increase in accuracy.

For additional validation purposes, built models were also tested on data collected using probes in an operational mobile network. However, due to the fact that the smartphone used for testing was either connected to 4G or 2G, calculated MOS (based on app-level measurements and the ITU-T P.1203 model implementation) was observed to be either very high or very low (Figure 6a). Ground-truth classes in Dataset D3 for each of the model types are shown in Figure 6b. Because

![Fig. 5: Distribution of target variables in Dataset D2 (WiFi).](image1)

![Fig. 6: Distribution of target variables in Dataset D3 (mobile).](image2)
TABLE II: Machine learning results - mobile.

<table>
<thead>
<tr>
<th>Target variable</th>
<th>Accuracy [%]</th>
</tr>
</thead>
<tbody>
<tr>
<td>MOS {high, low:1}</td>
<td>up to 100</td>
</tr>
<tr>
<td>MOS {high, medium:1, low:1}</td>
<td>up to 95.2381</td>
</tr>
<tr>
<td>Longest resolution {ld, sd}</td>
<td>up to 94.2857</td>
</tr>
<tr>
<td>Stalling occurrence {yes, no}</td>
<td>up to 76.3158</td>
</tr>
</tbody>
</table>

of such distributions of MOS and QoE-influencing factors, the results when classifying the data from dataset D3 using models built on dataset D2 are too optimistic (Table II). While these results provide initial insights, a larger dataset collected in a mobile network, covering a wider variety of QoE degradations, is needed to obtain a more realistic picture of model applicability.

IV. CONCLUSIONS

ISPs are increasingly facing challenges related to monitoring QoE for popular OTT video streaming services, in particular given the widespread use of traffic encryption. We address these challenges in the context of YouTube, and provide the following contributions. First, we address the challenge of client-side monitoring of app-level KPIs impacting QoE, and show that different approaches used in existing monitoring apps result in different YouTube application behavior and traffic characteristics. More specifically, approaches relying on embedding YouTube videos for test purposes and utilizing available YouTube APIs exhibit different characteristics as compared to the official YouTube app. We therefore propose a monitoring approach that utilizes OCR libraries to extract application-level KPIs from video recordings (with the Stats for Nerds option enabled) captured while watching YouTube videos with the official app. A further benefit of this approach is that it is applicable across different platforms (Android and iOS). An improvement being addressed in our ongoing work is to extract the text from Stats for Nerds while avoiding video recording, thus avoiding large amounts of collected data and long processing time.

Second, we address the challenge of classifying QoE and KPIs using ML-based models with input relying solely on IP-level network traffic features. As such, the models may be trained using both TCP/TLS and QUIC traffic. Results show promising accuracy when classifying MOS in 2 or 3 classes, longest resolution played into 2 classes, and stalling occurrence into 2 classes. Finally, built models were used to classify MOS and KPIs using a dataset collected in an operational mobile network. While initial results showed promising model applicability, further tests are needed involving a dataset with more diverse application quality degradation scenarios. Ongoing work will also address methods for improving test methodologies, consideration of a wider range of use case scenarios (e.g., involving user interactions when viewing YouTube videos), and improvement of existing models.

ACKNOWLEDGEMENT

This research has been conducted in the scope of the project “QoMoVid: QoE Monitoring Solutions for Mobile OTT Video Streaming” funded by Ericsson Nikola Tesla, Croatia. This work has also been supported in part by the Croatian Science Foundation under the project UIP-2014-09-5605 (Q-MANIC). The authors would like to thank Ivan Bartolec and Marko Jurlkovic for their help in developing the used tools.

REFERENCES