SSQoE: Measuring Video QoE from the Server-Side at a Global Multi-tenant CDN

Anant Shah(✉), Juan Bran, Kyriakos Zarifis, and Harkeerat Bedi

CDN Engineering, Edgecast, 13031 W Jefferson Blvd., Los Angeles, CA 90094, USA
{anant.shah,juan.bran,kyriakos.zarifis,harkeerat.bedi}@edgecast.com

Abstract. Over the past decade, video streaming on the Internet has become the primary source of our media consumption. Billions of users stream online video on multiple devices with an increasing expectation that video will be delivered at high quality without any rebuffering or other events that affect their Quality of Experience (QoE). Video streaming platforms leverage Content Delivery Networks (CDNs) to achieve this at scale. However, there is a gap in how the quality of video streams is monitored. Current solutions rely on client-side beacons that are issued actively by video players. While such approaches may be feasible for streaming platforms that deploy their own CDN, they are less applicable for third-party CDNs with multiple tenants and diverse video players.

In this paper, we present a characterization of video workload from a global multi-tenant CDN and develop SSQoE: a methodology deployed on the server side which estimates rebuffering experienced by video clients using passive measurements. Using this approach, we calculate a QoE score which represents the health of a video stream across multiple consumers. We present our findings using this QoE score for various scenarios and compare it to traditional server and network monitoring metrics. We also demonstrate the QoE score's efficacy during large streaming events such as the 2020 Superbowl LIV. We show that this server-side QoE estimation methodology is able to track video performance at an AS or user agent level and can easily pinpoint regional issues at the CDN, making it an attractive solution to be explored by researchers and other CDNs.

1 Introduction

Streaming video constitutes a large portion of traffic on the Internet. Content Delivery Networks (CDNs) deliver tens of Terabits per second of video for large video streaming platforms that users rely on for news, entertainment, and communication. Live streaming services have further gained popularity with the rise of over-the-top (OTT) services.

The increasing volume of video traffic and the user expectations for high quality necessitate visibility into client-perceived performance of video streaming. A key performance metric for video Quality of Experience (QoE) is the

O. Hohlfeld et al. (Eds.): PAM 2022, LNCS 13210, pp. 600–625, 2022.
https://doi.org/10.1007/978-3-030-98785-5_27

rebuffering experienced by video players. Rebuffering can be caused by a variety of reasons including problems at the client, at the ISP, or at the CDN layer. Content providers that employ their own video players usually extend the players to generate reports about performance metrics directly from the client. Alternatively, third-party services can be utilized to report such data from a set of clients. However, in many cases where commercial third-party CDNs are used for video delivery, the CDN operators may not have visibility into such client reports. Therefore, CDN operators typically use more indirect server and network performance metrics to identify performance degradation of video streams that they deliver. Such metrics may not provide a clear picture of client-perceived performance, and make it harder to evaluate if a problem is impacting clients enough to warrant traffic engineering actions like choosing alternative peering links or sending a client request to a different location.

To address the gap in visibility and the challenges of client-side beacon based monitoring, we characterize the video workload at a multi-tenant CDN that spans over 160 Points of Presence (PoPs) distributed globally, making it one of the largest egress networks in the world [10], and we design and deploy a server-side QoE monitoring tool called SSQoE on the CDN in order to estimate client video QoE based on server logs. In particular, our work makes the following contributions:

- We identify and characterize QoE metrics that can be tracked using only server-side logs and analyze their implications on video performance. These include the timestamp when a video request is received at the CDN, gaps in requests, changes in bitrate, and time taken to serve a request.
- We propose a simple, scalable, and intuitive methodology called SSQoE, which uses these metrics that are available from CDN access logs to estimate rebuffering on the client side. We use other relevant QoE indicators to calculate a QoE score that can be used to track video performance agnostic to the type of video player, device, and type of video traffic.
- We show the value of our methodology by comparing it with commonly used client beacon based reporting. We demonstrate use cases from our deployment, like tracking regional per-PoP anomalies from the 2020 Superbowl event where the CDN served the live video stream to millions of clients.
- We demonstrate the shortcomings of the server and network based monitoring metrics by comparing their efficiency in representing client-perceived performance to our methodology during incidents like transit provider connectivity failures or cache fill errors.

This paper is structured as follows. We describe the current video distribution pipelines commonly used by video providers and motivate our work by elaborating on the challenges faced by multi-tenant CDNs to monitor video performance in Sect. 2. In Sect. 3, we present insights on the video workload delivered by the CDN, and we describe relevant performance indicators. In Sect. 4 we present our QoE score calculation methodology. We validate our methodology using a testbed and production traffic in Sect. 5 and show its value using several examples in Sect. 6. We discuss the limitations, related work, and conclude in Sects. 7, 8, 9 respectively.

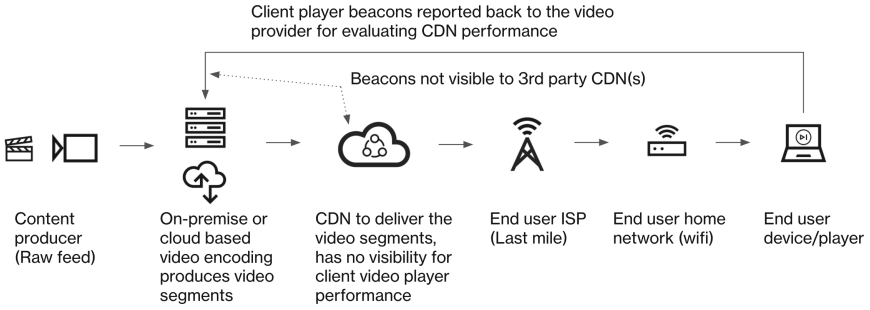


Fig. 1. Live video streaming pipeline. Each step can be a cause of increased latency. Client player reports performance metrics back to content provider or a 3rd party beaconing service using their SDKs. These reports are analyzed to evaluate CDN performance from client's perspective.

2 Background

Video streaming workflows are complex, especially while delivering a live stream. As with any complex system, each involved component can fail and lead to degradation in end-to-end performance. In the case of video streaming, such degradations translate to reduced QoE for the video consumer. In this section, we describe various pieces of a video pipeline starting from data capture to distribution, point out gaps in visibility, and motivate the need for server-side video monitoring.

To deliver scale and achieve robustness, many video streaming providers use multiple CDNs to deliver content to the clients. Third-party beaconing platforms [2] have become a popular mechanism by which content providers monitor CDN performance to get insights on client experience and use beacon data to steer traffic across multiple CDNs.

Components in Video Streaming

Figure 1 illustrates the components of a live video streaming pipeline. An on-premise camera captures the video and the raw feed is sent to the video encoder. Video encoders can be on-premise installations or cloud based services. The video encoder gathers a sufficient video chunk (usually less than 2 to 5 s) before it can generate an encoded video segment that will be distributed. Next, the system generates a manifest file, which describes the set of segments that a video player will need to request for the given video stream. The video encoder encodes the segments into multiple bitrates denoting different quality levels. Each unique time range of video may exist as multiple segments in different bitrates. The video encoder then pushes the segments to a CDN origin server or the CDN can pull them periodically. The client starts the video streaming on their device and requests the video asset. Each client is identified as a unique *session*. First, the manifest is delivered to the client. Based on current network conditions, the player chooses an appropriate bitrate from the manifest and

requests the corresponding video segment. These segments, which are available from the CDN origin server, are cached at the CDN edge when a client requests them. The client request traverses through the client ISP link to its backbone and then to the CDN peering point where it finally gets served by the CDN.

Each component in the pipeline adds to the latency and is a potential source of video performance degradation. It is possible that the video ingest at the video encoder is delayed. In such cases most clients will start lagging in their live stream. Next, there can be delay when ingesting the video segments from the encoder to the origin server or at the CDN cache due to degradation in backbone or transit provider connectivity. Delays can also occur at the CDN PoP due to overloaded servers or congestion at the peering link. Congestion is also common at the client ISP, in the middle or the last mile. Finally, the load on the client device, e.g. available RAM, CPU, can also play a role in how the video player performs. Given all these potential sources of delay and performance degradation, the process of pinpointing an exact the root cause for QoE impairments becomes challenging.

QoE Metrics in the Wild

Existing video performance monitoring techniques have been focused on analyzing client-side metrics. The video player instruments a beacon that periodically reports how the player is performing. Those beacons are collected and analyzed to extract QoE insights. Some video providers own the application or video player and therefore can implement their own data collection strategy, which the CDN may not have the visibility into. There are also third-party vendors such as [2] that are commonly used for such beaconing. Key metrics that are monitored by the video providers are:

Startup-delays: This measures delays experienced when starting a stream.

Rebuffering ratio: This is the most commonly monitored metric [2,7,18,21]. It represents the ratio of the amount of time a video player was stalled waiting for new segments to be downloaded over the total video duration. For example, if a client played a 60-second video but in the middle of the playback it ran out of buffered segments and it had to wait for 2 s before resuming, then the rebuffering ratio is $2/60 \doteq 3.3\%$.

Bitrate: This denotes the number of bits used to encode a given segment of video. Higher bitrate represents better video quality. Video providers who use the CDN service to distribute the content expect the bitrate to be high.

Video playback failure: This represents cases where video player had trouble playing the content it received. This can be a result of expired token, digital rights management (DRM) issues, etc. which are used to secure the video segment so that only approved clients, such as paying subscribers, can view the content.

Motivation for Server-side Video QoE Monitoring

Commercial CDNs deliver a mixed workload of video traffic for many live streams. Each video provider (CDN customer) can have a completely different set of configurations for encoding and caching the video segments. Their performance goals could also be different, e.g. some might value lower latency

over higher bitrate/quality. Furthermore, popular live streams can have millions of concurrent users requesting the same video asset from the CDN.

Client-side metrics captured via beacons provide a clear view per-session of how the client experiences the video. However, these methods rely on some control over the client. They are by design made for the video provider to consume and not for the CDN provider. Content providers that use their own CDN and video client player (i.e. both ends of the connection) can easily implement this beaconing and use it for CDN performance monitoring. In contrast, commercial CDN operators face the challenge of analyzing the performance of video streams without complete visibility. The root cause of a performance issue can very well be outside of the CDN stack, such as at the encoder or at the client ISP. This large disparity in type of workloads for different CDN customers and type of content, dependency on video player metrics, and lack of complete visibility makes the previously studied client-side approaches less viable for a commercial CDN.

3 Characterizing the CDN Video Workload

In this section we analyze a large CDN video workload and extract insights that serve as guiding principles for designing a server-side QoE monitoring strategy. For this analysis, we use 24 hours (one weekday) of CDN access logs for a large live video streaming provider that powers multiple live sports, news, and entertainment services. The dataset spans more than 10 million HTTP requests.

Unique Session Tracking
In order to extract aggregate performance information for a stream that is delivered through the CDN, we first need to understand performance of each session separately. For that purpose, there needs to be a unique identifier that characterizes a particular client stream. Some video streaming providers maintain a unique session ID per client session, which can make session tracking easier. However, SSQoE does not rely on such a session ID. In particular, we notice that in the majority of cases a unique client ID can be inferred by using the client IP and the device User Agent. In our dataset, 99% of the session IDs map to a unique {client_ip, user_agent} tuple. In the reverse direction, the same {client_ip, user_agent} tuple maps to the same session ID for about 80% of the cases. This may be due to the presence of large carrier-grade NATs [24] which can lead to aggregation of multiple clients into one session. However, based on our analysis of the CDN access logs, this noise is minimal and it is still possible to infer per-session characteristics using the hash of client IP and User Agent tuple.
Takeaway: *A session or client ID is important for measuring per-session QoE from the server side. In the absence of such an identifier in the CDN logs, a hash of the client IP and User Agent is an adequate alternative.*

Total Duration of Video Sessions
Video streams delivered by multi CDNs can vary significantly in terms of duration. Additionally, the complexity of user viewing patterns is high, especially for

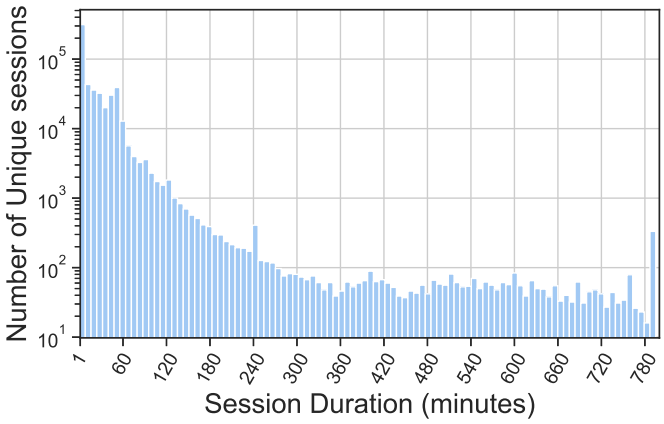


Fig. 2. Session durations. Most sessions are short lived.

live streams where clients can join or leave a stream at anytime. This can add noise when trying to measuring the QoE at scale for millions of users. In Fig. 2 we show the number of sessions ordered by playback duration over the course of one day. As shown in the figure, most sessions are short-lived (note the log scale on the Y-axis). This highlights the variety and scale of our dataset, but also reveals a key observation that needs to be taken into consideration while designing server-side QoE measurements: QoE decisions need to be made near-real time over a short time duration, because most sessions are short lived (in the order of minutes).

Takeaway: *Due to the short playback duration and high churn of clients in live streams, tracking the performance of a few sessions over a long period of time will provide little value; instead, tracking a large number of session over a shorter time window is more feasible.*

Time Taken to Serve a Request

Next we try to understand which metrics can provide insights on the performance of a session. A metric that impacts video delivery is the amount of time it takes the server to deliver a response. For a given video stream, the time it takes to deliver the same video segment remains fairly constant, and fluctuations in the flow completion time are a good indicator of performance change.

Tracking the time taken to serve a request when grouping the streams by different dimensions or characteristics also reveals insights about misbehaving sections of the traffic. For example, in Fig. 3 we show the distribution of time taken to serve a request grouped by top User Agents of a video provider. For this distribution we look at requests for video segments betweeen 8-10MB to keep the comparison fair across all requests and user agents. The User Agent names are anonymized. Here, we observed the CDN taking several seconds longer to serve the request to one particular User Agent (UA-1). This User Agent belonged to the application of the video provider on an older generation of a large Smart TV

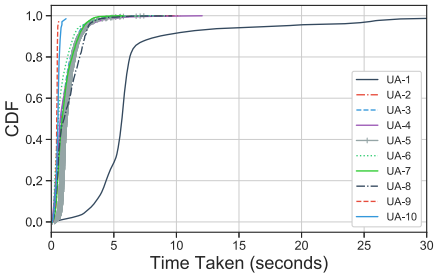


Fig. 3. CDF of time taken to serve a request. User Agents show different performance profiles for requests of similar-sized video segments.

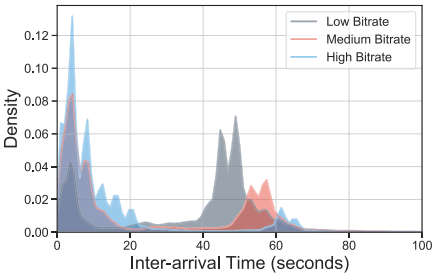


Fig. 4. KDE interarrival times for sessions. During normal operation, the inter-arrival time between requests is short.

manufacturer. When we compared the performance of this User Agent in the client-side QoE monitoring systems, it also accounted for top rebufferring events for that provider on the CDN.

The time taken to download a video segment is not only a function of client player behavior but also of server performance (high CPU utilization times can lead to higher response time), network congestion, and size of the video segment. When the bitrate of a video stream changes, the size of the video segment also changes. For example, ads can be inserted into a video stream, and tracking this change is an important indicator of the behavior of the video stream since the ads can be encoded in different bitrate.

Takeaway: *Understanding the time taken to serve subsets of requests of the same stream that share similar characteristics can reveal anomalous behavior in different dimensions, which is important for server-side QoE monitoring.*

Request Arrival Times and Bitrate Changes

The behavior of the video player at the client plays an important role in how the user experiences the video stream. Most common players employ Adaptive Bit Rate (ABR) [1]. During playback, players try to estimate the achievable bandwidth, i.e. which corresponding bitrate/quality can be achieved using the current network conditions. When the available bandwidth drops, the ABR algorithm drops the bitrate i.e., requests a subsequent video segment of a lower quality.

It is possible to understand how the client-side player perceives the connection by tracking when the request arrived and what bitrate the client requested. We capture the request timing information in comparison to previous requests on the same session by tracking the inter-arrival time of requests. In Fig. 4 we show the Kernel Density Estimation (KDE) distribution of requests for a live streaming provider over one day. We bucket each request into three bitrate types: Low, Medium, and High. In most cases the request inter-arrival times are in the order of few seconds, indicating normal player behavior where the player maintains sufficient video segments in the buffer to play next. However, we also notice that in some cases, and more commonly when the used bitrate quality is low, the

inter-arrival times can be high (greater than 30 s). To evaluate if the high inter-arrival times correlate with change in bitrate, for each case where inter-arrival times are higher than 12 s, we check if the current bitrate matches that of the previous request of the same session. We chose the threshold of 12 s since the particular provider examined here starts playback as soon as the player receives a minimum of 12 s of video. We categorize the bitrate change into Same, Up, or Down based on the direction of the quality change. We see that in more than 70% of the cases where inter-arrival times are high, the subsequent request is for a lower quality segment (direction: Down). This shows that the player suffered some delays and ABR reduced the quality of the video. ABR only switches quality to higher if it is sure the client will not see a negative impact. In about 20% of the cases the direction of change in quality was Up, meaning even though the inter-arrival times increased, the client may not have experienced a rebuffering event.

Takeaway: *The change in bitrate is a good metric to capture client player behavior and to use as indication for client-perceived quality, but it is not enough by itself to accurately estimate if the client experienced rebuffering.*

4 Server-side Video QoE Measurement Methodology

Based on the takeaways from Sect. 3, in this section we present SSQoE: a server-side QoE score calculation method that relies solely on CDN access logs, and requires no other input from the client-side. The goal of this methodology is to be content agnostic (it should work for any video customer at the CDN), player agnostic (it should not make any assumptions about client video player behavior), and to account for noise that is caused by millions of users connecting to the platform as well as from video artifacts like quality change, ad breaks, etc.

Information Needed from the CDN Access Logs

As our approach relies solely on the CDN access logs, we first describe the necessary fields that are extracted from the logs to perform this analysis. We use three key pieces of information, a *timestamp* of when a client requested a video asset, the *segment ID* (file/segment name) of the video asset, and a *session ID*. We know the video length of the segments before hand, either via a configuration file or via estimations done using `ffmpeg` [3]. These fields are easily available in most web server logs. In case of absence of explicit session ID, a hash of the {client_ip, user_agent} tuple can be used. Hashing the tuple obviates the need for IP-level tracking. Note the video asset segment ID usually encodes the information of the bitrate. A full video such as a movie comprises of many segments. Each segment is numbered incrementally, for example, A1.ts, A2.ts, ... , etc. In this example let us say the first letter is the quality type: A is lowest, B is higher than A, C is higher than B, and so on. With this knowledge, we look at requests from each client and check them in sequence. If their quality changes, for example, A1.ts, B2.ts, A3.ts, we then add it to the rate of fluctuation metric.

Player Buffer Estimation

To estimate if the client side video player ran out of video to play (i.e. suffered rebuffering), we estimate the amount of video the player has in its buffer at a given time. For every request received from a player, we add the respective duration of video (in seconds) to the estimated buffer size. We note that the segment length can change across segment types. For example, ad segments used for server-side ad insertion can be shorter than main content video segments. Therefore, for each session we have in memory the amount of video in the player's buffer. Every time we see a request for a segment of a session, we compare the time difference between the previous request and current timestamp. Using this information we can estimate how much video has been consumed. For example, if at a given time the estimated buffer length for a client is 12 s, i.e. we estimate that the player has 12 s of video available in its buffer, and the next request is seen 15 s after the previous request, then we know that for at least for 3 s the player must not have had any video to play. We refer to this as *rebuffer duration*. With this approachs, without the need for any client-side beacons, we can measure a key element that influences user's QoE.

Calculating QoE Score

Estimating video rebuffer duration from the server-side provides the missing piece needed to measure QoE without client-side participation. However, as shown in the previous section, there are other metrics that prove to be useful.

First, video quality is a function of the bitrate. A higher average bitrate means better video quality, and a better viewing experience. It has been shown before that viewers tend to respond negatively to fluctuations in bitrate and prefer a constant bitrate [8]. Thus a constant lower bitrate impacts user engagement less than many quality switches. A session can have low bitrate due its network subscription package limits, device capabilities, etc. It is not accurate to count every low bitrate session as lack or drop in QoE. Therefore, we keep track of jitter in the bitrate i.e., the *number of times the video stream changes its quality*.

Second, we observed that in most cases the time taken to serve the requests for a client remains fairly constant. We use this information to extract an average time taken to serve a client in a time bin. We analyze this at per ASN or per User Agent granularity to be able to compare similar clients. *Any large fluctuations in average time taken metric is also a good indicator of anomalies.*

Finally, we saw that for a stable video stream the player requests video segments at a fairly constant rate from the server, when performance changes or the player falls behind in a live stream it might request more segments to change quality or catch up in a live stream. *Any large fluctuations in average requests rate is also a good indicator of anomalies.*

We define R_b, T_b, B_b, A_b to describe each metric, as described in Table 1. These metrics represent aggregate information from all sessions in a time bucket b. Equations 1–4 describe how each metric is calculated.

$$R_b = \frac{\sum_{s=1}^{total\_unique\_sessions_b} estimated\_rebuffering}{bucket\_duration_b * total\_unique\_sessions_b} \tag{1}$$

Table 1. Definitions of metrics used for QoE score.

Metric	Definition
R_b	Average estimated rebuffering ratio in bucket b
T_b	Average time taken to serve request in bucket b
B_b	Average bitrate drops in bucket b
A_b	Average requests per session in bucket b

$$T_b = \frac{\sum_{s=1}^{total\_unique\_sessions_b} time\_taken}{total\_unique\_sessions_b} \tag{2}$$

$$B_b = \frac{\sum_{s=1}^{total\_unique\_sessions_b} number\_of\_bitrate\_drops}{total\_unique\_sessions_b} \tag{3}$$

$$A_b = \frac{total\_requests_b}{total\_unique\_sessions_b} \tag{4}$$

We track these metrics individually and calculate a derived anomaly indicator score as well. We represent the QoE score at a given time bucket as a combination of R_b, T_b, B_b, A_b (Eq. 5). We note that it is possible to add more dimensions to our analysis to update the granularity of QoE score.

$$qoe\_score_b = R_b * T_b * B_b * A_b \tag{5}$$

Lower values are better for each dimension of QoE score. For example, a good video stream should have lower estimated rebuffers, less time taken to server requests, lower number of bitrate drops, etc. therefore ***higher values of the QoE score represent anomalies***. Having a single metric to track QoE anomalies provides operational simplicity and a quick litmus test if further analysis is needed. A single metric also simplifies automated anomaly detection and alerting, since standard techniques such as tracking changes (more than 3 standard deviations), cosine similarity, etc. can be easily used.

In this paper, we normalize the value of QoE score between 0 and 1 for comparison with other monitoring metrics, by dividing each calculated value by the maximum QoE score seen in a given time window. However, SSQoE tracks the raw QoE score values and does not rely on this normalization; we simply do this for easier representation and comparison of the results.

Detailed Algorithm to Extract Session Info

In Algorithm 1 we describe how SSQoE calculates the estimated rebuffering along with the other metrics. For the sake of simplicity we present this method as one procedure but our implementation comprises of optimizations that enable us to perform such analysis at scale and at different granularities, e.g. for different CDN customers, ASNs, user agents, etc. For each session, we extract all requests (r_s) seen in CDN logs. We initialize arrays d_s, tt_s, Δt_s, b_s to keep track of rebuffer durations per request, time taken values per request, inter-arrival times between

Algorithm 1. Extract Session Info

1: **procedure** EXTRACTSESSIONINFO(s)
2: $r_s \leftarrow$ *Requests for session s sorted by time*
3: $d_s \leftarrow array[]$ ▷ //Initialize array to store rebuffering duration for every timestamp seen in current bucket
4: $tt_s \leftarrow array[]$ ▷ //Initialize array to store time taken to serve request
5: $\Delta t_s \leftarrow array[]$ ▷ //Initialize array to store inter-arrival times
6: $b_s \leftarrow array[]$ ▷ //Initialize array to track bitrate changes
7: $bsize_s \leftarrow 0$ ▷ //Initialize variable to track estimated video length in client player buffer
8: **for** each request r_{si} in r_s **do**
9: $seg_s \leftarrow$ *Get segment ID from* r_{si} *URL*
10: $b_{s_i} \leftarrow comp(b_{s_i}, b_{s_{(i-1)}})$ ▷ //Track bitrate changes
11: **if** seg_s *was requested before for session* s **then**
12: *Ignore duplicate requests for same* seg_s
13: $vlen_{si} \leftarrow len(seg_s)$
14: $tt_{s_i} \leftarrow$ *time taken to serve the request* r_{si}
15: **if** $i > 0$ **then**
16: $t_i \leftarrow$ *timestamp of request* r_{si}
17: $\Delta t_i \leftarrow t_i - t_{(i-1)}$
18: $bsize_s \leftarrow bsize_s - \Delta t_i$ ▷ //Remove the time difference between requests from estimated buffer length
19: **if** $bsize_s$ ¡ 0 **then**
20: $d_{s_{t_i}} \leftarrow abs(bsize_s)$ ▷ //Player rebuffered same amount as the gap detected in estimated buffer length
21: $bsize_s \leftarrow 0$ ▷ //Reset buffer length
22: $bsize_s \leftarrow bsize_s + vlen_{si}$
23: $tt_{s_i} \leftarrow moving\_average(tt_{s_{(i-1)}}, tt_{s_i})$
24: *return* d_s, tt_s, b_s

subsequent requests, and bitrate changes per request for session s (Steps 1–6). Duplicate requests are ignored, since from the server's perspective same amount of video will be available in the player buffer for two duplicate requests. For each unique request received, for the session in current time bucket, we extract the session ID, compare change in the bitrate from the previous request and finally subtract the time difference between the current and previous request time from our estimated client buffer, *bsize* (Steps 7–18). This describes the duration of video that the client has already consumed. If a subsequent request of a session does not arrive on time, *bsize* falls below zero, indicating a rebuffering event (Steps 19–24). We extract all these metrics per session to be aggregated next.

Global Deployment

As mentioned in the previous paragraph, while the methodology can be described as standalone process running on a server, our implementation of SSQoE leverages the global scale of the CDN. Figure 5 describes an overview of our global deployment. An edge service consumes CDN access logs for a given video provider (a new instance is launched per video provider/customer). Each edge video QoE

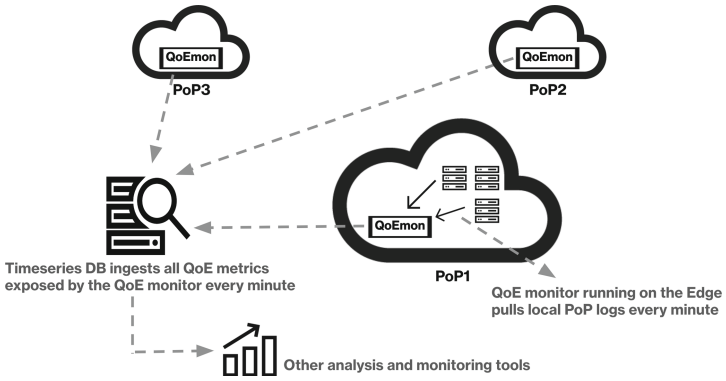


Fig. 5. Distributed log processing for live server-side QoE monitoring.

service only consumes logs for the past one minute from the same PoP as itself, calculates the QoE score and exposes these metrics to a time-series monitoring service at the CDN. Performing this computation at the edge locally per PoP achieves two goals: 1) Get the QoE score as close to real-time as possible, which is most beneficial for live streams, and 2) Achieve redundancy. Operations at one PoP does not affect the QoE monitoring service at another PoP. This provides resiliency in how we capture per PoP QoE anomalies.

5 Validation

In order to validate the methodology used by SSQoE we employ testbed consisting of a server and client that we control, in order to compare the rebuffering reported directly by the client to that measured by SSQoE using server-side logs. We then also perform a detailed comparison between two weeks worth of client-side beacon rebuffering data obtained from a large live sports video streaming provider and the QoE estimates derived using SSQoE on the CDN during same time period.

5.1 Testbed Evaluation

Testbed Setup. For the controlled experiment, we use Nginx as a streaming video server, compiled with the RTMP module [6] to enable HLS live streaming. For the client, we instrument a Javascript HLS video player [4] that we extended with the ability to log timestamps and durations of rebuffering events, as well as timestamps of video segment requests and download durations. The client runs on a local machine while the server is hosted on a cloud VM, with approximately 20ms mean RTT between the two.

We use a 1200-second video that consists of 100 segments. We export the player logs and analyze them to establish the ground truth for the timings of player events like segment download, playback, and rebuffering. On the server

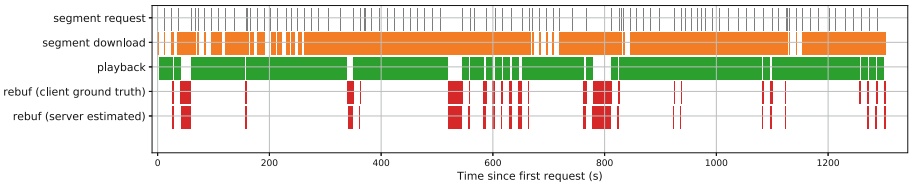


Fig. 6. Testbed evaluation. Comparison of Server-side estimated rebuffering with client reported rebuffering.

side, we export logs that contain request timestamps, request URL and segment duration. We then feed that information to the QoE algorithm in order to estimate rebuffering events and their durations, and we compare that to the ground truth reported from the client logs. During each experiment run we induce artificial network bottleneck using `tc` [5], with inter-arrival times of the throttling events and their durations drawn from a Poisson distribution.

Validation Results

Figure 6 visualizes a single experiment run. The top four rows plot the timestamps and durations of events (request, download, playback and rebuffering) as reported from the client, representing the ground truth. The last row is the output of the QoE estimation based on the server-side logs. The estimated client rebuffering events plotted on that row align well with the client-reported rebuffering events on the row above it, showing that the algorithm detected both the segments for which rebuffering occurred, as well as the corresponding rebuffering durations for the session. The total rebuffering duration reported by the client was 120.8 s and the server estimated 111.3 s of rebuffering (7.9% error).

To evaluate the accuracy of our method over multiple runs, we repeat the experiment 10 times such that the client downloaded 1000 segments (large enough sample to produce meaningful results) and measure the accuracy of detecting rebuffering duration. First we calculate total rebuffer duration as reported by the client and estimated by our method. We calculate error as the absolute difference between rebuffer durations as reported by the two methods. In this test the median error across all the runs was 8.1%. This aligns with our previous single-run error (7.9%) indicating the method's consistency.

In the Fig. 6 we can visualize that rebuffering was detected at the same time by the server and the client. To confirm same is true for our experiment with many runs we evaluate if the rebuffering was seen for the exact same segment by both server and client. Out of 1000 segments across all runs combined (200 minutes of playback time), 121 segments were reported by the client to suffer rebuffering. The algorithm estimated rebuffering in 107 segments, with 3 false positives (segments for which rebuffering was detected, but did not occur) and 17 false negatives (segments for which rebuffering happened but was not detected). There were 104 true positives (segments for which rebuffering happened and it was detected), providing 85.95% accuracy (precision).

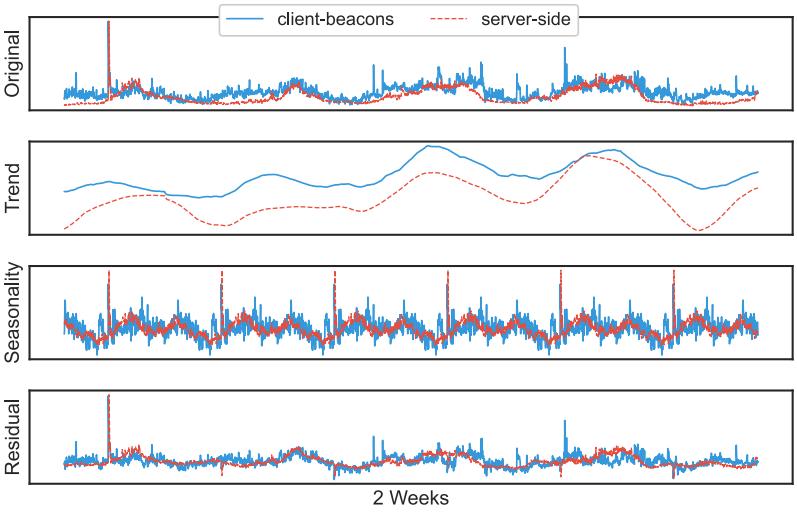


Fig. 7. Time-series decomposition analysis of client-side beacon reported rebuffering vs server-side rebuffer estimation for a large video provider over two weeks. Server-side QoE is able to track both general trend and instantaneous spikes.

5.2 Comparison with Client Beacon Data

As stated in Sect. 2, video providers often use client-side beacons to evaluate video and CDN performance. In order for SSQoE to provide value, it is important that its QoE estimations match the trends of rebuffering ratio reported by such client-side beacons. To validate this, we obtain two weeks of client-side data from a large video provider for which we deployed SSQoE, and compare the results.

Time-series Seasonality Decomposition
Time-series data is composed of systemic components (lower order trends that repeat) and non-systemic components (higher order fluctuations that are local or instantaneous). SSQoE aims to capture both components, i.e. both the general trend of rebuffering as well as large instantaneous anomalies. Therefore, we decompose the client beacon and the SSQoE score time-series into the following three components:

Trend: Lowest component, changes very slowly over long time period (days)
Seasonality: Predictable repeating component, captures local change (hours)
Residual: Anomalous instantaneous component, not predictable (minutes)

Figure 7 (top) shows the original time-series for client-side beacon data and server-side QoE score. To simplify comparison, we normalize each dataset between 0 and 1 as described in Sect. 4. We observe that the two datasets match in overall trends with a *Spearman correlation of 0.7*, and server-side QoE score (SSQoE) matches large anomalies in the client-side beacon data. However, we note that in these original signals there are a few spikes seen in beacon data that

are not obvious in the QoE score data. We also note that there is a large amount of noise (local fluctuations) in both datasets.

We use a standard additive model to decompose the two time-series such that *original = trend + seasonality + residual*, as described in [17]. This well established model quickly provides us an ability to analyze the data at different granularities. Figure 7 also plots these three sub-components, which, combined, compose the original time series. We analyze each component separately to evaluate the precision, recall, and F1 score of SSQoE. We detect anomalies in each component using a sliding window, where a datapoint is marked as anomalous if it is greater than three standard deviations ($> 3\sigma$) compared to past 5 minutes. For each anomaly in the client beacon data we check if the QoE score data also captured an anomaly in the same 5-minute window. If it did, we count that as a true positive, otherwise we flag that as a false negative. Similarly, if we detect an anomaly in QoE score data but no anomaly is reported in the same 5-minute window in the beacon data, we count that as a false positive.

Table 2. For each sub-component, server-side QoE shows high precision values. Server-side QoE tracking is accurate for detecting large instantaneous spikes and long term trend but some local short term fluctuations could be missed.

	Precision	Recall	F1 score
Trend	1	1	1
Seasonality	1	0.57	0.72
Residual	0.89	0.76	0.82

Table 2 shows the results of our precision/recall analysis for each time-series component. The Trend sub-component is expected to be stable and slow changing, and thus anomalies are easily caught. SSQoE captures the trend of the client beacon data accurately. For the Seasonality and Residual component, we get high precision (1 and 0.89) indicating the QoE score is highly accurate in detecting anomalies for local changes (minutes or hours level granularity). Recall is 0.57 for the Seasonality component and 0.76 for the Residual component, indicating that the QoE score can have some false negatives for anomalies that are short lived.

6 Video Performance Monitoring at the CDN

This section presents results from using SSQoE to measure video performance at the CDN. We first demonstrate how per-PoP analysis can aid in fast detection and response to incidents by showing how we used SSQoE to measure streaming performance for the Superbowl LIV event while delivering millions of live streams.

We then explore ways that the measured QoE score can provide additional insight to those gained by existing approaches. CDNs employ a variety of server

and network based performance analysis systems to detect and resolve performance issues [12,26]. In Sect. 6.2 and 6.3 we show examples where server and network metrics were not sufficient during anomalies and how SSQoE exposed the impact of the issues on client-perceived QoE.

6.1 Using Server-Side Video QoE

Region-Wise Performance Evaluation. One challenge with client beacon based reporting, even in cases where access to such data is available to the CDN operators, is that it is not always possible to map the reported problems to the CDN region (PoP) that might be the root cause of a degraded client experience. This problem is much more pronounced for a large anycast-based network such as ours where the decisions regarding which PoP a client talks to is largely decided by the BGP policies. Since SSQoE is implemented on the server-side, it allows to easily perform PoP-level breakdown which can reveal regional anomalies that can happen only in a handful of PoPs.

Superbowl LIV
During the live streaming of Superbowl LIV, along with several other performance monitoring tools, we used SSQoE to measure the performance of the stream delivery. We detected short lived spikes in the QoE score which matched beacon-reported rebuffering ratios. More interestingly, mid-way into the game, the QoE score for the Seattle PoP started trending upwards. Figure 8 shows the normalized QoE score for several PoPs[1]. This trend was not exposed by third-party beacon data used for monitoring performance. We confirmed that this was not a false positive, and were able to identify the root cause of this issue: cache servers hit CPU limits, inflating the time it took to serve the video asset. While a large impact was observed at 02:00 UTC on Feb. 3rd (client reported problems to the NOC[2]), SSQoE actually reported spikes in the QoE score earlier than that. As a result SSQoE reduced the time to take action during one of the largest online streaming events in the US. This example emphasizes the ability of SSQoE to detect problems that might even be missed by client-side beacon metrics. This can enable CDN operators to pinpoint issues in the CDN infrastructure proactively and without any external dependencies.

QoE by User Agents
As shown in Sect. 3, it is possible to evaluate the performance of a video stream over various dimensions using the time taken to serve the requests. Here, we analyze the QoE grouped by user agent. In Fig. 9 we show QoE scores for the most popular browser/operating systems that are observed in the CDN access logs. Chrome on Windows 7 was the user agent with the highest QoE score, which translates to more rebuffering. While the exact version number of Chrome is not

[1] For visualization simplicity in the figures, each PoP is represented by the city/metro name it is located in.

[2] Network Operations Center (NOC) is responsible for 24×7 monitoring of global CDN performance and respond to customer incidents.

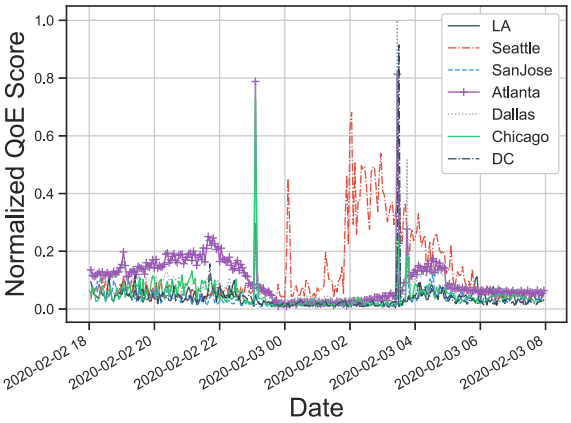


Fig. 8. Normalized QoE scores during Superbowl 2020 per PoP. Spike in the QoE score of Seattle PoP due to CPU bottleneck could be easily identified (3rd 02:00 UTC). Other rebuffering spikes were also accurately caught at several PoPs.

shown in the figure, we note that this is an older version of Chrome (74). We also carefully acknowledge that lower performance of this browser/OS combination might also be an artifact of the device CPU/memory since older devices running these older versions of browsers and OSes also tend to have older specifications.

6.2 QoE vs Server Metrics

In this section we look at QoE scores in the context of CDN performance metrics. One of the metrics tracked in order to monitor CDN health is the ratio of total number of server-side errors (HTTP code 5xx) to the total number of requests, defined as the *error ratio*. Under normal operations, this error ratio is under 0.3%. The baseline behavior of this metric can differ at different PoPs based on current conditions, which can introduce noise to the metric.

Monitoring the error ratio is useful to understand performance of the CDN and origin cache performance for cache-fill. If a large number of users request for the same video segment (e.g. during a popular live event) and if the error ratio is high, many clients can take a performance hit. However, we argue that QoE score can be a better metric for tracking such impact. By looking at the error ratio, it is hard to estimate how long the impact of missing segments lasted, or what the intensity of the impact was i.e., how many concurrent users suffered. Moreover, video players are designed to buffer segments a few seconds ahead of current play time, hence a small ephemeral spike in error ratio may not always affect the video playback.

Figure 10 shows SSQoE scores and error ratios from a North American PoP. In this case, for a few hours the video provider origin had performance issues and returned 5xx responses. During such events it is operationally difficult to evaluate the actual impact on end users based on HTTP errors alone. In Fig. 10 we see

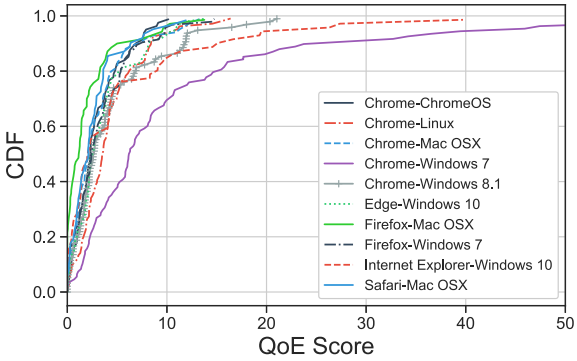


Fig. 9. CDF of QoE Scores by top user agents. Older browsers/OSes often perform worse.

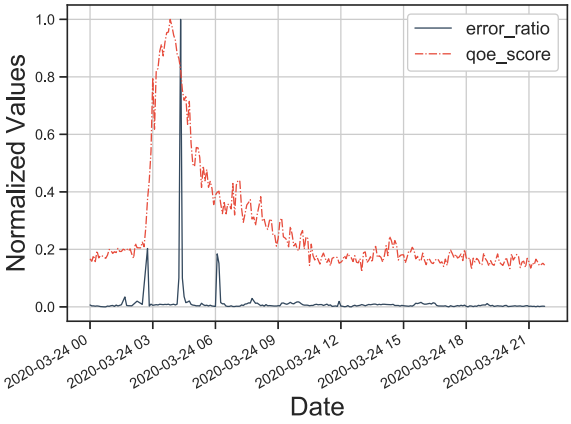


Fig. 10. QoE score compared to HTTP error ratio. QoE score provides a picture of user impact during origin server performance problems than the error ratio.

a correlated spike in error ratio and the QoE score. However, we observe that error spikes were more instantaneous (once a content is available at the origin, the error ratio subsides). In contrast, the QoE score was increased for a few hours after the last spike in the error ratio, indicating availability but degraded origin server performance. Several mitigation steps were taken to alleviate this problem. In this case, monitoring the QoE score instead of error ratio provided a more accurate picture on the intensity of the impact. This also emphasizes the SSQoE's ability to track performance issues whose root cause lies in early steps of the live video streaming pipeline (Fig. 1).

6.3 QoE vs Network Metrics

A common practice in performance monitoring at content providers is to monitor the RTT and retransmits towards clients. This information is then used to infer problems in connectivity either at the client ISP, transit provider, or at the CDN itself. Systems such as *BlameIt* [19] operate in such fashion. Network metrics, however, do not capture a complete view of the media delivery stack. In our experience, other factors (as described in Sects. 2 and 3) can cause client level impacts and lead to rebuffering.

To demonstrate this, we show the QoE score vs average RTTs and average retransmits for a 2-day period from a North American PoP in Figure 11. One of the transit providers for this PoP faced a connectivity issue during this period. As shown in the figure, the RTT and retransmit aggregates did not fluctuate much. However, we clearly see a spike in the QoE score around the time connectivity issue was reported (starting at 20:00 UTC on the 17th). Due to fallback routes and other network-layer load balancing it is possible that network layer issues get masked, but client sessions resetting or suffering playback issues are captured by the QoE score.

Wireless ISPs

SSQoE measurements also enable analysis by ISP. This adds value for operational monitoring by providing an additional dimension to compare and evaluate performance. Nowadays users consume large volumes of video content on their mobile phone over wireless networks. It is common practice to monitor network metrics towards such wireless carrier ISPs by tracking RTTs and retransmits. Here, we evaluate whether lower RTT translates to lower rebuffering. In Fig. 12a we plot the average RTTs from one of the CDN's North American PoPs towards the top three wireless carriers in the U.S. over 8 days. We normalize these values between 0 and 1, where maximum RTT across the ISPs is set to 1. The RTTs profiles for the three wireless carriers are different at this PoP, making it a good candidate to evaluate if lower RTTs lead to less rebuffering. Since we are trying to eliminate the network impact, we only look at estimated rebuffering captured using the method described in Sect. 4.

Interestingly, we note that Wireless ISP 1 has the lowest rebuffering but the 2nd highest RTT. We also note that the distribution of estimated rebuffering is similar for wireless ISPs 2 and 3 even though their RTT profiles vary. This indicates that network metrics do not necessarily capture user perceived experience.

The above analysis shows how SSQoE can be used to profile ISPs based on QoE scores. Categorizing ISPs by performance in video delivery can help identify providers which are persistently under-performing, and can help drive traffic engineering decisions during large live streaming events.

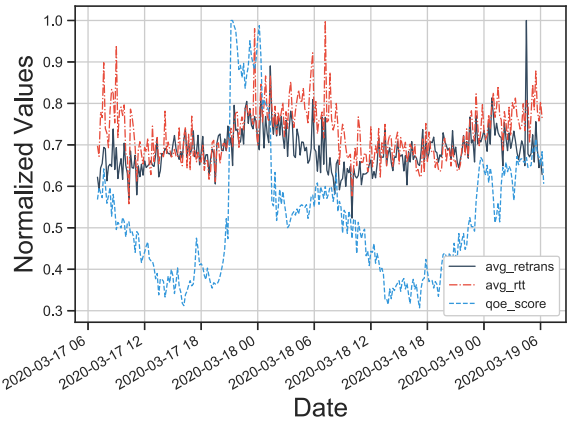
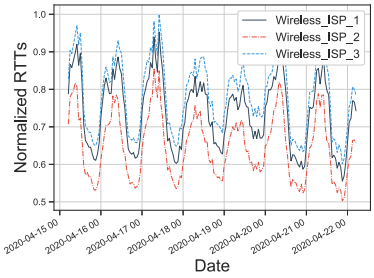
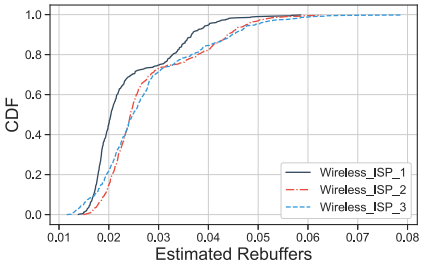


Fig. 11. QoE score compared to RTT and retransmits. A transit provider connectivity problem, from 17th 20:00 to 18th 01:00, is better captured using the QoE score.



(a) Normalized RTTs for 3 largest wireless carriers in the U.S from a North American PoP. Each ISP has a unique RTT profile.



(b) CDF of estimated rebuffers used in QoE calculations. Wireless ISP 1 shows lowest rebuffering but has 2nd highest RTT measurement (left).

Fig. 12. RTT vs QoE for U.S wireless carriers.

7 Discussion

There are some aspects of video content delivery that this paper does not cover in detail and must be kept in mind while deploying such method.

Variable Segment Size and Quality
A common practice to deliver large files is to chunk the file. Here the original file (could be a few GBs) is broken into similarly sized chunks (in the order of MBs). Each chunk is treated as a separate content asset. In such cases, one video segment request from the client can map to more than one chunk. Care needs to be taken to account for flow completion times of each chunk as well as add only the corresponding video segment duration to buffer estimation algorithm (described in Sect. 4) that the chunk represents. Note that subsequent requests

from the client can be of different quality therefore video duration of chunks in subsequent requests can vary.

Estimation During Fewer Video Sessions
While the proposed method detects rebuffering on a per-session granularity, we note that relying on this signal when the number of sessions is too few may lead to noise. We aggregate the estimated rebuffering for all sessions that lie in the current time bin and draw aggregated average value, therefore, it is imperative that there are enough data points where the average is a reliable representative value. Too few data points result in averages sensitive to outliers.

We perform our analysis using a minimum of 50 sessions per minute, a threshold which showed empirical evidence in tracking meaningful rebuffering ratios. We have designed SSQoE with CDN delivery and performance-based decisions in mind. We do not focus on all client behavior metrics that video providers might want to track from their players, such as video start up failures, ad engagement metrics, protocol performance level A/B testing like QUIC vs TCP clients, etc. At present, these analyses are not supported by our system.

Impacts of user Interactions with the Video Stream
It is possible that user actions such as pausing, switching from WiFi to LTE, etc. can impact buffer occupancy and lead to estimation inaccuracies. SSQoE score is by design an aggregate metric that combines the performance (rebuffer estimations, time taken to download segments, etc.) of all the sessions in a time window. It is highly unlikely that a large fraction of viewers pause the video or otherwise introduce similar user behavior at the same time. Thus on a large scale, individual session inaccuracies become negligible. The aggregate signal represents how performance looks on average for many users that are e.g. in the same ASN or that connect to the same PoP. Future work can quantify and account for this small fraction of error margin in SSQoE.

Change Management
The QoE score's ability to track user level impacts due to degraded server side performance such as CPU bottlenecks (Fig. 8) or origin server issues (Fig. 10) makes it a good candidate for change/configuration management. During a recent update to the cache management software at the CDN, a bug caused degraded I/O performance. The impact this bug caused on live video streams was captured by our methodology where the QoE score spiked up by 3 times its baseline value. Thanks to our automated detection, our site-reliability team could proactively work on rolling back the change. We are working towards integrating QoE score into the CDN's ML-based monitoring service.

Other Considerations
We do not make any protocol assumptions while estimating the QoE score. We have tested the feasibility of SSQoE for HLS and DASH with success. It is possible, however, that future protocols that support low latency streaming such as WebRTC might need to be evaluated to determine the efficacy of SSQoE.

Ethical considerations are kept in mind while designing proposed method. We do not track or expose anything more than what is already captured in the

CDN access logs. We only track the performance of the video stream by looking at the meta information. The content segments remain encrypted (HTTPS).

8 Related Work

Measuring video performance and evaluating QoE has been studied from many angles. To the best of our knowledge our work is the first to propose a pure server-side client player buffer estimation and show its feasibility at a large multi-tenant scale. However, there are several methods that are relevant to our work. Broadly we classify these into three categories based on whether they use QoE for: a) video performance monitoring and characterization of traffic, b) steering traffic at large content providers, or c) evaluating change in network configurations.

Video Performance Monitoring
Poor video performance leads to less user engagement [15,23]. In [7] authors quantify the impact of QoE metrics in user engagement during a large live OTT streaming event. Specifically, this work points out that bitrate and rebuffering have the most impact on how users engage. The authors propose using PCA and Hampel filter for live detection of QoE impairments. We take insights from this work and build more scalable method that instead of performing resource consuming PCA, estimates the client player behavior to detect rebuffering. In [18], using client-side metrics authors identify that video quality is determined by subset of critical features. Tangential to our server-side scoring mechanism, their methodology provides a QoE prediction scheme that can be deployed on beacon data at scale.

Authors in [11,14,20] address the challenge of detecting QoE in encrypted traffic. As video streaming platforms provide end-to-end encryption, it has become challenging for middle-mile network providers to perform video specific optimizations to specifically target video traffic with the goal of improving QoE. Our work differs from the focus of these papers. SSQoE is designed for the video provider infrastructure (i.e., CDN) where the TLS termination occurs. The CDN can identify unique video traffic, video segments, etc. to perform the proposed QoE estimation. In [22], similar to our work's motivation, authors emphasize that network metrics alone may miss QoE degradation event. Here, the authors propose using user behavior information such as pausing, reducing viewing area as indicators to predict QoE degradation. This differs from our server-side methodology where we do not rely on user metrics. User metrics are often tracked by the video player and in case of 3rd party CDNs they may not be available to estimate QoE.

In [21] use automated tests on the client side to interact with several online services and throttle throughput to monitor video performance. This work focuses on understanding the diversity in how different streaming services operate. Different platforms might optimize their player's ABR behavior for different guarantees of QoE, such has utilizing only limited amount of available bandwidth or compromising bitrate to keep up with live stream. The methods used in this paper revolve around a one time study to understand the landscape of video

streaming. It is not designed to be an operational component of a multi-tenant CDN provider. On the other hand, given the diversity in streaming landscape this work motivates the need to perform measurements from the server-side and use a method such has ours that is completely player agnostic. YouTube's video traffic has been studied by many researchers [9,13]. These works differ from ours since they are either aimed at specifically analyzing YouTube's traffic behavior or trying to understand the ISP and CDN caching implications on QoE all using client-side data.

Traffic Management Systems
Recent performance measurement and traffic management systems developed by Google [28], Facebook [26,27], Mircrosoft [12,19] use several measurement schemes to evaluate performance and use that information to either localize faults or pick an alternate route for egress traffic that will lead to better QoE. In particular, `EdgeFabric` [27] and `Espresso` [28] focus on egress steering methodologies. Both of these systems leverage data from the client apps to gain performance insights. `Odin` [12] uses several data sources from both client and server-side along with active measurements for CDN performance evaluation. Although these systems are relevant for measuring performance, they do not explicitly track video performance. A more general approach proposed by Facebook is by tracking `HDratio` [26], which indeed focuses on video performance. The authors propose using the achievement of 2.5Mbps throughput i.e., enough to serve HD content as an indicator to measure performance. Similar to our work, this method also relies on only server-side measurements and can be applied for a multi-tenant environment. However, relying on such hard thresholds in a multi-tenant mixed-workload streaming landscape does not scale well. It is operationally hard to perform different analysis for HD, Ultra-HD, 4K, multiple bitrate qualities using multiple thresholds. Moreover, tracking throughput gives you a red flag on network performance degradation, there is no guarantee that the client player did actually suffer rebuffering. Our estimating buffer algorithm tracks the actual client player buffer therefore every time an event is detected i.e., estimated buffer at client is zero, we know with high accuracy the video has rebuffered. Using throughput based metrics also do not work well for server-side Ad insertion in the video stream. Ads in the middle of the video could be encoded in a lower or higher bitrate in which case a change in throughput is expected and may not necessarily indicate performance degradation. As shown in Sect. 6, our proposed methodology does not suffer from such unintended impacts.

Configuration Evaluation
In [16] authors measure QoE to evaluate network buffer change. Authors in this work attempt to measure the impact on QoE while tuning network buffer in a testbed. However, authors do not measure important user engagement impacting metric such as rebuffering [7]. We agree with the motivation of this work, that QoE impacts of network configurations are largely unknown. We have proposed an easily scalable server-side methodology that we hope will be used by future research on network parameter tunings and evaluate impacts on client player buffer. The proposed methodology is part of our change/configuration

management strategy at the CDN, including for a recent change we made for our network buffer sizes. While the evaluation of that change is out of scope of this work, we note that server-side QoE was able to accurately track the progress of this network change and we encourage other large content providers to include such metric in their change management process as well.

9 Conclusion

With the rise of video streaming applications on the Internet, their ingest, distribution, and performance monitoring has become increasingly complex. Current state of the art monitoring solutions rely on client-side beacons which require considerable instrumentation. Moreover, this beacon data is not easily exposed to multi-tenant third-party CDN providers. This results in cases where CDNs deliver a bulk of the video traffic without proper visibility into client-perceived QoE and performance.

In this paper, we analyzed the video processing pipeline, characterized the video streaming workload on a large scale CDN and derived key features that can be tracked from the server-side to understand client-perceived QoE. We then presented and validated SSQoE, a method for estimating client rebuffering using passive measurements, by analyzing a sequence of requests from CDN access logs to derive a QoE score that represents the health of video stream. Traditional client-side metrics can only reveal the device or last-mile problems. Mapping them to the CDN infrastructure is generally not easy, making client-side beacons less viable for large scale CDN operations. Server-side QoE estimation has been in operation globally on our CDN for the past year. To the best of our knowledge, this is the largest deployment of server-side video monitoring at a commercial CDN. It is currently used for monitoring some of the biggest live news, sports events, conferences, movie releases that millions of users engage with, and it has helped identify issues using the QoE score in near-real-time and correlate performance degradation with other CDN insights during large scale events.

We have explored the possibilities of server-side QoE analytics and invite the industry and academia to collaborate, contribute, and explore more use cases in this direction.

References

1. Adaptive bitrate (ABR). https://en.wikipedia.org/wiki/Adaptive_bitrate_streaming
2. Conviva platform. https://www.conviva.com/about/
3. FFmpeg utility. https://ffmpeg.org
4. HLS.js player. https://github.com/video-dev/hls.js
5. Linux Traffic Control (TC) utility. https://man7.org/linux/man-pages/man8/tc.8.html
6. NGNIX RTMP. https://github.com/arut/nginx-rtmp-module

7. Ahmed, A., Shafiq, Z., Bedi, H., Khakpour, A.: Suffering from buffering? detecting qoe impairments in live video streams. In: 2017 IEEE 25th International Conference on Network Protocols (ICNP), pp. 1–10 (2017)
8. Ahmed, A., Shafiq, Z., Khakpour, A.: Qoe analysis of a large-scale live video streaming event. In: Proceedings of the 2016 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Science, SIGMETRICS 2016, pp. 395–396. Association for Computing Machinery, New York (2016)
9. Añorga, J., Arrizabalaga, S., Sedano, B., Goya, J., Alonso-Arce, M., Mendizabal, J.: Analysis of Youtube?s traffic adaptation to dynamic environments. Multimedia Tools Appl. **77**, 7977–8000 (2017). https://doi.org/10.1007/s11042-017-4695-9
10. Böttger, T., Cuadrado, F., Uhlig, S.: Looking for hypergiants in PeeringDB. SIGCOMM Comput. Commun. Rev. **48**(3), 13–19 (2018)
11. Bronzino, F., Schmitt, P., Ayoubi, S., Martins, G., Teixeira, R., Feamster, N.: Inferring streaming video quality from encrypted traffic: practical models and deployment experience. Proc. ACM Meas. Anal. Comput. Syst. **3**(3), 1–25 (2019)
12. Calder, M., et al.: Odin: microsoft's scalable fault-tolerant CDN measurement system. In: USENIX NSDI, April 2018
13. D'Alconzo, A., Casas, P., Fiadino, P., Bar, A., Finamore, A.: Who to blame when YouTube is not working? detecting anomalies in CDN-provisioned services. In: 2014 International Wireless Communications and Mobile Computing Conference (IWCMC), pp. 435–440 (2014)
14. Dimopoulos, G., Leontiadis, I., Barlet-Ros, P., Papagiannaki, K.: Measuring video QoE from encrypted traffic. In: Proceedings of the 2016 Internet Measurement Conference, IMC 2016, pp. 513–526. Association for Computing Machinery, New York (2016)
15. Dobrian, F., et al.: Understanding the impact of video quality on user engagement. Commun. ACM **56**(3), 91–99 (2013)
16. Hohlfeld, O., Pujol, E., Ciucu, F., Feldmann, A., Barford, P.: A QoE perspective on sizing network buffers. In: Proceedings of the 2014 Conference on Internet Measurement Conference, IMC 2014, pp. 333–346. Association for Computing Machinery, New York (2014)
17. Hyndman, R.J., Athanasopoulos, G.: Classical decomposition of time-series data. https://otexts.com/fpp2/classical-decomposition.html
18. Jiang, J., Sekar, V., Milner, H., Shepherd, D., Stoica, I., Zhang, H.: CFA: a practical prediction system for video QoE optimization. In: Proceedings of the 13th USENIX Conference on Networked Systems Design and Implementation, NSDI 2016, pp. 137–150. USENIX Association, USA (2016)
19. Jin, Y., et al.: Zooming in on wide-area latencies to a global cloud provider. In: ACM SIGCOMM, August 2019
20. Khokhar, M.J., Ehlinger, T., Barakat, C.: From network traffic measurements to QoE for internet video. In: 2019 IFIP Networking Conference (IFIP Networking), pp. 1–9 (2019)
21. Licciardello, M., Grüner, M., Singla, A.: Understanding video streaming algorithms in the wild. In: Sperotto, A., Dainotti, A., Stiller, B. (eds.) Passive and Active Measurement. pp, pp. 298–313. Springer International Publishing, Cham (2020). https://doi.org/10.1007/978-3-030-44081-7_18
22. Mok, R.K., Chan, E.W., Luo, X., Chang, R.K.: Inferring the QoE of http video streaming from user-viewing activities. In: Proceedings of the First ACM SIGCOMM Workshop on Measurements up the Stack, W-MUST 2011, pp. 31–36. Association for Computing Machinery, New York (2011)

23. Nam, H., Kim, K., Schulzrinne, H.: QoE matters more than QOS: why people stop watching cat videos. In: IEEE INFOCOM 2016 - The 35th Annual IEEE International Conference on Computer Communications, pp. 1–9 (2016)
24. Richter, P., et al.: A multi-perspective analysis of carrier-grade NAT deployment. In: Proceedings of the 2016 Internet Measurement Conference, IMC 2016, pp. 215–229. Association for Computing Machinery, New York (2016)
25. Rüth, J., Wolsing, K., Wehrle, K., Hohlfeld, O.: Perceiving QUIC: do users notice or even care? arXiv:1910.07729 (2019)
26. Schlinker, B., Cunha, I., Chiu, Y.-C., Sundaresan, S., Katz-Bassett, E.: Internet performance from facebook's edge. In: Proceedings of the Internet Measurement Conference, IMC 2019, pp. 179–194. Association for Computing Machinery, New York (2019)
27. Schlinker, B., et al.: Engineering egress with edge fabric: steering oceans of content to the world. In: Proceedings of the Conference of the ACM Special Interest Group on Data Communication, SIGCOMM 2017, pp. 418–431. Association for Computing Machinery, New York (2017)
28. Yap, K.: et al.: Taking the edge off with espresso: scale, reliability and programmability for global internet peering (2017)