ReCLive: Real-Time Classification and QoE Inference of Live Video Streaming Services

Sharat Chandra Madanapalli*, Alex Mathai†, Hassan Habibi Gharakheili* and Vijay Sivaraman*
*School of Electrical Engineering & Telecommunication, UNSW, Sydney, Australia
†BITS Pilani, India
Emails: {sharat.madanapalli, h.habibi, vijay}@unsw.edu.au, f2016339p@alumni.bits-pilani.ac.in

***Abstract*—Social media, professional sports, and video games are driving rapid growth in live video streaming, on platforms such as Twitch and YouTube Live. Live streaming experience is very susceptible to short-time-scale network congestion since client playback buffers are often no more than a few seconds. Unfortunately, identifying such streams and measuring their QoE for network management is challenging, since content providers largely use the same delivery infrastructure for live and video-on-demand (VoD) streaming, and packet inspection techniques (including SNI/DNS query monitoring) cannot always distinguish between the two. In this paper, we design and develop *ReCLive*: a machine learning method for live video detection and QoE measurement based on network-level behavioral characteristics.**

***Index Terms*—traffic classification, video streaming, QoE inferencing, machine learning.**

I. Introduction

Live video streaming consumption grew by 65% from 2017 to 2018 [1] and the recent pandemic situation fueled further growth with major events being streamed online [2]. YouTube Live is widely used for concerts, sporting events, and video games. Twitch is a popular platform for streaming video games from individual gamers as well as from tournaments. Internet Service Providers (ISPs) are keen on gaining fine-grained visibility into live video streams, enabling them to monitor quality of experience (QoE) for live video streaming over their networks, and where necessary enhance QoE for their subscribers by dimensioning bandwidth appropriately or applying policies for traffic management. However, ensuring good QoE for live video streams is challenging, since clients per-force have small playback buffers (a few seconds at most) to maintain a low viewing latency. Even short time-scale network congestion can cause buffer underflow leading to a video stall, causing user frustration.

Network operators lack the tools today to distinguish live streaming flows in their network, let alone know the QoE associated with them. Content providers like YouTube use the same delivery infrastructure for live streaming as for on-demand video, making it difficult for deep packet inspection (DPI) techniques to distinguish between them. Indeed, most commercial DPI appliances use DNS queries and/or SNI (Server Name Indication) certificates to classify traffic streams, but these turn out to be the same for live and on-demand video (*e.g.,* in Youtube), making them indistinguishable. In this work, we therefore pursue an alternative approach that is based on the behavioral profile of the traffic flows. Extracting key attributes from the network behavior allows us to build machine learning models that can distinguish live from on-demand video, as well as estimate user QoE metrics in terms of resolution and buffer stall events. This allows network operators to detect and measure live video streams purely from network behavior, without requiring any assistance from end-clients or server/CDN end-points.

Our specific contributions are three-fold: **First**, we collect traffic traces from field (university campus) and lab (using synthetic network conditions) of around 23,000 video streams spanning the two popular providers, namely Twitch and YouTube, and analyze their patterns (§III). We make a key observation that requests for manifest files and media segments display markedly different patterns. Focusing on the time series of content requests, therefore, lets us develop our **second** contribution wherein an LSTM (long short term memory) model is trained to distinguish live from on-demand video with an accuracy over 95% across both providers (§IV). Our **third** contribution develops a method that uses the chunk-based features collected from the network flows to estimate QoE metrics for live streaming in terms of resolution using a random forest classifier with 93% accuracy, and predict buffer stalls using a statistical model with an accuracy of 90% (§V). Our methods offer real-time visibility into live video streaming and its QoE metrics.

II. Related Work

Live Video Streaming: Several aspects of live video streaming have been studied by researchers including QoE modeling/measurement. Prior work on QoE of live videos range from theoretical models [3] to study buffer dynamics to analysis of HTTP logs of CDNs to predict resolution and buffer stalls [4], [5]. Our work, instead, focuses on identification of video streams and predicting QoE of encrypted live streams from real-time network traffic behavior. We note that HTTP logs and QoE metrics are typically available to the CDNs or content providers. Our work is positioned to support ISPs who do not have access to such logs but require to infer the QoE of live videos traversing their network.

Application Classification: Traffic classification has been a widely studied cross-disciplinary field and more recently, researchers have begun the use of machine learning/deep learning models for classification of network traffic [6]. Authors of [7] classify video streaming versus large downloads by using

978-0-7381-3207-5/21/$31.00 ©2021 IEEE

manually extracted features from network flow activity to train random forest classifiers. In contrast, deep learning-based methods leverage the automatic feature extraction: work in [8] classifies type of traffic (*e.g.,* Mail, VoIP, Chat) using a CNN on the first 784 bytes of a session. In our work, we use LSTM-based model for classifying live and VoD streams. Unlike existing methods that use expensive packet-level and/or byte-level features, we rely on periodic flow-level request counters (collected every half-second) making it relatively scalable to high traffic rates.

Video QoE From Network: Recently, many researchers [9], [10], [11], [12], [13] have studied QoE metrics for video streaming services across providers such as YouTube, Netflix, Facebook, Bilibili and Amazon, particularly focusing on VoD. Among existing works, only [11] studied QoE for live streaming services (Twitch) by estimating only the resolution metric. We note that not only does live video differ in delivery, but it also has more stringent QoE requirements. Further, prior works predominantly performed post-facto analysis of video streaming QoE using features extracted from pcap traces [9], [10], [13], or CDN logs [4], [5]. Our work develops methods to distinguish live streams and predict their QoE metrics (resolution and buffer stalls) in real-time by building upon existing literature. Our design choices aim at scalability and ease of deployment by identifying inexpensive traffic attributes (to compute), and building machine learning models that are "general" (work across providers) and "simple" (lower-memory footprint, ease of training and deployment).

III. Live Video Characteristics & Dataset

Live video streaming refers to video content which is simultaneously recorded and broadcasted in real-time. The content uploaded by the streamer sequentially passes through ingestion, transcoding, and a delivery service of a content provider before reaching the viewers ([14], [15], [16]). Modern live streaming clients typically use protocols (*e.g.,* HTTP Live Streaming) wherein they fetch *manifest* files containing URLs to the latest transcoded media segments. The client then downloads few segments and maintains a short buffer to keep the latency between streamer and viewer to a minimum. This increases the chances of buffer underflow as network conditions vary, making live videos more prone to QoE impairments such as resolution drop and video stall ([4], [5]).

In contrast, VoD streaming uses HTTP Adaptive Streaming (HAS) and involves the client requesting segments from a server which contains pre-encoded video resolutions. This not only enables the use of sophisticated multi-pass encoding schemes which compress segments to smaller sizes, but also lets the client maintain a larger buffer making it less prone to QoE deterioration.

A. Network Activity Analysis

Fig. 1(a) and 1(b) show the client's network behavior (at 100 ms granularity) of representative live and VoD Twitch streams from our dataset. The live streaming client downloads video segments every two seconds. In contrast, the VoD client begins by downloading multiple segments to fill up a long buffer and then fetches subsequent segments every ten seconds. Thus, the periodicity of segment downloads seems to be a very important feature to distinguish live from VoD streams.

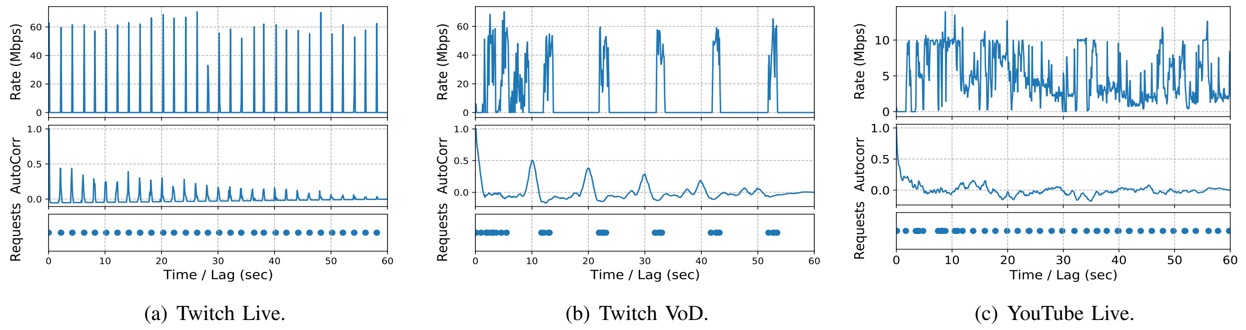
We first estimate the periodicity for download signals by applying auto-correlation function followed by peak detection. Fig. 1 shows the auto-correlation value at different time lags (integral multiple of a second) for both live and VoD Twitch streams. Note that the auto-correlation sequence displays periodic characteristics just the same as the signal itself, *i.e.,* lag = 2s for live Twitch and lag = 10s for VoD Twitch. Further, we also notice peaks at multiples of the periodicity value. From this observation, we attempt to classify video streams as live or VoD using a Random Forest classifier whose inputs are the first three lag values at which the auto-correlation signal peaks, and achieved an accuracy of about 89.5% for Twitch videos. However, this method does not generalize well to other content providers due to several identifiable challenges. Varying network conditions causes the auto-correlation to fail in identifying the periodicity, as shown in Fig. 1(c) for a sample of YouTube live streaming. Further, user triggered activities like trick-play for VoD seem to distort the time-trace signal, causing it to be mis-classified as a live stream.

To overcome these challenges and better understand the delivery mechanism of live videos, we collected the playback data from the video client such as latency modes, buffer sizes and resolutions (using browser automation tool described in §III-B). We used Wireshark (configured to decrypt SSL) to gain insights into protocols being used, patterns of the requests made for content and manifest files, their periodicity, and the available latency modes as shown in Table I.

While SNIs may seem sufficient to distinguish live and VoD streams at least for Twich, they can be changed by content providers at any time. Further, with an increasing adoption of eSNI (encrypted SNI) supported by TLS 1.3, server names will not be accessible from the network traffic. Thus, it is needed to identify certain patterns in the network behavior of live streaming applications in order to distinguish them from VoD streams. We observed that for each media segment/manifest file, the video client in the browser made an HTTP request that was also seen as an upstream packet on the wire. Due to the use of TLS, the HTTP request is hidden in the packet data. Thus, we tag the upstream packets to be request packets when they contain a payload greater than 26 bytes (the minimum size of HTTP payload). Fig. 1 clearly illustrates how the request packets correlate with the video segments being fetched – however, the auto-correlation approach failed to capture it (as indicated in the YouTube instance). We found that the time-trace signal of request packets: (a) is periodic and indicative of the streaming type, even in varying network conditions, (b) is less prone to noise in case of user triggered activities, and (c) can be well generalized across content providers.

B. Dataset

Having identified request packets as a key feature to distinguish live from VoD streams, we collect data of around $23,000$ video streams from Twitch and YouTube. In this



(a) Twitch Live. (b) Twitch VoD. (c) YouTube Live.

Figure 1: Network behavior: download rate profile, its auto-correlation, and request packets.

Table I: Fetch mechanisms of Twitch and YouTube video streaming.

Provider	Type	Protocol	Request for Manifest	Frequency	Latency modes	Service Endpoint SNI
Twitch	VoD	HTTP/2	Once	10s	-	`vod-secure.twitch.com`
	Live	HTTP/1.1	Periodic (different flow)	2/4s	Low, Normal	`video-edge*.abs.hls.ttv.net`
YouTube	VoD	HTTP/2 + QUIC	Once	5-10s	-	`*.googlevideo.com`
	Live	HTTP/2 + QUIC	Manifestless	1/2/5s	Ultra Low, Low	`*.googlevideo.com`

section, we describe two tools which we built to: (a) automate the playback of video streams, and (b) collect data of video streams from our campus network – we obtained appropriate ethics clearances (UNSW Human Research Ethics Advisory Panel approval number HC16712) for this study.

Automated Data Collection: The first tool we built records both network telemetry (flow-level counters) and user experience metrics of video streams. The tool automatically plays (on personal computers) live and VoD streams from Twitch and YouTube. The tool has three main containerized components: a browser controlled using the Selenium [17] library, a network telemetry component called "FlowFetch", and an orchestrator that co-ordinates the playback of videos and data collection. This tool also has a network conditioner module to artificially impose network conditions with the help of the `tc` tool available in Linux distributions.

The orchestrator first fetches a list of videos to be played. For both live and VoD, the tool fetches the top trending videos from a particular provider. It then iterates through the video list and performs the following steps for each video: (1) signals the FlowFetch component to start collecting network data, (2) plays the video on the browser, (3) collects the experience metrics reported by the player such as resolution, buffer level, and stores them in a csv file. During the playback of the video, FlowFetch collects and stores network data into multiple csv files. After the video is played for a fixed amount of time (5 minutes in this work), the orchestrator signals FlowFetch to stop collecting data and repeats the steps on the next video.

We have developed the FlowFetch component in Golang to collect network telemetry data. It can read packets from a pcap file or a network interface. FlowFetch collects telemetry for a network flow, identified by 5-tuple: *src* and *dst* IP addresses, transport layer ports and protocol. In this component, multiple fully programmable telemetry functions can be associated with a flow. Two functions used in this paper are (1) ***request packet counters*** and (2) ***chunk telemetry***. The first function exports the number of request packets (identified by the packet payload length) observed on the flow every 500ms. The second function (further described in §V) exports metadata on isolated media chunks for estimating QoE metrics.

Table II: Summary of our dataset: number of streams.

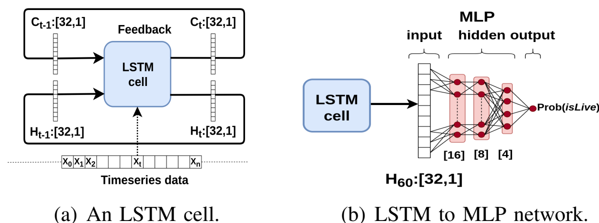
Source	**Twitch** Live	**Twitch** VoD	**YouTube** Live	**YouTube** VoD
Tool	2587	2696	1430	1719
Campus	12534	1948	-	-

To isolate network flows corresponding to the video stream, FlowFetch performs a regex match on SNI field captured in the TLS handshake of a HTTPS flow (mentioned in Table I). Along with network telemetry data collected for each video, the orchestrator collects playback metrics like resolution and buffer health from the video player. These playback metrics that are stored along with the network telemetry data will form a collocated time series dataset for each video stream.

Campus Data Collection: We additionally collected data for Twitch videos from our university campus traffic (from both WiFi and Ethernet clients). We received a mirror of all the traffic between campus and the Internet to one of our servers. We used FlowFetch to collect data of real user-generated Twitch live and VoD flows from a variety of device types like personal computers, tablets, and phones. As described above, by using SNI regex matches, FlowFetch filters and tags the collected flow as Live or VoD. Since we have no control over the device/user streaming the videos, none of the playback metrics such as resolution etc. are available. Hence, this data set can only be used for classification purposes. Table II. shows the number of video streams collected across providers using our tool and from the campus traffic. As evident in Twitch campus data, live streams vastly outnumber VoD streams. While time limits were set for our automated tool, there were none for the campus traffic. In total, we collected over 1000 hours of playback of videos across both the providers.

IV. Classification: Live versus VoD

We now design a general neural network architecture for a classifier that takes a time-series vector consisting of request



(a) An LSTM cell. (b) LSTM to MLP network.

Figure 2: $Model$ structure for binary classification.

packet counts. Using our collected data, we then train one instance of the classifier for each provider.

A. LSTM Model Architecture

We demonstrated in §III that for a network flow, the requests made for content are evidently different in live streaming compared to VoD streaming. This feature is captured in our dataset wherein the count of requests is logged every 500ms for a given network flow. To enable real-time classification, we consider only 30 seconds of the playback as a time window over which we aim to classify the stream. We thus obtain 60 data-points that form the input to our model as denoted by:

$$\vec{X} = [x_1, x_2, ..., x_{59}, x_{60}] \tag{1}$$

As we saw in Fig. 1, live streams display more frequent data requests, distinguishing their network behavior across various providers. For example in case of Twitch, after initial buffering, data is requested every two seconds during the stable phase. Hence, the stable $\vec{X}$ is ideally expected to be in the form of "`200020002000...`" – non-zero values occurring every four data points (4 × 0.5s = 2s interval). Such patterns can be extracted by features such as *zeroFrac i.e.,* fraction of zeros in the window, *maxZeroRun i.e.,* maximum consecutive zeros and so on. They can then be used to train a machine-learning model. However, for different providers, the feature types and their combinations would differ significantly. Hence, instead of handcrafting features from $\vec{X}$, we aim for a classification model that derives higher level features automatically from training data. Note, that unlike the lag values of top peaks in the auto-correlation function (§III) that capture limited properties of the intended signal, $\vec{X}$ is a vector of raw time-series data, inherently capturing all temporal properties of video requests. To automatically derive features of this temporal dimension, we use a very popular time series model called the Long Short Term Memory (LSTM) neural network.

An LSTM maintains a hidden state ($\vec{h_t}$) and a cell state ($\vec{c_t}$), shown as upper and lower channels respectively in Fig. 2(a). The cell state of the LSTM acts like a memory channel, selectively remembering information that will aid in the classification task. In the context of our work, this could be the analysis of periodicity and/or the pattern by which x_is vary over time. The hidden state of the LSTM is an output channel, selectively choosing information from the cell state required for classifying a flow as live or VoD. Fig. 2(a) shows that at epoch t, the input x_t is fed to the LSTM along with the previous hidden state h_{t-1} and cell state c_{t-1}, obtaining current h_t and c_t. In other words, at every epoch, the information of previous steps is combined with the current input. Using this mechanism, an LSTM is able to learn an entire time series sequence with all of its temporal characteristics.

As detailed above, we sequentially feed individual x_i's from $\vec{X}$ into the LSTM (layers=1, dim(h_t) = dim(c_t) = 32×1) to obtain the final hidden state ($\vec{h_{60}}$) which retains all the necessary information for the classification task. We then feed $\vec{h_{60}}$ to a multi-layer perceptron (MLP) to make the prediction, as shown in Fig. 2(b). The final output of the MLP is the posterior probability of the input time-series being an instance of live streaming.

Table III: Monitoring duration impact on the accuracy (*ReCLive* $model$ versus Random Forest).

Provider	Monitoring Duration T=10 sec	T=20 sec	T=30 sec	T=30 sec **[RF]**
Twitch	94.33%	96.13%	**96.82%**	89.50%
YouTube	96.57%	98.28%	**99.80%**	68.93%

Table IV: Confusion matrix of the models.

Provider	**Twitch**		**YouTube**	
Class	Live	VoD	Live	VoD
Live	<u>0.981</u>	0.019	<u>1.000</u>	0.000
VoD	0.117	0.883	0.004	0.996

B. Training and Results

We would like to emphasize that the architecture of the neural network is consistent across providers, thus highlighting the generality of our approach for classifying live and VoD streams. For the remainder of the paper, the combination of the LSTM and MLP is referred to as $model$. During training, $model$ learns to classify streams by reducing prediction error (the binary cross entropy loss function) via back propagation and Adam optimization. The trained $model$ achieved high accuracies (with 80/20 train/test split) across both providers, as shown in Table III. For our baseline, we trained a Random Forest (RF) by lag values of the three highest peaks (using the auto-correlation function described in §III). Comparing the last column (baseline accuracies) with the other three columns (*model* accuracies) in Table III, the performance of $model$ is far superior than that of the baseline classifier. To further understand the impact of monitoring duration on accuracy, we quantify the performance of our $model$ with 10, 20 and 30 seconds of data, as shown in Table III.

Table IV shows the confusion matrix of our 30-sec $model$ across providers. For live flows, we observe almost perfect true positive rates (underlined values) across Twitch and YouTube. However, for VoD flows, we observe a lower performance in Twitch. We believe this is because the Twitch data consists of real-user generated streams (collected from the production network of our university campus) whereas the data for YouTube was selectively collected in a lab environment. In particular, we found certain instances of Twitch VoD in low-bandwidth conditions where the client makes numerous video requests, resulting in an input $\vec{X}$ that is similar to a window of a live stream.

V. Estimating QoE Metrics of Live Video

While the QoE of a live video stream is subjective, we capture it with two major *objective* metrics; video quality

and buffer stalls. Video quality can be measured using: (a) resolution of the video, (b) bitrate (number of bits transferred per sec), and (c) more complex perceptual metrics like VMAF [18]. In this paper, we develop a method to estimate the resolution of the playback video since the ground-truth data is available across both providers. Also, resolution is typically reported (or available to select) in popular live streaming services. In addition to video resolution, we devise a method to detect the presence of buffer stalls which are more likely to occur in case of live streaming (compared to VoD), since a smaller buffer size is maintained on the client to reduce the latency between the producer and the viewer. In what follows, we present our analysis of data collected from the network consisting of audio/video segments versus metrics recorded on the client. Subsequently, we develop methods that estimate video resolution and detect buffer stalls.

A. Network-Level Measurement

To estimate QoE metrics for the live stream, we need to estimate the size of the media segments being fetched. The amount of data downloaded between two consecutive requests reasonably estimates the size of media segments. We refer to this estimate as a *chunk*. Hence, we use the term *segment* for a unit of media requested by the player, while *chunk* denotes a corresponding unit of data observed on the network (demarcated by the request packets). We build upon existing network chunk-detection algorithms [10], [9] to isolate the video chunks fetched by the live player. In short, the algorithm identifies the start of a chunk by an upstream request packet, and aggregates all subsequent downstream packets to "form" the chunk. For each chunk, it extracts the following features: ***requestTime***, *i.e.,* the timestamp of the request packet, ***requestPacketLength***, ***chunkStartTime*** and ***chunkEndTime***, *i.e.,* timestamps of the first and the last downstream packets following the request, and lastly ***chunkPackets*** and ***chunkBytes***, *i.e.,* total count and volume of downstream packets corresponding to the chunk.

During the playback of a live video stream, the chunk telemetry function operates on a per-flow basis in our FlowFetch tool, and exports the above features for every chunk observed on five-tuple flow(s) carrying the video. In addition, as earlier mentioned in §III-B we collect resolution and buffer health metrics reported by the video client. In what follows, we correlate and analyze the chunk data obtained from the network and client metrics to train our models for estimating resolution and detecting the presence of buffer stalls.

B. Estimating Resolution

The resolution of a live video stream indicates the frame size of a video playback – it may also sometimes indicate the rate of frames being played. For example, a resolution of 720p60 means the frame size is 1280×720 pixels while playing 60 frames per sec. For a given fixed duration video segment, the video segment size (and hence our corresponding chunk estimate) usually increases in higher resolutions as more bits need to be packed into the segment.

We analyzed the live video streams played using our tool for both content providers to better understand the distribution of video segment sizes across various resolutions. We also consider four bins of resolution namely Low Definition (LD), Standard Definition (SD), High Definition (HD), and Source (originally uploaded video with no compression, only available in Twitch) – Table V shows the distribution of streams across these bins. The bins are mapped as follows, anything less than 360p is LD, 360p and 480p belong to SD, 720p and beyond belongs to HD. If the client tags a resolution (usually 720p or 1080p) as Source, it is binned into Source. Such binning serves two purposes: (a) it accounts for the similar visual experience for a user in neighboring resolutions and (b) it provides a consistent way to analyze across providers. Fig. 3 shows the distribution of chunk sizes versus resolutions, and will be further explained later in §V-B2. We estimate the resolution in two steps: (a) first, separating chunks corresponding to video segments, and (b) then, developing an ML-based model to map the chunk size to resolution.

1) Separation of video chunks: Network flows corresponding to a live stream can carry chunks of data that correspond to any of video segments, audio segments, or manifest files, and hence the video component needs to be separated out to estimate its resolution. The method to isolate video segments can be developed by analyzing a few streaming sessions by decrypting SSL connections and analyzing the request URLs.

Twitch usually streams both audio and video segments on the same 5-tuple flow for live video streaming, and manifest files are fetched in a separate flow. We observed that audio is encoded in fixed bitrate, and thus its chunk size is consistent ($\approx$ 35 KB). Further, Twitch video chunks of the lowest available bitrate (160p) have a mean of 76 KB. Thus, video chunk identification is fairly simple for Twitch live streams, *i.e.,* all chunks more than 40 KB in size.

YouTube live usually uses multiple TCP/QUIC flows to stream the content consisting of audio and video segments – Youtube operates manifestless. As indicated in Table I, Youtube live operates in two modes, *i.e.,* Low Latency (LL) with 2 sec periodicity of content fetch, and Ultra Low Latency (ULL) with 1 sec periodicity. We found that the audio segments have a fixed bitrate (*i.e.,* size per second is relatively constant) regardless of the latency mode – audio chunk size of $28-34$ KB for the ULL mode, and $56-68$ KB for the LL mode. However, separating out the video chunks is still nontrivial as video chunks of 144p and 240p sometimes tend to be smaller in size than the audio chunks.

To separate the audio chunks, authors of [10] used the *requestPacketLength* as they observed that the audio segment requests were always smaller than the video requests. We used this method for TCP flows, but found it to be inaccurate in case of UDP QUIC flows as the audio segment requests are sometimes larger than video segment requests due to header compression. In addition, QUIC flows pose specific challenges for live video streams. Because of bi-directional stream support available in HTTP/2 + QUIC, a request for a media segment can be sent before the previous segment

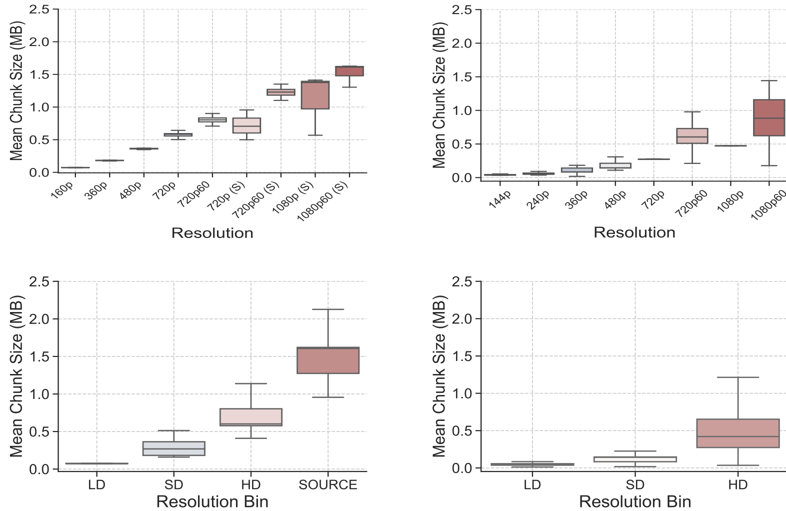


Figure 3: Chunk size versus resolution for Twitch (left), and YouTube (right).

Table V: Resolution distribution of dataset.

Provider	LD	SD	HD	SOURCE
Twitch	17%	32%	34%	17%
YouTube	36%	36%	28%	-

Table VI: Accuracy of resolution prediction.

Provider	Resolution	Resolution bin
Twitch	90.64%	**97.62**%
YouTube	75.17%	**90.08**%

Table VII: Buffer stall prediction results.

	Accuracy	Recall	FP rate
Twitch	90.1%	90.0%	10.3%
YouTube	89.6%	88.2%	14.2%

finishes downloading. Since our chunk telemetry function relies on the request packets to mark the start of chunks, the *chunk* sizes computed by the network telemetry function differ from the actual size of media *segments*. For this reason, we cannot accurately capture individual media segments for YouTube videos delivered over QUIC flows. Thus, while we detect QUIC live video (as the request patterns are still distinguishable), QoE inferencing for YouTube QUIC video streams is beyond the scope of this paper.

2) Analysis and Inference: After identifying the chunks corresponding to the video segments for each provider, we now look at the distribution of chunk sizes across various resolutions at which the video is played. Fig. 3 shows box plots of mean (video) chunk size in MB versus the resolution (*i.e.,* actual value or binned value) in categorical values. Note that the mean chunk size is computed for individual video streams of duration 2-5 minutes. Further, the label *(S)* on the X-axis indicates a Source resolution. Looking at Fig. 3, we make the following observations: (a) video chunk size increases with resolution across both the providers; (b) chunk sizes are less spread in lower resolutions; and (c) chunk sizes of various transcoded resolutions (*i.e.,* not the source resolution) do not overlap much with each other for Twitch, however overlap of neighboring resolutions becomes more evident in YouTube. Such overlaps make it challenging to estimate the resolution.

We use the Random Forest algorithm for mapping chunk sizes to the resolution of playback as it creates overlapping decision boundaries using multiple trees and then uses majority voting to estimate the best possible resolution by learning the distribution from the training data. Using the mean chunk size as an input feature, we trained two models, *i.e.,* one estimating the exact resolution and other estimating the resolution bin. We perform 5-fold cross validation on the dataset with 80/20 train/test split and our results are shown in Table VI.

C. Predicting Buffer Stalls

Buffer stalls occur when the playback buffer is emptied out because the video segments cannot be fetched in time. This QoE metric is especially important for live streams which typically maintain short buffers (4 seconds for Twitch LL and Youtube ULL modes). Network instability even for a few seconds can cause the live buffer to deplete, leading to a stall causing viewer frustration.

To better understand the live buffering mechanism across the two providers, we collect data for live video streams ($\approx 5min$ per session), while using the network conditioner component of our tool to impose synthetic bandwidth caps. We created a commonly occurring situation in a household where in cross traffic (browsing/e-mail etc.) is introduced for a few seconds while a live stream is going on. To do so, the tool starts with a cap of 10Mbps (typical household bandwidth) and then after every 30 seconds caps the download/upload bandwidth at a random value (between 100 Kbps to 2 Mbps) for a duration of 10 seconds (mimicking the congestion due to cross traffic). Live videos being played in the browser are accordingly affected by these bandwidth switches. We found that if videos are played at *Auto* resolution then the clients avoid stalls most of the time by switching to lower resolutions. Therefore, we forced the video streams to play at one of the HD resolutions (1080p or 720p) to gather data of buffer stall events. In total, we collected more than 250 video streams across the three providers. On average, 15% and 6% of the playback time was spent in stall state for Twitch and YouTube.

Fig. 4 shows the dynamics of buffer health for a representative stream in our dataset. We observe that this low latency Twitch video starts with 2 seconds of buffer. It soon encounters the first stall (highlighted by the red bar) around second 25 due to network congestion caused by cross traffic. Following that, the stream linearly increases its buffer to 10 seconds, but experiences stalls a couple more times until second 100. After this point, the buffer value increases to more than 20 seconds. It can be seen that a stall event not only deteriorates user experience but also increases the latency of the live stream as the user is watching content that was recorded at least 20 seconds ago – defeating the purpose of live streaming.

To predict such stalls, our buffer estimator algorithm takes two parameters as input. The first is Seg_{dur}; live video streams

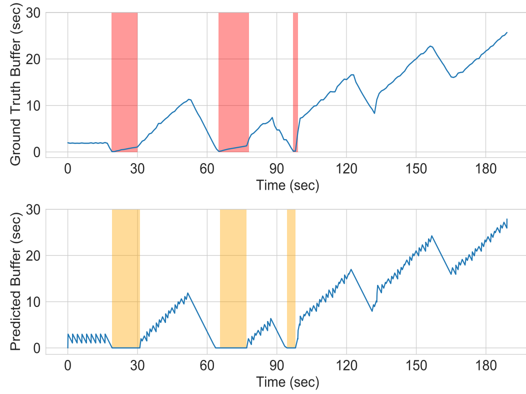


Figure 4: Time-trace of buffer health value: ground-truth predicted, for a sample Twitch stream.

typically encode content into video segments of fixed duration. This duration depends on the playback mode – *e.g.,* YouTube ULL streams have $Seg_{dur} = 1$ sec, while YouTube LL streams have $Seg_{dur} = 2$ sec (§III). We automated this estimation by equating Seg_{dur} to be the median inter-request time (IRT) of video segments in the first window of n seconds (empirically configured to be 20 sec). The second parameter is Buf_{min}; a client typically fetches few video segments (at least one) until a minimum buffer is filled before it begins playback. In the case of Twitch, playback begins after the first segment finishes downloading – hence, $Buf_{min} = 2$ sec (one segment long). However, in the case of YouTube, Buf_{min} seemed to vary between 2-10 seconds. Thus, we conservatively choose the mean value in our dataset – 3 sec for ULL streams and 6 sec for LL streams.

Using the parameters above and the isolated video chunks mentioned above (§V-B1), the buffer estimation algorithm (Algorithm 1) works as follows. At the beginning of a stream, its buffer is initialized at zero and increases by steps of Seg_{dur} at the end of every chunk observed on the network, until it reaches Buf_{min} (Algorithm 1, Lines: 4-7) . For every subsequent video chunk, the buffer value is adjusted by: (a) adding Seg_{dur} and (b) subtracting the time elapsed in the playback since its previous chunk (Algorithm 1, Lines: 4,8).

Our algorithm predicts the current buffer value (in seconds) of the client video player for both providers. To quantify the accuracy of predicting buffer stalls (buffer value = 0), we first divide a given video stream into 5-sec windows and assigned a boolean value (true when there was a stall and false otherwise) to each window. The ground-truth of buffer stalls comes from the playback metrics collected by our tool described in §III-B. Table VII summarizes the performance of predicting buffer stalls across all playback windows. Overall, our algorithm yields about 90% accuracy in predicting the presence of a buffer stall in a 5-sec window. Note that our method tends to underestimate the buffer health in YouTube videos (false positive rate 14.2%) since we choose a conservative Buf_{min} value, leading us to predict stalls even when the buffer value is small but non-zero. We found that in more than 50% of the false-positives, our algorithm underestimates the buffer value by at most 2.3 sec ($\approx$ the duration of a segment).

Algorithm 1: Predict Buffer Stall.

Parameters: Buf_{min}, Seg_{dur}
Data: *Chunks* detected on network $\{c_1, c_2, \ldots, c_n\}$
Output: Estimated buffer health

1 $b \leftarrow 0$ ▷ Tracks current buffer value
2 $t \leftarrow 0$ ▷ Tracks endTime of last chunk
3 **for** *each network chunk c* **do**
4   b += Seg_{dur}
5   **if** $b <= Buf_{min}$ **then**
6     $t = c.EndTime$
7   **continue** ▷ Wait until video starts
8   b –= $c.EndTime - t$ ▷ Decrement buffer
9   **if** $b <= 0$ **then**
10     $b = 0$ ▷ Stall detected
11   $t = c.EndTime$

VI. Conclusion

Live video streaming is a rapidly growing and ISPs today lack tools to infer its QoE metrics in their network as existing DPI-based solutions fall short in detecting and monitoring live streams. In this paper, we present *ReCLive*: an ML-based system to distinguish live streams from VoD streams using media requests patterns and to infer QoE in terms of resolution and buffer stall events for the detected live streams using chunk attributes extracted from the network flows.

References

[1] Conviva, “Annual State of the Streaming TV Industry,” https://bit.ly/2NgqvRt, 2018.
[2] “Live streaming: 2020 trends,” https://bit.ly/2NFCbzT, May 2020.
[3] T. Zhang *et al.*, “Modeling and analyzing live streaming performance,” in *2020 IEEE/ACM IWQoS*, 2020.
[4] A. Ahmed *et al.*, “Suffering from buffering? Detecting QoE impairments in live video streams,” in *Proc. IEEE ICNP*, Toronto, Canada, Oct 2017.
[5] T. Guarnieri *et al.*, “Characterizing QoE in Large-Scale Live Streaming,” in *Proc. IEEE GLOBECOM*, Singapore, Singapore, Dec 2017.
[6] F. Pacheco *et al.*, “Towards the deployment of machine learning solutions in network traffic classification: a systematic survey,” *IEEE Communications Surveys & Tutorials*, vol. 21, pp. 1988–2014, 2018.
[7] H. Habibi Gharakheili *et al.*, “iTeleScope: Softwarized Network Middle-Box for Real-Time Video Telemetry and Classification,” *IEEE TNSM*, vol. 16, no. 3, pp. 1071–1085, Sep. 2019.
[8] W. Wang *et al.*, “End-to-End Encrypted Traffic Classification with One-Dimensional Convolution Neural Networks,” in *Proc IEEE ISI*, Beijing, China, Jul 2017.
[9] T. Mangla *et al.*, “eMIMIC: estimating http-based video QoE metrics from encrypted network traffic,” in *Proc. IEEE/IFIP TMA*, Jun 2018.
[10] C. Gutterman *et al.*, “Requet: Real-Time QoE Detection for Encrypted YouTube Traffic,” in *Proc. ACM MMSys*, Amherst, MA, USA, Jun 2019.
[11] F. Bronzino *et al.*, “Inferring Streaming Video Quality from Encrypted Traffic,” *Proc. ACM POMACS*, Aug 2019.
[12] S. Madanapalli *et al.*, “Inferring Netflix User Experience from Broadband Network Measurement,” in *Proc. IEEE TMA*, Paris, France, 2019.
[13] M. Shen *et al.*, “Deepqoe: Real-time measurement of video qoe from encrypted traffic with deep learning,” in *Proc. IEEE/ACM IWQoS*, 2020.
[14] “Twitch Engineering: An Introduction and Overview,” http://bit.ly/2sb86hv, 2018.
[15] K. Pires and G. Simon, “Youtube live and twitch: a tour of user-generated live streaming systems,” in *Proc. ACM MMSys*, 2015.
[16] C. Zhang *et al.*, “On Crowdsourced Interactive Live Streaming: A Twitch.Tv-Based Measurement Study,” in *Proc. ACM NOSSDAV*, 2015.
[17] “Selenium Browser Automation,” https://www.selenium.dev/, July 2020.
[18] Z. Li *et al.*, “Vmaf: The journey continues,” http://bit.ly/2Nad05K, 2018.