Machine Learning based KPI Monitoring of Video Streaming Traffic for QoE Estimation

Özge Celenk
Technische Universität Chemnitz
Chemnitz, Germany
oezge.celenk@etit.tu-chemnitz.de

Thomas Bauschert
Technische Universität Chemnitz
Chemnitz, Germany
thomas.bauschert@etit.tu-chemnitz.de

Marcus Eckert
Technische Universität Chemnitz
Chemnitz, Germany
marcus.eckert.tuc@gmail.com

ABSTRACT

Quality of Experience (QoE) monitoring of video streaming traffic is a crucial task for service providers. Nowadays it is challenging due to the increased usage of end-to-end encryption. In order to overcome this issue, machine learning (ML) approaches for QoE monitoring have gained popularity in the recent years. This work proposes a framework which includes a machine learning pipeline that can be used for detecting key QoE related events such as buffering events and video resolution changes for ongoing YouTube video streaming sessions in real-time. For this purpose, a ML model has been trained using YouTube streaming traffic collected from Android devices. Later on, the trained ML model is deployed in the framework's pipeline to make online predictions. The ML model uses statistical traffic information observed from the network-layer for learning and predicting the video QoE related events. It reaches 88% overall testing accuracy for predicting the video events. Although our work is yet at an early stage, the application of the ML model for online detection and prediction of video events yields quite promising results.

CCS CONCEPTS

• Networks • Network Services • Network Monitoring

KEYWORDS

Machine learning, QoE, QoE monitoring, video streaming

1 INTRODUCTION

In the recent years, the demand for video streaming services increased exponentially, and by September 2019 video streaming had 60% share of the total Internet traffic [1]. In order to guarantee a satisfactory video Quality of Experience (QoE) for the users, network operators have to carefully monitor it and tune their network accordingly so as to avoid capacity bottlenecks. Assessing the user-perceived video QoE is a challenging task since it is subjective, and the operators do not have direct access to end-user devices to acquire video quality information. A feasible approach is to continuously monitor network-layer traffic statistics to infer video QoE metrics and apply traffic control mechanisms to influence the QoE. However, due to the increased usage of end-to-end encryption and new streaming technologies such as Dynamic Adaptive Streaming over HTTP (DASH), inferring the QoE from network-layer traffic information is getting more complex. Although streaming techniques changed in the recent years, it is still true that the most severe degradations in video QoE are caused by interruptions during video playback. This effect is also known as stalling event. Since stalling events have an immediate negative impact on the user's satisfaction with the video session, network operators need to take countermeasures by detecting them as soon as possible - ideally, before they occur. One way to overcome the challenge of end-to-end encryption is to monitor traffic statistics and apply machine learning (ML) techniques.

In this paper, we present a machine learning based framework which monitors the network traffic and predicts buffering events, as well as video resolution changes in YouTube streaming sessions in real-time (i.e. online). For this purpose, we first trained a machine learning model that is capable of predicting buffering events from basic statistical features (such as packet sizes and packet inter-arrival times) of the encrypted video streaming traffic. Furthermore, we designed a framework which contains a pipeline written in Python that constantly monitors the video traffic in the network. Each time slot (of one second duration) it makes a prediction on whether the video is playing or buffering or whether there is a video resolution change by analyzing the traffic in both the current and the previous time-slot.

2 RELATED WORK

Recent studies regarding QoE estimation of video streaming services confirm that the most QoE impacting factors are the number of stalling events, and the video resolution [2] [3] [4]. Furthermore, the time between the stalling events and the duration of an ongoing stalling event are also significant. Our

work at its current stage mainly takes the stalling events and the video resolution as QoE metrics.

As machine learning (ML) approaches in general gained popularity over deep packet inspection (DPI) based methods due to the increased usage of end-to-end encryption, several studies aimed to estimate the QoE of video streaming via ML. Some of these studies propose offline systems that are able to estimate the QoE of a video session only after the session is completed. For example, the authors in [5] developed a system named YouQ, which includes tools for monitoring and analysis of application-layer Key Performance Indicators (KPIs) such as video resolution and the number and duration of stalling events. As the entire video trace has to be sent to the server to obtain a prediction, this approach is not feasible for real-time QoE estimation. Other studies were performed with the target of online detection of stalling events and other KPIs that are important for QoE estimation. For instance, in [6] and [7] the Vicrypt system was introduced as an ML-based framework to monitor stalling events. Vicrypt analyzes a video data stream with one second granularity. It establishes three different time-windows and determines the values of the features based on the current time window, as well as on the previous T seconds (trend window), and then aggregates the results into a session time window. After this, a prediction is made for detecting stalling events using ML models. Later on, the Vicrypt system was extended to not only monitor stalling events, but also video resolution and bitrate changes [8]. The presented results indicate that this method exhibits some weaknesses in predicting buffering events. This might be caused by a class imbalance in the used dataset due to the adaptation strategy of YouTube, which leads to biased ML models.

Our framework takes Vicrypt as a reference, but different from this approach, we try to overcome the class imbalance problem in the dataset by using the Synthetic Minority Over-sampling Technique (SMOTE) approach during the model training. This technique basically is an oversampling method to balance the number of samples from each class. Furthermore, we deploy our trained ML model in a multiprocessing pipeline architecture to enable real-time decision making.

3 METHODOLOGY

3.1 Overview of the Proposed Method

Our ML-based real-time video QoE monitoring framework consists of a Python3 based multiprocessing pipeline. Multiprocessing is used for allowing the concurrent execution of multiple program modules of the framework for real-time usage. In the current version of the framework, four processes are defined (data collector, statistics collector, data preprocessor and ML-based predictor), where each process runs on a different CPU core concurrently – see Figure 1. Each second an estimation is made on whether the current video session is playing or buffering, or whether a video resolution change occurs.

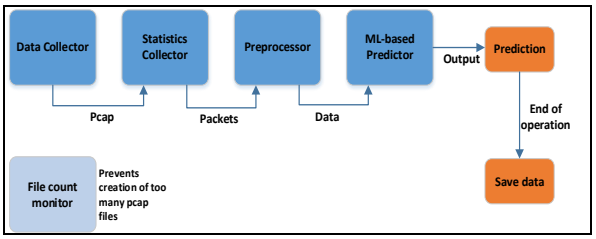

Figure 1. Workflow Overview

As depicted in Figure 1, the regular workflow is as follows: The tcpdump based data collector process starts capturing the traffic and creates a new pcap file every second. Upon generation of each pcap file, the statistics collector is triggered which imports the packets from the pcap file into the preprocessor. The preprocessor extracts the values of the statistical features from the encrypted packet stream for each second. It also combines the observed values of the features from the previous time interval (t-1) and the current time interval (t) to establish a trend window. This data is then fed into a previously trained ML-based predictor which makes an estimation on whether the video is buffering or playing, or whether there is a video resolution change in the current time interval (t). At the end, the collected data and the corresponding predictions are saved in a csv file for further analysis.

3.2 Experimental Setup and Data Collection

In order to effectively collect training data from YouTube, a custom test automation tool named QTS App is used [9]. Using this app, tasks such as video streaming or Web browsing can be automated via assigning mobile devices to previously prepared campaigns. In our case, two Android devices were assigned to the YouTube streaming campaign to watch ten different videos repeatedly under various network conditions. Once a mobile device is assigned to a YouTube streaming task, the device receives YouTube traffic for a predetermined timespan, and the traffic is captured using tcpdump. During a video session, the QTS App also leverages the YouTube's IFrame API to record application layer activities such as video events ('Playing', 'Buffering', 'Resolution Changes') together with their corresponding timestamps and stores them in a csv file. These events are later used for establishing the ground truth for labelling the dataset. At the end of every video session, both the pcap file and the recorded video events (csv file) are saved. For acquiring a balanced dataset which includes enough samples of all types of events ('Playing', 'Buffering', 'Resolution Changes'), an increase of 'Buffering' and 'Resolution Changes' is enforced by imitating bad network conditions. For this purpose, a Raspberry Pi 3 is set up as network bridge, and the Traffic Control tool (tc-tool) is used to enforce various network conditions. Six different scenarios are applied during the data collection phase:

1.Reference: No manipulation, stable network connection with 50/10 Mbit/s (downlink/uplink)
2.Bandwidth Limit: Limit downlink bandwidth between 500kbit/s and 3Mbit/s
3.Packet Delay: Add 50 ms (average) delay in downlink
4.Packet Loss: Create 5% packet loss in downlink
5.Packet Corruption: Corrupt 5% of downlink packets
6.Packet Loss II: Create between 25% and 50% packet loss in downlink
Once enough data is collected, an offline preprocessing is performed for obtaining network layer information such as packet statistics (per second time interval) and combining the application-layer video events with the corresponding network-layer metrics to create the training dataset.

3.3 Data Preprocessing

Since the video traffic is encrypted, only some basic statistical information is available in the network layer such as packet count and packet size, as well as packet inter-arrival time. Using the *dpkt* library (which is available in Python), we developed a program which calculates the following basic statistics features for each time interval t (of one second duration):
Total number of packets, number of uplink packets, number of downlink packets
Total number of bytes, total number of uplink bytes, total number of downlink bytes
Averages of the packet sizes (in uplink and downlink)
Maximum and minimum packet sizes (in uplink and downlink)
Averages of the overall, uplink and downlink packet Inter-Arrival times
These 15 features are then matched to the corresponding video events observed from the QTS App by comparing their timestamps. Afterwards the dataset is purged to remove corrupt data points. The next important step in the preprocessing phase is to merge the features that were derived in the previous time interval (t-1) and the current time interval (t) – by that a trend window is established. This results in a total of 30 features (15x2) that are used for describing each second of the video sessions in terms of network layer traffic statistics. Together with the corresponding application layer events (i.e. video events) that are used for labeling, the original dataset for ML training and testing is created.
Although bandwidth throttling is applied, the created dataset turns out to be still imbalanced. The number of samples related to a 'Playing' event is much higher than the number of samples related to a 'Buffering' or 'Resolution Change' event. Due to the video quality adaptation strategy applied by YouTube, it is hardly possible to observe more 'Buffering' events. In order to avoid a biased model, after setting 20% of the initial data aside for testing, the SMOTE algorithm from the imbalanced-learn toolset [10] is applied on the remaining dataset to generate more 'Buffering' and 'Resolution Change' (Quality Upgrade and Quality Downgrade) event samples by oversampling the available data. A detailed description of the SMOTE algorithm can be found in [11]. Before the application of SMOTE the dataset has 12016 data points ('Playing': 9214, 'Buffering': 1983, 'Quality Downgrade': 455, 'Quality Upgrade': 364) in total. After oversampling with the SMOTE algorithm, each label has an equal number of samples (9214), and the dataset comprises 36856 data points in total. As a last step before the ML model training, the entire data is normalized. The reason behind this is that the variations in the range of the independent values (features) might lead to a biased model. Therefore, each feature value is transformed to be in a range between 0 and 1.

3.4 ML Model Training and Testing

The machine learning component of our framework is executed using the open-source scikit-learn library which contains efficient tools for machine learning applications [12]. The detailed explanations of the ML algorithms are available in [13].
In order to investigate which ML model is the most suitable for our purpose, the following five ML models that are frequently applied for classification tasks were trained and cross-validated: Random Forest, Logistic Regression, K-nearest Neighbors, Decision Tree and SVM. According to our results, the Random Forest model reaches 96.34% training accuracy, followed by the Decision Tree Algorithm with 93.45%. The training of the Decision Tree model is much faster compared to the Random Forest model, but the latter shows a slightly higher accuracy.
Typically, a Random Forest model is built by using a certain number of decision trees that can be specified - this can potentially affect the model accuracy. Using validation curves, the optimal number of trees for our model is determined as 20. During the model training, a 10-fold cross validation technique is used where the training dataset is partitioned into 10 subsets. Out of the 10 subsets, a single subset is set aside for testing the model, and the remaining 9 subsets are used as training data. The training accuracy is then determined by calculating the average of the results from each fold. The reason behind the usage of the cross-validation technique is to minimize the variance and validate the model performance on multiple subsets of data.

4 RESULTS AND DISCUSSION

4.1 Offline Video QoE Estimation Results

The first round of tests is conducted offline with the initial 20% data that was set aside for testing. In this case, the data was already preprocessed and labelled and the whole batch of test data is fed into the prediction model.
Table I shows the classification performance of the Random Forest model w.r.t. different metrics.

Table I. Classification Performance of the Random Forest Model

Class	Precision	Recall	F1-score
Buffering	0.80	0.78	0.79
Playing	0.95	0.94	0.94
Quality Down	0.36	0.38	0.38
Quality Up	0.86	0.88	0.88

By definition, precision is the percentage of predictions which are relevant to the respective class, while recall refers to the percentage of total relevant results correctly classified by the model. F1-score is the weighted average of the precision and recall values - it can be interpreted as the balance between precision and recall. From Table I it can be seen that the model performs well at detecting the 'Playing' events and the video 'Quality Upgrade' events. However, it struggles to classify the 'Quality Downgrade' events in the test dataset. One reason for the low score of the 'Quality Downgrade' class could be the initial imbalance in the dataset. Using the SMOTE method seems to have a positive effect for the detection of 'Quality Upgrade' events, but detecting the downgrades still seems to be a challenge. Overall, the Random Forest model reached 88.91% testing accuracy, which is slightly lower than the training accuracy.

4.2 Online Video QoE Estimation Results

In the online scenario, the model is tested with data on-the-fly by using the whole pipeline but deploying a simplified version of the prediction model which detects 'Playing' and 'Buffering' events only. The real video events are detected by the QTS App in order to evaluate the predictions.

The obtained results are quite promising: many 'Playing' and 'Buffering' events are correctly detected while the video is running - however, there are more misclassified events compared to the offline case. Table II contains the results of five test cases. It shows the real events during each test run and the predictions made by the simplified model which was deployed in our pipeline.

Table II. Example Results for Online Testing

	Real Events			Predicted Events		
	Playing Dur. (s)	Buffer. Dur. (s)	Buffer. (%)	Playing Dur. (s)	Buf. Dur. (s)	Buffer. (%)
1	20	33	**62**	32	21	**39**
2	69	14	**16**	75	8	**9**
3	88	7	**7.3**	90	5	**5.2**
4	69	25	**26.5**	72	22	**23.4**
5	90	2	**2.1**	87	5	**5.4**

In the best case (Test 3), the model is able to detect almost all 'Buffering' events. On the other hand, in Test 1, the percentage of the detected buffering events is not as high as in the true case. One way to solve this issue is gathering more training data samples, possibly from more diverse network conditions. The performance might be further improved by considering a 'Stalling' event occurrence when 'Buffering' events are predicted for n consecutive time intervals. We also observed that the model's online testing accuracy is not as high as the offline accuracy since is not perfectly prepared to classify new samples that might not follow the same statistics as the training data. Further tests and results which will also include the online prediction of 'Resolution Change' events will be presented as follow-up work.

5 CONCLUSION

In this paper we present a ML framework for the real-time prediction of events that occur in video streaming. The prediction of these events allows to estimate the QoE of encrypted video traffic streams. Our proposed pipeline is active throughout the streaming session and predicts which event is happening in each second of the ongoing streaming session. In an offline setting it achieves 96.3% training and 88.9% testing accuracy, respectively and it also yields promising results in the online case. Compared to the state-of-the-art work our ML model is deployed within a pipeline architecture that serves as a complete tool for video KPI monitoring and it is also tested using data on-the-fly. Our observations also indicate that the model needs more data for training in order to better adapt and improve its accuracy in the online case. Furthermore, expanding the model with other important video streaming QoE metrics such as the initial delay of the videos is planned for future work.

REFERENCES

[1] Global Internet Phenomena Report 2019: 2019. https://www.sandvine.com/global-internet-phenomena-report-2019. Accessed: 2020- 10- 14.

[2] Marcus Eckert, Thomas Martin Knoll, Florian Schlegel. 2013. Advanced MOS calculation for network based QoE Estimation of TCP streamed Video Services, In ICSPCS'13, Carrara.

[3] Michael Seufert, 2018. Quality of Experience and Access Network Traffic Management of HTTP Adaptive Video Streaming, PhD thesis. In ACM SIGMultimedia Records 10, 3, 13-13.

[4] Michael Seufert, Tobias Hoßfeld, Christian Sieber. 2015. Impact of intermediate layer on quality of experience of HTTP adaptive streaming. In CNSM'15, Barcelona.

[5] Irena Orsolic, Dario Pevec, Mirko Suznjevic, Lea Skorin-Kapov. 2016. YouTube QoE Estimation Based on the Analysis of Encrypted Network Traffic Using Machine Learning. In IEEE Globecom Workshop, Washington, DC.

[6] Michael Seufert, Pedro Casas, Nikolas Wehner, Li Gang, Kuang Li. 2019. "Stream-based Machine Learning for Real-time QoE Analysis of Encrypted Video Streaming Traffic. In ICIN'19, Paris.

[7] Michael Seufert, Pedro Casas, Nikolas Wehner, Li Gang, Kuang Li. 2019. Features that Matter: Feature Selection for On-line Stalling Prediction in Encrypted Video Streaming. In IEEE INFOCOM Workshop, Paris.

[8] S. Wassermann, M. Seufert, P. Casas, L. Gang, K. Li. 2019. Let me Decrypt your Beauty: Real-time Prediction of Video Resolution and Bitrate for Encrypted Video Streaming. In TMA'19, Paris.

[9] Qualigon. QoS Testing on Smartphones. https://www.qualigon.de/en/test-systems/qos-testing-smartphones. Accessed: 2020- 10- 14

[10] Guillaume Lemaitre, Fernando Nogueira, Christos K. Aridas. 2017. Imbalanced-learn: A Python Toolbox to Tackle the Curse of Imbalanced Datasets in Machine Learning. In Journal of Machine Learning Research.

[11] Nitesh V. Chawla, Kevin W. Bowyer, Lawrence O. Hall, W. Philip Kegelmeyer. SMOTE: Synthetic Minority Over-Sampling Technique. 2002. In Journal of Artificial Intelligence Research, vol. 16, pages 321–357

[12] Fabian Pedregosa et al. Scikit-learn: Machine Learning in Python. 2011. In Journal of Machine Learning Research (JMLR'11), vol. 12, pages 2825–2830

[13] Scikit-Learn. Scikit-learn: User Guide. https://scikit-learn.org/stable/user_guide.html. Accessed: 2020- 10- 14.