Inferring Video Streaming Quality of Experience at Scale using Incremental Statistics from CDN Logs

Hugo Girault
Broadpeak
France

Loïck Bonniot
Broadpeak
France

Christoph Neumann
Broadpeak
France

firstname.lastname@broadpeak.tv

Abstract

With video streaming applications, end-user Quality of Experience (QoE) is generally estimated using metrics provided by the video players. While QoE monitoring is critical for Content Delivery Networks (CDNs), player metrics are not always provided due to technical, contractual or regulatory limitations. We propose to leverage CDN access logs to infer five common QoE player metrics, with the help of machine learning models. Our approach is based on constant-memory statistic accumulators, allowing large-scale analysis of video streams. We evaluated our implementation with more than 100 000 concurrent streaming sessions on a single CPU core, showing good correlation with QoE ground truth ($\rho > 0.7$, $R^2 > 0.5$).

CCS Concepts

• **Information systems → Multimedia streaming**.

Keywords

Video Streaming, Machine Learning, QoE, CDN

ACM Reference Format:
Hugo Girault, Loïck Bonniot, and Christoph Neumann. 2024. Inferring Video Streaming Quality of Experience at Scale using Incremental Statistics from CDN Logs. In *Mile-High Video Conference (MHV '24), February 11–14, 2024, Denver, CO, USA.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3638036.3640803

1 Introduction

Content service providers aim to deliver the best possible experience when streaming video to their end-users; it is therefore key to measure and estimate the QoE as perceived by end-users. In general, this is achieved by instrumenting the video player on end-users' devices to collect service metrics such as playback startup time, (re)bufferings, and information about the displayed video resolution [17, 24]. These metrics are then reported back to a centralized monitoring and analytics platform. While many player-side SDKs exist that collect these metrics, it is not always possible or desirable to integrate them. In some cases, the SDK may not be supported (e.g., on legacy devices or specific operating systems), the cost of integration and deployment is too important, or integrators are bound to specific third-party SDK providers. Finally, privacy regulations (e.g., GDPR in Europe) might restrict information collected from end-users.

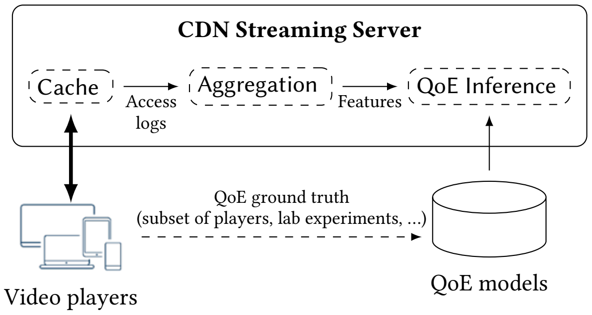


Figure 1: Architecture overview of the proposed QoE inference approach based on CDN access logs.

As an alternative data source for QoE estimation, video streaming CDN providers may want to exploit cache server logs. Web servers, including Nginx [12] and Apache [9], provide access logs with server-side information on the timings, payload size and state of all HTTP requests. In CDNs, these per-HTTP request metric logs are used to detect problems, e.g., by pinpointing long HTTP download times, internal server errors or cache misses. Cache server logs however do not directly indicate user-perceived QoE, especially in the case of video streaming [17]. Indeed, a single Adaptive BitRate (ABR) streaming session generates a large number of HTTP requests, and individual per-HTTP request metrics provide little information about the streaming session QoE as a whole.

In this paper, we describe how a video streaming CDN provider can exploit cache server logs to extract the video streaming QoE indicators at scale. The contributions of this paper are as follows:

- We propose a method to infer the QoE of end-users based on CDN cache server logs. We assume that the CDN logs provide a streaming session identifier for every HTTP request, either from query parameters or from HTTP headers (e.g., with CMCD [1]). This identifier allows the CDN server to perform incremental calculations using constant memory statistic accumulators for every streaming session. The resulting statistic data is fed into machine learning models that output typical QoE metrics such as video startup time, average ABR layer and number of layer switches, video playback stalls count and durations for every streaming session. As a side effect, our models are also able to infer the video player name when it is not known. An overall overview of the approach is depicted in Figure 1.

- We implemented our approach on a Nginx-based CDN cache server. CDN access logs are shared using inter-process communication, and our machine learning models are executed by the highly performant Open Neural Network Exchange (ONNX) [10] runtime. These technical choices, combined with the fact that we only rely on constant-memory online computations on CDN log streams, allow us to handle up to 100 000 concurrent streaming sessions on one CPU core. Such workload is in line with classical CDN cache servers workloads that usually serve hundreds of thousands of concurrent streaming sessions.
- We trained and evaluated our models by running around 10 000 DASH [16] and HLS [19] video streaming sessions in an emulated network with randomized network impairments and several video players (Dash.js [8], HLS.js [6], Shaka [20] and THEOPlayer [25] on Firefox [5] and Chrome [11]). The QoE metrics ground-truth has been recorded using a client-side library. We achieve a good correlation with QoE ground truth ($\rho > 0.7$, $R^2 > 0.5$).

2 Related work

QoE metrics for ABR video streaming have been recognized in a large number of studies [7, 15, 17, 24]. For instance, Nam et al. [17] have evaluated the impact of startup latency, rebuffering and bitrate changes on more than 400 000 YouTube streaming sessions. They have demonstrated in particular that multiple stalls are more detrimental to QoE than a single stall of the same total duration, while startup time and layer switches can also be impactful on QoE.

Early works have focused on QoE estimation from the Internet Service Provider (ISP) perspective, using timing and bandwidth information from packet traces [3, 4, 22]. The proposed methods usually work over encrypted traffic, making them a good choice for the monitoring of third-party streaming services (e.g., Youtube). In this paper, we focus on HTTP layer information extracted from CDN access logs to improve QoE inference accuracy.

In this area, Loh et al. [14] have already proposed a lightweight approach to estimate playback behavior of YouTube videos from the CDN servers. The authors demonstrated that requests inter-arrival times can be valuable in estimating classic QoE metrics. They leveraged packet and request traces with random forests and neural networks to estimate initial startup delay, ABR layer switches and playback stalls. The paper primarily focuses on machine learning. In contrast, our work adopts a more generic approach, encompassing not only machine learning but also real-time request processing and large-scale applications. This comprehensive approach allows us to handle each request in real time using incremental calculations, enabling us to estimate the user-perceived quality of experience immediately after the session concludes.

Similarly, Shah et al. [23] proposed SSQoE, a methodology to model player buffer from inter-arrival timings, allowing ABR layer switches and playback stalls detection for large-scale video performance monitoring. However, their methodology makes several assumptions on player buffering behavior and segment duration that cannot be applied to most CDNs. Our approach is agnostic to the streaming video player and leverages only aggregated statistics extracted from CDN access logs. This strategy drastically lowers the memory and computing requirements, allowing large-scale QoE inference. We believe our work can improve SSQoE's performance and accuracy to assist in large-scale video performance monitoring at the CDN, leading to faster QoE anomaly detection and troubleshooting (see section 5 for a comparison of the two approaches).

3 Proposed approach

Our approach to infer the QoE without the use of any client-side data is illustrated in Figure 1. This solution involves video players sending requests to the streaming server to retrieve the necessary segments and manifests during a session. First, the CDN streaming server, equipped with caching functionality, delivers content to the video player and redirects CDN logs to the aggregation module. Then, the aggregation module performs incremental computations on the logs, significantly reducing data volume and providing metrics that offer a comprehensive overview of the session. Finally, the processed data is utilized as features to machine learning models with the objective of inferring the QoE. We designed these models to be easily modifiable: we have ensured that they can be retrained effortlessly, either by collecting new data from video players or for specific requirements such as the addition of a new streaming configuration to the system. (This includes supporting new operating systems, browsers, video players, ABR algorithms, and also new video encoding and packaging configurations.)

We assume that each CDN access log line includes a streaming session identifier. This assumption is reasonable as many video streaming CDNs include such an identifier either as part of the URL being requested or as an HTTP header. For instance, video players supporting Common Media Client Data (CMCD) [1] may provide the session identifier through the "sid" CMCD key. This session identifier allows the CDN server to perform aggregated calculations across the streaming session; these calculations output arrays of numerical values describing distributions (such as the mean and the median inter-request time for each session) and counts of sudden changes in distributions or values.

At the end of a streaming session, our system infers the QoE metrics related to the streaming session: video startup time, average ABR layer and number of layer switches, video playback stalls count and duration. We rely on machine learning models to infer these metrics (more precisely, for every metric we rely on a dedicated machine learning model). These models take as input the array of calculated numerical values describing the streaming session as extracted from the CDN access logs.

To build the machine learning models, we rely on ground truth data collected via a player-side enterprise SDK that retrieves the QoE metrics that we would like to estimate. The ground truth data is either collected via controlled lab experiments where a large variety of network conditions are emulated or by relying on a set of video players deployed in the wild. For instance, ground truth can be collected by relying on a subset of recruited end-users, by deploying dedicated video player probes at selected vantage points or by exploiting data coming from other deployments with a client-side SDK.

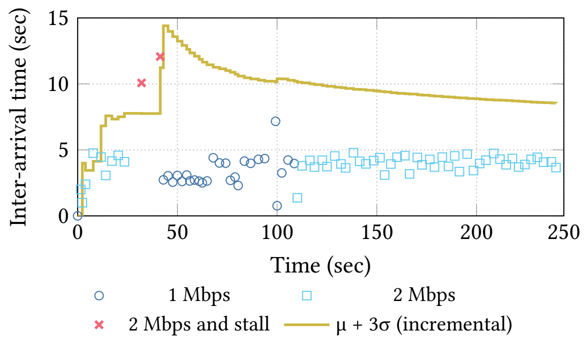


Figure 2: Shown layer over time vs. the incremental Three Sigma Rule [21] for a HLS.js session. This rule allows us to identify a stall at the beginning of the session.

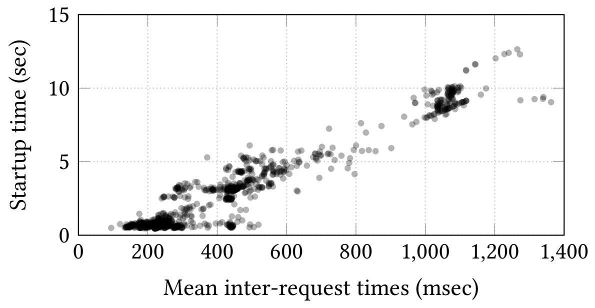


Figure 3: Startup time vs. mean inter-request time for thousands of Dash.js sessions (Pearson's correlation coefficient: 0.95; Spearman's rank correlation: 0.85).

3.1 Intuition and motivating example

In this subsection we provide a few simple examples to show that CDN cache server access logs provide a rich set of information and metrics that should allow us to infer QoE metrics.

A CDN cache server access log is a concatenation of log lines, each log line providing metrics related to a single HTTP request. A log line includes information such as timings (e.g., time of request and duration of request), payload size and HTTP response status. Other metrics, such as an indication of a cache miss or hit, a session identifier or specific headers might also be included in a log line. Taken individually, each log line provides little insight on the quality of an ongoing streaming session. However, if analyzed along a streaming session, the logged metrics reveal the state and behavior of the video player. As an example, a video player without any play-out problem generally fetches video segments of a single representation at regular timing intervals; in contrast, a player that encounters bandwidth variations or problems during download will not be able to fetch the segments at the same frequency and might switch to different representations. These patterns become visible in the access log if we analyze the log lines along a streaming session: inter-request times and payload sizes change over time.

Figure 2 shows an example of the behavior of inter-request times of a streaming session in the case of a stall. We can clearly observe an increase of inter-request times. The two outliers located above three standard deviations relative to the mean correspond to player stalls. Following the stall, the player decides to request smaller segments as it switches to a smaller representation of the video. Similarly, Figure 3 indicates a linear correlation ($\rho = 0.95$) between the mean time between the ten first queries of a session generated with a Dash.js video player and its startup time as measured by a player-side SDK. These observations indicate that we may use statistical methods to estimate the QoE metrics. (We recall that previous work [14, 23] also showed that inter-request times can be useful to estimate QoE.)

3.2 Incremental computations on CDN logs

CDN access logs should be processed in real time to react quickly to any QoE degradation issue. For performance reasons, it is unrealistic to store every access log in memory for the duration of every streaming session: some sessions may last several hours, resulting in thousands of log rows per session.

In this work we propose to perform *incremental* calculations using constant-memory statistic accumulators for every streaming session. When parsing an access log line, the QoE service extracts the streaming session identifier, the type of request (e.g., manifest or segment) and even the selected ABR layer from the provided URL. It uses the extracted identifier to aggregate metrics related to a given streaming session on-the-fly. For instance, it is possible to estimate the distribution of any metric using basic arithmetic operations (minimum, average, maximum) and approximate quantile sketches (such as the P^2 quantile estimator [13] or HdrHistogram [26]). We thereby compute approximate distributions of inter-request times, payload sizes and Time To Last Byte (ttlb) for both manifests and segments, and for cache hits and cache misses.

Additionally, it is also relevant to add several counters tailored for ABR streaming. For instance, we use the selected layer provided in access log to count changes in requested layers (note that a change in requested layer may not always lead to a playback layer switch, this counter should be refined with other statistics, see subsection 5.3). As shown in Figure 2, we also use the estimated distribution of segment inter-request times to count the number of segments with an inter-request time above the three sigma threshold. With this approach, the memory footprint of any concurrent streaming session is constant and limited, while distribution statistics are pre-computed and always available for QoE inference.

4 Implementation details

We implemented a prototype of our approach as a dedicated C++20 QoE service running alongside Nginx [12] cache server. For every streaming session served by Nginx, this QoE service provides access log parsing and aggregation, along with final QoE inference.

4.1 Scalable access log aggregation

Nginx is usually configured to write access logs to the server filesystem. While storing the logs can be useful for audits and analytics, we setup Nginx to send access logs to the QoE service via UDP sockets instead. This way, access log processing is simplified and there is no specific requirement for storage I/O. Moreover, multiple Nginx servers can feed the QoE service through the network.

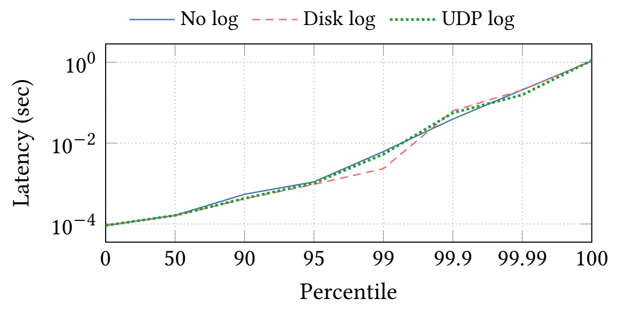


Figure 4: Distribution of HTTP query latencies for different logging configuration with Nginx (log scale, 60 000 concurrent streaming sessions). The logging configuration does not impact server latency.

Our benchmarks show that enabling access logs have no visible impact on Nginx performance in the case of video streaming (either when writing logs to a rotating disk, or when sending logs via UDP, see Figure 4).

We leverage the Boost Accumulators library [18] to store the statistics for every streaming session in a fixed-size structure, resulting in a fixed memory consumption of 30 kB by concurrent streaming session. To ensure good scalability, we decouple access log parsing from access log aggregation in multiple threads, and use lock-free queues to transfer data between threads. This decoupling is critical, as access log reception and parsing accounts for most of the processing time under load.

4.2 QoE inference

We leverage the ONNX format and runtime to execute QoE models within our prototype. The ONNX format allows easy interoperability between different machine-learning frameworks: this means that we can train QoE models using many available frameworks and execute the inference using many different runtime applications. In our case, we use the official C++ ONNX runtime. This approach decouples the QoE models' internals from the remainder of the prototype. It is therefore easy to change the QoE models without needing any change to the source code, allowing fast deployments of re-trained and specialized QoE models.

To maximize throughput, the prototype periodically runs QoE inference on the batch of streaming sessions that have been stopped. More specifically, the inference is done on sessions with no recent access log data, taking pre-computed Boost Accumulators' statistics for these sessions, and using these statistics as ONNX inputs. Inferred QoE results are then shared with external third-party analytics and can be used for global monitoring use-cases.

5 Evaluation

In this section we describe the methodology we followed to evaluate our approach. First, we describe how we collected access logs for diverse streaming sessions and the ground-truth QoE metrics associated with these sessions. Then, we give more details on the QoE models we trained on our collected dataset of streaming sessions. We finally give some accuracy figures and compare our results to work from Shah et al. [23].

Table 1: Distribution of the QoE metrics in the dataset described in subsection 5.1.

	Min	Mean	P50	P90	Max
Duration (sec)	9.0	300	310	540	600
Startup time (sec)	0.42	6.0	1.5	16	150
Average layer (kbps)	340	3700	3600	5100	12000
Layer switches (count)	0.0	6.4	4.0	12	120
Stalls (count)	0.0	5.2	1.0	11	260
Stalls duration (sec)	0.0	27	3.4	56	580

5.1 Dataset description

In many production deployments, CDN access logs are accompanied by ground-truth QoE metrics from client-side analytics solutions. While very realistic, production datasets are in fact often challenging to use for QoE inference training. In particular, most real streaming sessions do not suffer from any particular QoE degradation. This is especially true when the connectivity between customers and CDNs is good (e.g., with wired or fiber connectivity). In our experience, production datasets are thereby heavily unbalanced towards "good" QoE, and are biased towards specific network topologies.

To avoid these shortcomings, we used a dedicated testbed to emulate video streaming sessions with different parameters and diverse network conditions. On the server-side, we used Nginx as a cache server, along with our QoE inference implementation (section 4). On the client-side, our testbed ran real video players on real web browsers: the video players used for this experiment are Dash.js [8], HLS.js [6], Shaka [20] and THEOPlayer [25] on Firefox [5] and Chrome [11]. To emulate diverse network conditions, we introduced random latency, jitter and bandwidth shaping between the clients and the servers. The idea is that if the network conditions degrade too much, the video players can no longer retrieve segments in time, resulting in layer change down or even video playback stalls. With this approach, we have emulated around 10 000 DASH [16] and HLS [19] streaming sessions, with both live and on-demand content, of playback duration varying between a few seconds and tens of minutes.

This dataset is much more diverse and balanced than typical production datasets, making it more relevant for QoE inference. We give some QoE distribution insights in Table 1. For instance, we recorded sessions with startup delay ranging from 400 ms to 3 minutes, with the number of layer switches ranging from 0 to 120. More than half of the recorded sessions suffer from more than 1 second of playback stall, while 10% of the sessions suffer from more than 56 seconds of playback stall, something that can be considered as very bad QoE. Our take is that if we obtain good QoE inference models on this very diverse dataset, we could easily adapt these models to any (less diverse) production environment. We illustrate this claim in subsection 5.4.

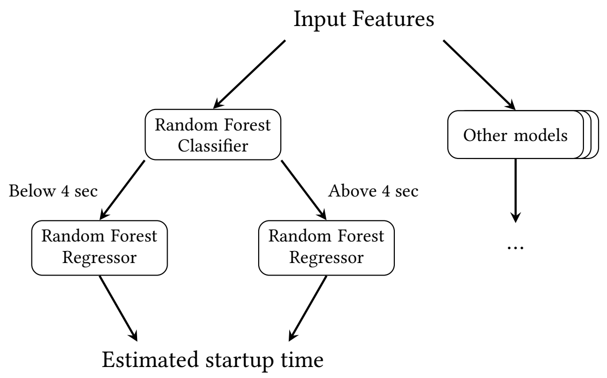


Figure 5: Extract of the QoE inference graph with details for startup time estimation. Input features are first used to discriminate between short and long startup time. Two specialized regressors are used to estimate the final startup time.

5.2 QoE model training

We adopt a unique approach for each metric that we aim to infer, generating a distinct machine learning model tailored to its specific characteristics and requirements. This section particularly focuses on the methods employed to estimate the startup time and identify the player. However, the methodologies and techniques discussed herein can be generalized and applied to estimate other metrics as well. We dedicate 80% of it towards the training of machine learning models. The remaining 20% of the dataset is employed for model validation and comparison (subsection 5.3). To estimate the QoE metrics, we conducted evaluations on various machine learning models. Our findings revealed that Random Forests [2], offered the most optimal balance between model accuracy and complexity.

As depicted in Figure 5, we employ a two-step approach to estimate the startup time. Initially, we utilize a Random Forest Classifier, which takes in 49 features, to determine if the startup time surpasses a predefined threshold of 4 seconds. Depending on the prediction from the classifier, we then employ one of two distinct Random Forest Regressors, each also utilizing the same 49 features. If the classifier predicts that the startup time is below the threshold, we use a regressor that has been specifically trained on sessions with startup times of less than 4 seconds. On the other hand, if the classifier predicts that the startup time exceeds the threshold, we use a different regressor that has been trained on the remaining sessions. This approach allows us to tailor our models to different ranges of startup times, thereby improving the overall accuracy of our estimations. The features that contribute most significantly to the performance of our models are those associated with the time intervals between the initial 10 requests of each session. Specifically, we found the average time, the 50th percentile, the 90th percentile, and the 99th percentile to be of particular importance. These features offer an in-depth insight into the distribution of request times during the initial phase of each session, thereby facilitating a more precise computation of the startup time.

As another example, we now describe our strategy to identify the video player used in a streaming session, along with its configuration (operating system, ABR algorithm, ...). We propose a straightforward player identification strategy based on a Random Forest Classifier. In our analysis, we found that no single statistical feature significantly dominates the decision-making process of the classifier. Our models consider a variety of features, including the average time between the first 10 requests of a session, the number of consecutive manifest requests, and the count of segments issued beyond the mean plus three times the standard deviation. While these features do not individually stand out, their collective influence contributes to the classifier's ability to accurately identify the video player configuration.

Table 2: Comparison of Mean Absolute Error (MAE) and Coefficient of Determination (R^2) for several QoE metrics (test dataset of 1000+ DASH and HLS streaming sessions).

	Baseline [23]		Our approach	
QoE metric	MAE	R^2	MAE	R^2
Startup time (sec)	N/A		**0.94**	**0.89**
Average layer (kbps)	337	0.83	**210**	**0.89**
Layer switches (count)	4.5	-0.02	**1.7**	**0.90**
Stalls (count)	4.2	0.23	**1.51**	**0.51**
Stalls duration (sec)	25	-0.05	**8.3**	**0.72**

5.3 Results

To compare our proposed approach with the state of the art, we implemented an alternative QoE inference method from Shah et al. [23]. This alternative also leverages CDN access logs for video streaming QoE inference, and provides similar QoE metrics except for startup time estimation. We show inference accuracy results for both approaches in Table 2 (the depicted numbers are all based on the evaluation part of our dataset described in subsection 5.1). On all tested QoE metrics, our approach has a better accuracy than the baseline, and provides sufficient accuracy for production usage.

Startup time: it is particularly important to estimate video playback startup delay, as it can severely reduce QoE if it becomes too large. Because our approach computes statistics on the first inter-request times, it provides excellent startup time estimation, with less than 1 second of Mean Absolute Error (MAE) and less than 100 ms of Median Absolute Error (MedAE). Unfortunately, the baseline provides no startup time estimation.

ABR layers: in many configurations, layer information can be extracted from the requested segments' URLs. Intuitively, inferring the average ABR layer for a given session shall be an easy task: one only has to parse access logs timings and URLs to retrieve requested layers history. In practice however, it highly depends on the video player ABR strategy with respect to layer changes. For instance, in its default configuration, Dash.js attempts to retrieve higher layers periodically, and evaluates whether or not it should upgrade the layer shown on screen. We show an example of this behavior in Figure 6. Our machine learning models are trained to understand different video player strategies, leading to more accurate average layer inference (40% MAE improvement over the baseline) and layer switches inference (60% improvement).

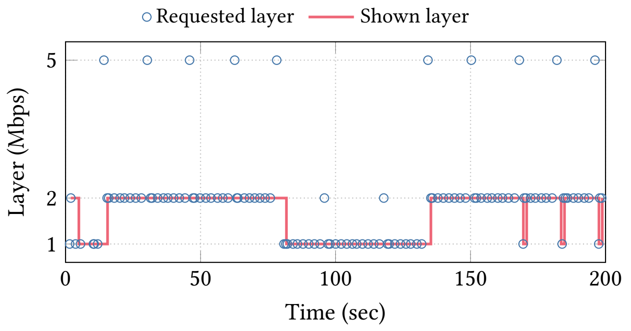
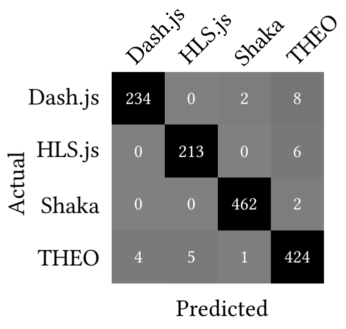


Figure 6: Requested layers vs. shown layer over time for a Dash.js session with default ABR algorithm. The player does not necessarily change the shown layer when it tries to download higher layers.


Dash.js
HLS.js
Shaka
THEO
Actual
Predicted

Actual \ Predicted	Dash.js	HLS.js	Shaka	THEO
Dash.js	234	0	2	8
HLS.js	0	213	0	6
Shaka	0	0	462	2
THEO	4	5	1	424

Figure 7: Confusion matrix showing actual vs. inferred video player in our test bed. Video players are correctly identified in 98% of sessions.

Stalls: when a video player is unable to download content segments in time, it may stall video playback. Depending on the frequency and duration of stalls, the perceived QoE may be drastically reduced. While our machine-learning approach can infer stall occurrence with 85% accuracy, results highlight that estimating the number of stalls and their total duration is relatively hard: our models have an MAE of 1.5 stalls and 8.3 stalled seconds, with an MedAE of 1 stall and 4 stalled seconds. This is a substantial improvement over the baseline results (MAE of 4.2 stalls, 25 stalled seconds) and should be sufficient for QoE inference in most cases.

Player identification: we show a confusion matrix for player identification in Figure 7, based solely on CDN access logs. With an accuracy of 98%, it is clear that HTTP request patterns are sufficiently different between players to allow re-identification with a properly trained machine-learning model. This re-identification can be useful for analytics purposes, e.g. when video players cannot be identified from user-agent information. For simplicity, in this evaluation we configured the video players with their default ABR algorithm. If different algorithms are used, it is possible to train the player identification model with other (sub)classes, or generalize the existing classes with more diverse data. For instance, "Dash.js" output may refer to Dash.js player in Chrome or Firefox, with ABR strategy set to either "abrDynamic" or "abrThroughput".

5.4 Performance in the wild

We evaluated the models trained on our lab dataset on several streaming sessions running over the Internet (including sessions running over unreliable Wi-Fi). Results on this test dataset were very similar to those presented in Table 2, with slightly higher MAE: 1.7 s for startup time, 220 kbps for average ABR layer, 1.8 for layer switches count, 1.6 for stalls count and 9.7 s for stalls duration. This shows that our methodology produces generalizable models, which can be used in the field without re-training. It is naturally also possible to fine-tune the models with additional data coming from production environments if available, to further improve QoE inference accuracy.

6 Conclusion

In this paper we presented a scalable and generic methodology to estimate ABR video streaming QoE from CDN access log, with the help of statistical accumulators and machine learning models. We implemented this methodology using a Nginx cache server, built a dataset of video streaming session with diverse network and QoE conditions, and compared our approach to the state of the art. Our results over both synthetic and production data demonstrate improved accuracy and generalizability.

Several avenues for future work in server-side QoE estimation remain open. We believe that more complex machine learning models could be used to improve the QoE inference accuracy. Deep learning models are good candidates for the task, including sequential models such as transformers [27] that could process (and aggregate) access logs query by query. Additional statistical features could also be used to improve the models accuracy; for instance some players may share relevant CMCD [1] metrics such as object duration (d), playback rate (pr), or even buffer length information (bl, bs). Different types of streaming sessions could also be studied, using additional video players, different ABR layer selection algorithms, low-latency streaming, and sessions including pauses or user-induced rebufferings.

References

[1] Consumer Technology Association. 2020. *Web Application Video Ecosystem - Common Media Client Data CTA-5004.* https://cdn.cta.tech/cta/media/media/resources/standards/pdfs/cta-5004-final.pdf
[2] Leo Breiman. 2001. Random forests. *Machine learning* 45 (2001), 5–32. https://doi.org/10.1023/A:1010933404324
[3] Francesco Bronzino, Paul Schmitt, Sara Ayoubi, Guilherme Martins, Renata Teixeira, and Nick Feamster. 2019. Inferring Streaming Video Quality from Encrypted Traffic. *Proceedings of the ACM on Measurement and Analysis of Computing Systems (POMACS)* 3, 3 (2019). https://doi.org/10.1145/3366704
[4] Pedro Casas, Alessandro D'Alconzo, Pierdomenico Fiadino, Arian Bär, Alessandro Finamore, and Tanja Zseby. 2014. When YouTube does not work—Analysis of QoE-relevant degradation in Google CDN traffic. *IEEE Transactions on Network and Service Management* 11, 4 (2014), 441–457.
[5] Mozilla Corporation. 2023. *Firefox.* https://www.mozilla.org/en-US/firefox/
[6] Video Dev. 2023. *HLS.js source code.* https://github.com/video-dev/hls.js
[7] Florin Dobrian, Vyas Sekar, Asad Awan, Ion Stoica, Dilip Joseph, Aditya Ganjam, Jibin Zhan, and Hui Zhang. 2011. Understanding the impact of video quality on user engagement. *ACM SIGCOMM CCR* 41, 4 (2011), 362–373.
[8] Dash Industry Forum. 2023. *Dash.js source code.* https://github.com/Dash-Industry-Forum/dash.js

[9] The Apache Software Foundation. 2023. *Log files.* https://httpd.apache.org/docs/2.4/en/logs.html

[10] The Linux Foundation. 2023. *Open Neural Network Exchange.* https://onnx.ai/

[11] Google. 2023. *Google Chrome.* https://www.google.com/intl/en/chrome/

[12] NGINX Inc. 2023. *Module ngx_http_log_module.* http://nginx.org/en/docs/http/ngx_http_log_module.html

[13] Raj Jain and Imrich Chlamtac. 1985. The P 2 algorithm for dynamic calculation of quantiles and histograms without storing observations. *CACM* 28 (10 1985), 1076–1085. https://doi.org/10.1145/4372.4378

[14] Frank Loh, Fabian Poignée, Florian Wamser, Ferdinand Leidinger, and Tobias Hoßfeld. 2021. Uplink vs. Downlink: Machine learning-based quality prediction for http adaptive video streaming. *Sensors* 21, 12 (2021). https://doi.org/10.3390/s21124172

[15] Ricky KP Mok, Edmond WW Chan, and Rocky KC Chang. 2011. Measuring the quality of experience of HTTP video streaming. In *IM*. IEEE, 485–492.

[16] MPEG-DASH. 2023. *ISO/IEC 23009 - Dynamic Adaptive Streaming over HTTP.* https://www.mpeg.org/standards/MPEG-DASH/

[17] Hyunwoo Nam, Kyung Hwa Kim, and Henning Schulzrinne. 2016. QoE matters more than QoS: Why people stop watching cat videos. In *INFOCOM*. https://doi.org/10.1109/INFOCOM.2016.7524426 ISSN: 0743166X.

[18] Eric Niebler. 2023. *Boost.accumulators.* https://www.boost.org/doc/libs/1_63_0/doc/html/accumulators.html

[19] R Pantos. 2023. *RFC 8216 - HTTP Live Streaming.* https://datatracker.ietf.org/doc/html/rfc8216

[20] Shaka Project. 2023. *Shaka Player source code.* https://github.com/shaka-project/shaka-player

[21] Friedrich Pukelsheim. 1994. The Three Sigma Rule. *The American Statistician* 48, 2 (1994), 88–91. https://doi.org/10.1080/00031305.1994.10476030

[22] Raimund Schatz, Tobias Hoßfeld, and Pedro Casas. 2012. Passive YouTube QoE monitoring for ISPs. *Proceedings - 6th International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing, IMIS 2012* (2012). https://doi.org/10.1109/IMIS.2012.12

[23] Anant Shah, Juan Bran, Kyriakos Zarifis, and Harkeerat Bedi. 2022. SSQoE: Measuring Video QoE from the Server-side at a Global Multi-tenant CDN. In *PAM*. 600–625. https://doi.org/10.1007/978-3-030-98785-5

[24] Shoko Takahashi, Kazuhisa Yamagishi, Pierre Lebreton, and Jun Okamoto. 2019. Impact of Quality Factors on Users' Viewing Behaviors in Adaptive Bitrate Streaming Services. In *QoMEX*.

[25] THEO Technologies. 2023. *THEOPlayer.* https://www.theoplayer.com/

[26] Gil Tene. 2023. *A High Dynamic Range (HDR) Histogram.* https://hdrhistogram.github.io/HdrHistogram/

[27] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *NeurIPS* 30 (2017).