Real-time Video Quality of Experience Monitoring for HTTPS and QUIC

M. Hammad Mazhar, Zubair Shafiq
The University of Iowa
{muhammadhammad-mazhar, zubair-shafiq}@uiowa.edu

***Abstract*—The widespread deployment of end-to-end encryption protocols such as HTTPS and QUIC has reduced the visibility for operators into traffic on their networks. Network operators need the visibility to monitor and mitigate Quality of Experience (QoE) impairments in popular applications such as video streaming. To address this problem, we propose a machine learning based approach to monitor QoE metrics for encrypted video traffic. We leverage network and transport layer information as features to train machine learning classifiers for inferring video QoE metrics such as startup delay and rebuffering events. Using our proposed approach, network operators can detect and react to encrypted video QoE impairments in real-time. We evaluate our approach for YouTube adaptive video streams using HTTPS and QUIC. The experimental evaluations show that our approach achieves up to 90% classification accuracy for HTTPS and up to 85% classification accuracy for QUIC.**

I. Introduction

Internet traffic encryption is rapidly increasing due to security and privacy concerns. According to Sandvine [26], 70% of the global Internet traffic is already encrypted. HTTPS deployment is accelerating, driven in large part by video content providers such as YouTube and Netflix. While encryption efforts are mainly focused at the application layer (e.g., TLS), recent attempts have focused on deploying end-to-end encryption at the transport layer (e.g., QUIC) as well. It is safe to assume that the encryption trend will continue and perhaps further accelerate in the future. While end-to-end encryption is essential for security and privacy, it fundamentally limits the scope of Quality of Experience (QoE) and network management by network operators.

A majority of Internet traffic now comprises of video. Almost three-quarters of all Internet traffic was video in 2016 and the volume of video traffic is expected to grow fourfold from 2016 to 2021 [1]. Network operators employ a variety of network management mechanisms to satisfy strict QoE requirements of modern video streaming applications. Specifically, since the licensed radio spectrum is an expensive and scarce resource, mobile network operators employ Deep Packet Inspection (DPI) based network management techniques for bandwidth optimization and to efficiently share the radio spectrum among users [2]. However, encryption breaks the network management mechanisms currently used by network operators to monitor and optimize QoE. Prior work has tried to address the problem of QoE monitoring for encrypted traffic by developing various heuristics and machine learning based approaches [10], [25]. Prior research on encrypted video QoE monitoring has two key limitations. First, they approach encrypted video QoE monitoring as a onetime post hoc inference problem. However, network operators require finer-grained real-time QoE monitoring for adaptive resource allocation on the fly. Real-time video QoE inference allows network operators to detect and mitigate QoE impairments as they impact user experience. Second, prior work has only considered QoE monitoring for encrypted videos streamed over HTTPS. However, the use of new encrypted transport protocols such as QUIC, which runs over UDP, is becoming prevalent for video streaming. For example, YouTube is increasingly serving video traffic over QUIC [5]. In this paper, we present a machine learning based approach to monitor QoE metrics in real-time for videos streamed over HTTPS and QUIC.

We summarize our key contributions and results as follows.

- We develop a comprehensive set of features based on network and transport layer header information. For transport layer, we design features based on TCP flags, retransmissions, goodput, etc. It is noteworthy that network operators cannot observe TCP-like transport layer features for QUIC. Therefore, we further develop network layer features based on inter-arrival time, packet sizes, packet/byte counts, throughput, etc.
- We implement a supervised machine learning based approach to infer QoE metrics such as startup delay, rebuffering events, and video quality in real-time for encrypted video streams. Our machine learning based approach is capable of inferring QoE metrics continuously in 10 second time windows as compared to one-time post hoc inference.
- We evaluate our machine learning based approach for inferring QoE metrics on YouTube using HTTPS and QUIC. Our trained machine learning models achieve up to 90% accuracy for HTTPS and up to 85% for QUIC.

II. Related Work

Our work builds on a long line of research on traffic classification and recent advancements in QoE monitoring. In this section, we summarize prior work and highlight differences from our proposed approach. We categorize related research into traffic classification, encrypted traffic analysis, QoE monitoring, and QoE monitoring for encrypted traffic.

Machine Learning Based Traffic Classification. Network operators employ a variety of network management techniques to implement Quality of Service (QoS) aware adaptive resource allocation [20]. For example, network operators typically want to prioritize VoIP flows over bulk transfer flows. To this end, network operators identify suitable QoS Class Indicator (QCI) for traffic flows [12] using different heuristics [29] and machine learning [24] based approaches. In a seminal work, Moore and Zuev [22] used a Naive Bayes classifier on traffic features such as ports, packet inter-arrival time statistics, and payload sizes. Bernaille et al. [8] used a similar machine learning approach to identify application types such as FTP, SMTP, and HTTPS within the first few packets of a TCP flow. Prior work on traffic classification typically categorizes encrypted traffic in a separate category (e.g., HTTPS). As the share of encrypted Internet traffic has increased over the last few years [26], network operators are interested in finer-grained encrypted traffic analysis.

Encrypted Traffic Analysis. Researchers have used statistical and machine learning based approaches to analyze the content of encrypted traffic [21], [31]. White et al. [30] used Hidden Markov Models to extract transcripts of encrypted VoIP transmissions. Schuster et al. [28] used convolutional neural networks with traffic burst sizes as features to identify YouTube, Netflix, Amazon, and Vimeo videos. In contrast to prior work on encrypted content analysis, our focus is on analyzing application layer QoE metrics for encrypted traffic.

Monitoring Video QoE. For non-encrypted traffic, network operators relied on Deep Packet Inspection (DPI) to access packet header information for monitoring QoE metrics such as startup delay and rebuffering events for streaming video. Prometheus [4] used LASSO regression on passive measurements of TCP flag counts, TCP throughput statistics, network type (2G, 3G), and HTTP request counts to estimate QoE metrics for mobile video streaming. YouQMon [9] analyzed video frame offsets from HTTP headers to identify rebuffering events for YouTube videos. QoE metrics have differing impacts on overall user experience. Mean Opinion Score (MOS) represents a user's assessment of experience. Prior work relied on user studies to map QoE metrics to user-perceived MOS scores. Katsarakis et al. [18] used statistical signatures to causally link QoE (e.g., startup delay, rebuffering events) and QoS (e.g., throughput, retransmissions) metrics to MOS scores. ITU's P.1203 [15] recommendation also provides a set of equations to map QoE metrics to MOS scores. Since these methods require access to application layer information to monitor QoE metrics, they are unusable for encrypted traffic.

Monitoring Video QoE for Encrypted Traffic. The increased popularity of video content [1] and usage of encryption by popular video content providers [13], [23] has spurred new research on encrypted video QoE monitoring. Recent work has focused on inferring video QoE metrics using network and transport layer information that is readily available to network operators for both encrypted and non-encrypted traffic. Dimopoulos et al. [10] used a Random Forest classifier with features such as packet loss, RTT, and bandwidth-delay product to infer the average quality and rebuffering events in YouTube streams. Orsolic et al. [25] tried different machine learning approaches with features such as inter-arrival time, throughput, and packet sizes to infer QoE classes (e.g., {high, medium, low}). Prior work on encrypted video QoE monitoring has focused on inferring QoE metrics over the whole video session. In contrast, we aim to infer QoE metrics in real-time which allows network operators to react to QoE impairments much faster. Moreover, prior work has focused on encrypted video QoE monitoring for HTTPS running on TCP. However, video content providers are increasingly using encrypted transport protocols implemented on top of UDP [5], which hides transport layer information such as retransmissions and throughput from network operators. We aim to infer QoE metrics for videos using QUIC, which is a new encrypted UDP-based transport protocol.

III. Background

Adaptive Video Streaming. Over-The-Top (OTT) video providers primarily use HTTP Adaptive Streaming (HAS) to serve content to end-users. The video player downloads the video by requesting a sequence of fixed duration (generally 4 seconds) video segments from the server. The video player stores the downloaded video segments in a local buffer, which allows the video player to continue smooth playback while it is waiting for video segments to arrive. Each video segment is available at multiple *quality* levels typically ranging from 144p to 1080p. The video player needs to specify the quality in its request for each video segment. The video quality adaptation logic, implemented in the video player, decides the requested quality based on factors such as the available throughput [17] and buffer status [14]. The video streaming process consists of two phases: *startup* and *steady state*. In the startup phase, the video player waits for the buffer to be filled up to a certain threshold (typically 20 seconds) before starting the playback. The delay experienced in the startup phase of video streaming is known as *startup delay*. In the steady state phase, the playback uses the downloaded segments from buffer as more segments are requested to replenish the buffer. While waiting for new segments, if the buffer is completely drained, the player has to pause the playback process. This interruption is known as a *rebuffering event*. The playback process then resumes when the buffer is sufficiently replenished.

QoE and User Engagement. The video playback measures such as startup delay, rebuffering events, and quality are important QoE metrics that can significantly impact user engagement. Krishnan and Sitaraman [19] showed the causal relationships between QoE metrics and user engagement. For example, an increase of 1 second in startup delay led to a 5.8% increase in user abandonments. Dobrian et al. [11] showed a negative correlation between the rate of rebuffering events (number of rebuffering events per second) and play time. At a high level, users abandon when the playback process takes too long to start or it is frequently interrupted. Therefore, all stakeholders in the video delivery ecosystem, including content providers and network operators, need to monitor these QoE metrics in order to mitigate QoE impairments and maximize user satisfaction.

QoE Monitoring by Network Operators. Video content providers can easily monitor QoE impairments because they have direct access to QoE metrics that are measured at the video player. QoE impairments can occur due to a variety of issues that are in control of the content provider (e.g., overloaded server) or the network operator (e.g., network congestion). To mitigate QoE impairments due to network issues, network operators need to be able to measure and monitor QoE metrics. Network operators can accomplish this by conducting DPI of application layer header information [27] However, the increasing deployment of end-to-end encryption by video content providers prevents network operators from using DPI to monitor QoE metrics. Therefore, network operators now need to infer QoE metrics only from information visible to them (at network and transport layers) for encrypted traffic.

Problem Statement. Network operators want to infer QoE metrics for ongoing video streams in real-time. We assume that a network operator is capable of identifying traffic streams of a video content provider. To identify traffic streams of a video content provider, network operators can (a) identify IP addresses used by video content providers, (b) inspect DNS responses, (c) analyze Server Name Indication (SNI) in TLS handshakes, or (d) employ machine learning techniques [7], [22]. A traffic stream can comprise of one or multiple sequential or parallel traffic flows (i.e. identified by a unique 5-tuple of the source address, destination address, source port, destination port, and the transport protocol type). The network operator collects network and transport header information for both downstream and upstream traffic. However, application header information is inaccessible due to the use of end-to-end encryption protocols such as HTTPS or QUIC. For each traffic stream, the network operator is capable of storing packet information in a moving time window of 10 seconds with 5 second shifts. The exact duration of the time window and shift is selected arbitrarily and can be changed if desired. Note that when the time window is shifted, data for only the last 5 seconds of the previous time window is retained. Given this information, the network operator wants to infer QoE metrics (startup delay, rebuffering events, and video quality) for the video stream in real-time. Such real-time monitoring can enable network operators to react to QoE impairments by adaptive resource allocation (e.g. SONs [3]).

IV. Proposed Approach

We propose a machine learning approach to infer QoE metrics for encrypted video traffic features. The proposed features incorporate a wide range of traffic characteristics derived from network- and transport-layer header information.

Overview. Figure 1 provides an overview of our machine learning based encrypted video QoE inference approach. In the training phase, a network operator plays a set of videos and records their packet traces in the network as well as ground truth video QoE metrics directly from the video player. For each packet trace, the network operator extracts a comprehensive set of features from the network and transport layer headers. Given a sufficient set of videos played under diverse network conditions, the network operator then trains supervised machine learning models for different QoE metrics. In the testing phase, the network operator extracts the feature set for a test video from its packet trace and uses the machine learning models to infer QoE metrics. Network operators can use the inferred QoE metrics to detect and mitigate QoE impairments using different network management techniques.

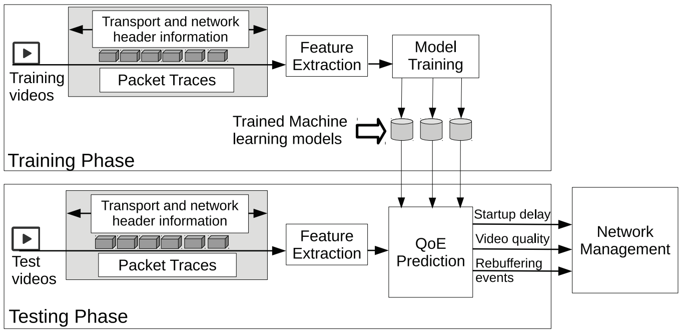


Fig. 1. Overview of proposed approach

	Network-layer	Transport-layer
Window	Byte Counts Packet Counts Throughput Idle Time	*TCP Flag Counts Out-of-order bytes/packets TCP Goodput ‡Retranmission ratio ***Starting bytes-in-flight ***Ending bytes-in-flight
†Packet	Packet Inter-arrival times Bytes per packet	Retransmissions per packet Receive Window **RTT Bytes-in-Flight

(*) We consider SYN, ACK, FIN, URG, PSH and RST flags.
(†) Computed statistics are used as individual features: mean, min, max, median, standard deviation, skewness, kurtosis.
(‡) Ratio of 0, 1, 2 and >2 retransmissions are considered.
(**) Only for upstream traffic.
(***) Only for real-time QoE inference.

TABLE I
LIST OF OVERALL FEATURES CONSIDERED

Features. Table I provides a summary of network and transport layer features that we propose to extract from packet traces. Network layer features are derived from information in IP headers. Transport layer features are derived from information in TCP/UDP headers. We compute every feature separately for upstream and downstream traffic. Our proposed features can be further divided into window and packet based features. Window based features are calculated by aggregating information for all packets that arrive in a time window. Packet based features are calculated by analyzing individual packets. For packet based features, we compute summary statistics such as mean, median, max, min, standard deviation, kurtosis, and skewness. It is noteworthy that features for startup delay are extracted only once in the first time window. In contrast, features for rebuffering events and video quality are extracted in a continuous fashion for every time window. Moreover, since our 10 second time window operates in 5 second shifts, these features are derived separately for the first and last half of each time window. Overall, we extract a set of 109 features to infer startup delay and 226 features for quality and rebuffering events. We next discuss our proposed features.

- *Network layer window features* include packet counts, byte counts, and throughput. Idle time is derived by first splitting the time window into 100 millisecond bins and assigning packets to bins by rounding their arrival time to 100 milliseconds. The number of empty 100 millisecond bins (with no packets) constitutes idle time.
- *Network layer packet features* include packet inter-arrival times and packet size in terms of bytes per packet. Note that network layer features do not take into account retransmissions due to packet loss.
- *Transport layer window features* include flag counts for TCP connections, out-of-order bytes/packets (ignoring retransmissions), TCP goodput (i.e throughput minus retransmissions), and retransmission ratios. Starting and ending bytes-in-flight correspond to number of bytes which have not been acknowledged at the start and at end of the current time window.
- *Transport layer packet features* include retransmissions per packet and advertised receive window. Round Trip Time (RTT) is calculated only for upstream traffic by using packet sequence numbers and their corresponding acknowledgments, ignoring retransmitted packets. Note that RTT is dependent on the vantage point (i.e., RTT at the last mile would differ from RTT at the core network.) Bytes-in-flight are bytes which have been sent but not acknowledged, and are calculated based on the highest TCP sequence number and acknowledgment seen. We use sequence numbers from packets from the video server and acknowledgments from the user for downstream and vice versa for upstream, ignoring retransmissions.

Network layer features. The set of network layer features are derived only using IP packet header information. As we discuss next, many of these features have significant correlations with different QoE metrics. Figures 2, 3, and 4 plot the conditional Cumulative Distribution Functions (CDFs) of different network layer features for different QoE metrics.

Startup Delay. To plot conditional distributions for startup delay, we label sessions with up to 3.3 seconds startup delay as Started and the remaining as Not Started. We expect sessions with smaller startup delay to have downloaded more data in the *startup* phase of the video streaming process. Figure 2(a) shows that Started sessions download a median of 1.5 more megabytes (MB) from server than Not Started sessions in the first 3.3 seconds. The slower download process is also reflected in the distribution of average packet inter-arrival time for downstream traffic plotted in Figure 2(b). We note that 20% Not Started sessions and only 3% Started sessions have average packet inter-arrival time larger than 18 milliseconds. Overall, we also observe this pattern in the distribution of downstream throughput in Figure 2(c). We note that median downstream throughput of Started sessions is 18 megabits-per-second (Mbps) more than that of Not Started sessions.

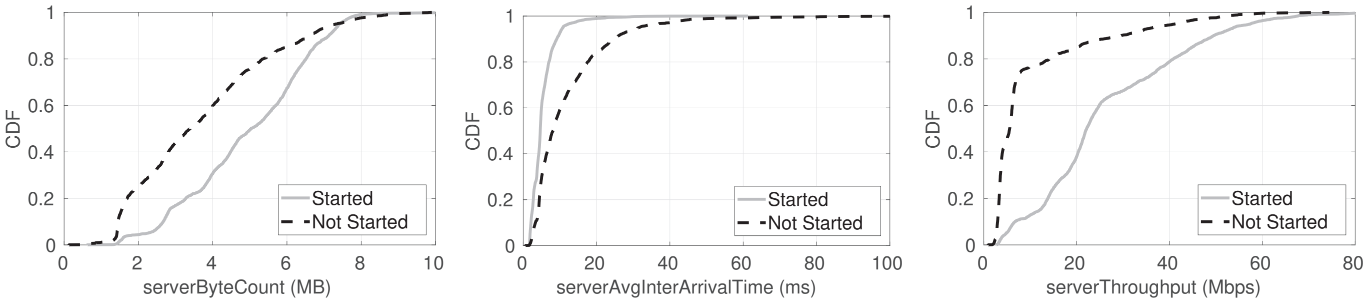
Rebuffering Events. To plot conditional distributions for rebuffering events, we label 10 second time windows with at least one rebuffering event as Rebuffering and the remaining as No Rebuffering. Features are calculated separately for the first and last half of every time window. We expect time windows with rebuffering events to have downloaded less data than those without rebuffering events. Figure 3(a) shows that 60% Rebuffering time windows download no data in the first half of time windows as compared to less than 20% for No Rebuffering time windows. This pattern is also evident for downstream throughput in the first half of time windows. Figure 3(b) shows that more than 90% Rebuffering time windows get less than 10 Mbps downstream throughput in the first half of time windows as compared to only 50% No Rebuffering time windows. Figure 3(c) shows the same pattern for downstream throughput in the second half of time windows.

Video Quality. To plot conditional distributions for video quality, we label 10 second time windows as High quality if the average resolution is higher than 480p and the remaining as Low quality. We expect more data to be downloaded for High quality time windows than Low quality time windows because higher resolution videos are encoded at much higher bitrates. Figures 4(a) and 4(b) plot distributions for downstream throughput in the first and last half of time windows, respectively. We note that more than 90% Low quality and approximately 50% High quality time windows get less than 10 Mbps downstream throughput. We also expect lower quality video streaming to send less data upstream due to less signaling [6] and acknowledgments for downloaded data. Figure 4(c) shows that 45% Low quality and 30% for High quality time windows have negligible upstream throughput.

Transport layer features. The set of transport layer features are derived using TCP packet header information. Note that these features are unavailable for QUIC, which runs over UDP. Figures 5, 6, and 7 plot the conditional CDFs of different transport layer features for different QoE metrics.

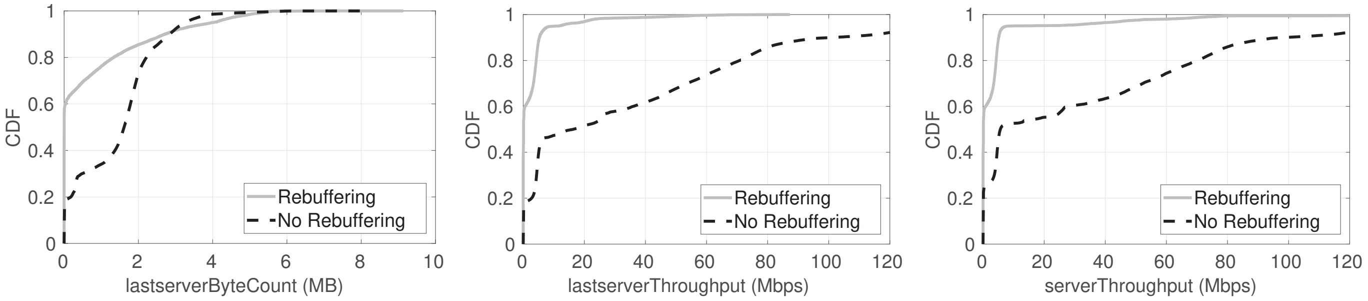
Startup Delay. Figure 5(a) plots the distribution of downstream TCP goodput (throughput excluding retransmissions) for Started and Not Started sessions in the first 3.3 seconds. We expect video streaming sessions with low startup delay to have higher TCP goodput than videos with high startup delay. We observe that 85% Not Started sessions have less than 20 Mbps downstream goodput. In contrast, only 40% Started sessions have less than 20 Mbps goodput. The difference in goodput is explained, in part, by more frequent retransmissions. Figure 5(b) shows that 95% Started sessions have an average of at most one retransmission per downstream packet as compared to less than 50% for Not Started videos. Low average packet retransmissions are also correlated with packet retransmission ratio. We plot the distribution of ratio of downstream packets that are never retransmitted in Figure 5(c). More than 90% Started sessions have 80% packets on average with zero retransmissions. In contrast, only 50% Not Started sessions achieved the 80% mark.

Rebuffering Events. We expect time windows with rebuffering events to have more unacknowledged bytes than those with no rebuffering. Figure 6(a) shows that 18% Rebuffering time windows have more than 50 MB of bytes-in-flight on average on the downstream. In comparison, less than 5% No



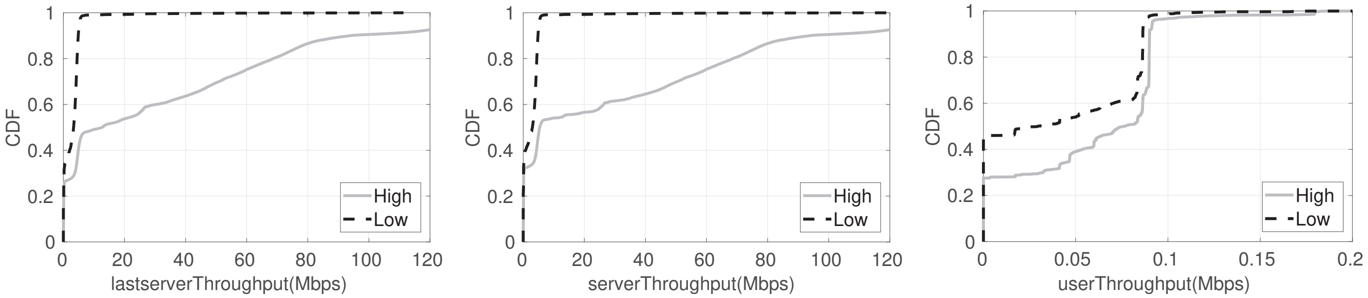
(a) downstream byte count (b) downstream average packet inter-arrival time (c) downstream throughput

Fig. 2. Network-layer feature distributions for startup delay of 3.3 seconds.



(a) downstream byte counts in first half of time windows (b) downstream throughput in first half of time windows (c) downstream throughput in second half of time windows

Fig. 3. Network-layer feature distributions for rebuffering events.



(a) downstream throughput in first half of time windows (b) downstream throughput in second half of time windows (c) downstream throughput in second half of time windows

Fig. 4. Network-layer feature distributions for video quality.

Rebuffering time windows have more than 50 MB bytes-in-flight. With more unacknowledged bytes, we expect more frequent downstream retransmissions for Rebuffering time windows. Figure 6(b) shows that 75% No Rebuffering time windows have zero retransmissions per packet downstream as compared to 62% for Rebuffering time windows. Finally, we expect higher downstream goodput for No Rebuffering time windows as compared to Rebuffering time windows. Figure 6(c) shows that 50% No Rebuffering time windows have more than 5 Mbps goodput as compared to less than 10% for No Rebuffering time windows.

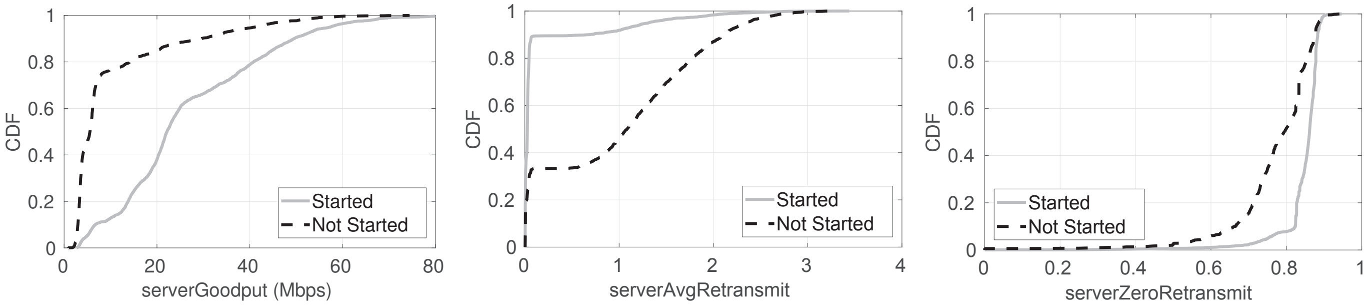
Video Quality. For video quality, we again expect fewer unacknowledged bytes for time windows with High quality. Figure 7(a) plots downstream minimum bytes-in-flight for the second half of time windows. We note that 90% High quality time windows have zero bytes-in-flight at some point in the second half of time windows as compared to approximately 80% of Low quality windows. We again expect more unacknowledged bytes to lead to more frequent retransmissions. Figure 7(b) shows that 80% High quality time windows have zero average retransmissions per downstream packets. In contrast, only 40% Low quality windows have zero average retransmissions per downstream packet. As shown in Figure 7(c), retransmissions in turn impact downstream goodput. 55% High quality time windows get less than 5 Mbps goodput as compared to 99% of Low quality time windows.

Classification. To infer video QoE metrics using network and transport layer features, we use decision tree based machine learning classification algorithms that are known to outperform other classification algorithms for similar problems [10], [25] and are also human interpretable. We train our machine learning classifiers using J48 implementation of the C4.5 decision tree algorithm. We also use AdaBoost ensemble meta-classification approach to reduce misclassifications.

V. Results & Discussions

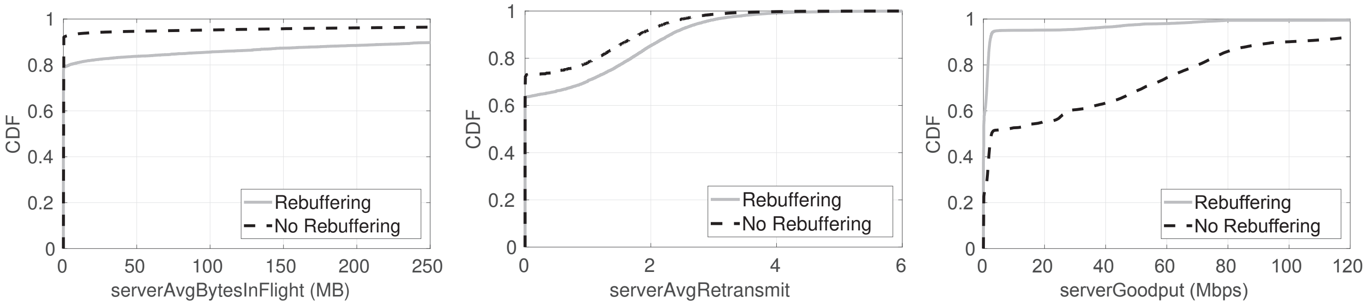
A. Data

We use a controlled lab testbed to collect data for evaluating our proposed approach. We automatically play YouTube videos using Selenium WebDriver in Google Chrome running on Ubuntu 16.04 OS. A set of pre-selected videos are played



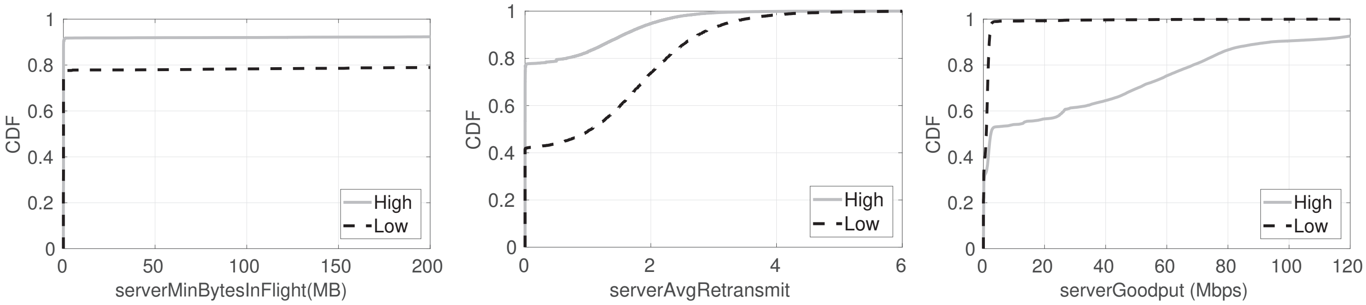
(a) downstream TCP goodput (b) downstream average retransmissions per packet (c) downstream zero retransmission ratio

Fig. 5. Transport-layer feature distributions for startup delay of 3.3 seconds.



(a) downstream average bytes-in-flight for last half of time windows (b) downstream average retransmissions per packet for last half of time windows (c) downstream TCP goodput for last half of time windows

Fig. 6. Transport-layer feature distributions for rebuffering events.



(a) downstream minimum bytes-in-flight for last half of time windows (b) downstream average retransmissions per packet for last half of time windows (c) downstream TCP goodput for last half of time windows

Fig. 7. Transport-layer feature distributions for video quality.

on fresh instances of Google Chrome in a sequential manner. These videos include movie trailers, short interviews, and sport highlights. The videos are available in quality levels ranging from 144p to 1080p. The average video duration is approximately 2 minutes. Each video streaming session elapses up to 4 minutes to allow for delays due to page load, startup delay, and rebuffering events.

To collect ground truth for video QoE metrics, we use YouTube's IFrame API [32] to capture playback events reported by the video player. The API outputs a set of values which indicate player state (not started, paused, playing, completed, buffering) using the *getPlayerState()* function. The API also provides functions for accessing information about play time, buffer status, and video quality. We use `tcpdump` to extract our proposed features from the network and transport layer headers during video playback.

To emulate real-world dynamic network conditions, we use network emulation tools `netem` and `tc`. Using these tools in Linux, we can emulate different network conditions in terms of bandwidth, delay, and packet loss. We randomly set network conditions before every video playback session. We further fluctuate network conditions every 10 seconds during the video playback to allow YouTube's video quality adaptation logic to adapt to changing network conditions. We use experimental flags supported by Google Chrome to switch between QUIC and TCP as the underlying transport protocol. The experimental flags can be accessed using *chrome://flags* and QUIC can be enabled by selecting the *Experimental QUIC protocol* option.

We collect YouTube video streaming data over the duration of four months (starting March 2017) using our controlled lab testbed. Over this period, we collect data for 5,488 sessions running over QUIC and 5,375 sessions running over standard HTTPS using TCP.

We define startup delay as a binary variable for machine learning based classification. Specifically, we define startup delay for a video streaming session as a binary indicator of it being higher or lower than k seconds during the *startup* phase of video streaming. For this work, we consider the values of k at 3.3, 6.6, and 10 seconds. Based on this definition, we label videos as Started if the startup delay is below k, Not Started otherwise. We note that the class imbalance between Started and Not Started classes increases for larger values of k.

We also define rebuffering events and video quality as binary variables for machine learning based classification. In contrast to startup delay, rebuffering events and video quality are quantified in a continuous fashion for 10 second time windows during the *steady state* phase of video streaming. More specifically, we label every 10 second time window as No Rebuffering or Rebuffering (i.e., at least one rebuffering event) based on the frequency of rebuffering events in the time window. Similarly, we label every 10 second time window as Low or High quality based on the average video resolution in the time window. More specifically, we label a time window as High quality if the average resolution is higher than 480p and Low quality otherwise. Note that we try to extract features (from network and transport layer headers) and ground truth (from video player) for each 10 second time window. Since videos are sometimes completely buffered before playback completes, we observe some time windows where we do not capture any packet traces. Since we cannot extract our features, we discard these time windows. We note that class instances for rebuffering events are imbalanced by a factor of around 3.2:1 in favor of the Rebuffering class for HTTPS. For QUIC, the class imbalance is around 1.6:1 in favor of the Non Rebuffering class. We define binary QoE metrics for problem simplification and to ensure sufficient data for model training. An operator may choose to use more granular QoE metrics such as quality levels of 144p, 240p, 480p, 720p, and 1080p.

B. Classification Results

We evaluate our machine learning learning classifiers using the standard 10-fold cross-validation. We report the classification results in terms of both *precision* and *recall* separately for each class.

For startup delay, we train models that classify video sessions based on whether or not the playback process started in the first k seconds. Tables I(a), II(a), and III(a) show the classification results for YouTube videos played using standard HTTPS with startup delays of $k = 3.3$, 6.6, and 10 seconds, respectively. We observe that the classifiers achieve at least 75% precision and recall on average for different values of k. We note the highest precision and recall values of 81.5% for $k = 3.3$. It is noteworthy that precision and recall decline as k increases. We surmise that degradation in classification performance happens for larger k value due to increased class imbalance as well as due to fewer rare class instances available for training. Tables I(b), II(b), and III(b) show the classification results for YouTube videos player using QUIC. We see similar trends for QUIC, with at least 77% precision and recall on average for different values of k. For $k = 3.3$, the classifier for QUIC achieves 3.4% more precision and 3.3% more recall than the classifier for HTTPS. For $k = 6.6$, there is negligible difference in performance. For $k = 10$, the classifier for QUIC performs around 3% better than the classifier for HTTPS.

(a) HTTPS

Predicted Actual	Started	Not Started	**Precision**	**Recall**
Started	2824	505	82.8%	87.5%
Not Started	585	1561	72.7%	79.4%
		Weighted Average	81.5%	81.5%

(b) QUIC

Predicted Actual	Started	Not Started	**Precision**	**Recall**
Started	1122	408	72.5%	73.3%
Not Started	426	3532	89.6%	89.2%
		Weighted Average	84.9%	84.8%

TABLE II

CONFUSION MATRIX AND PERFORMANCE METRICS FOR CLASSIFICATION OF INITIAL DELAY FOR $k = 3.3$

(a) HTTPS

Predicted Actual	Started	Not Started	**Precision**	**Recall**
Started	3178	467	81.4%	87.2%
Not Started	725	1005	68.3%	58.1%
		Weighted Average	79.9%	80.0%

(b) QUIC

Predicted Actual	Started	Not Started	**Precision**	**Recall**
Started	1826	533	74.5%	77.7%
Not Started	624	2505	82.5%	80.1%
		Weighted Average	79.0%	78.9%

TABLE III

CONFUSION MATRIX AND PERFORMANCE METRICS FOR CLASSIFICATION OF INITIAL DELAY FOR $k = 6.6$

(a) HTTPS

Predicted Actual	Started	Not Started	**Precision**	**Recall**
Started	3413	476	81.6%	87.8%
Not Started	771	715	48.1%	60.0%
		Weighted Average	74.8%	75.8%

(b) QUIC

Predicted Actual	Started	Not Started	**Precision**	**Recall**
Started	2038	560	75.3%	78.4%
Not Started	669	2221	79.9%	76.9%
		Weighted Average	77.7%	77.6%

TABLE IV

CONFUSION MATRIX AND PERFORMANCE METRICS FOR CLASSIFICATION OF INITIAL DELAY FOR $k = 10$

Tables IV(a) and IV(b) report classification results for rebuffering events for HTTPS and QUIC, respectively. Our trained classifiers achieve around 90% precision and recall for YouTube over HTTPS and around 80% for YouTube over QUIC. As expected, we note that the dominant class in terms of number of instances has better classification performance than the non-dominant class for both protocols. We surmise that better classification performance for HTTPS as compared to QUIC, despite more class imbalance, reflects the benefit of using transport layer features in inferring rebuffering events.

Tables V(a) and V(b) report classification results for video quality for YouTube over HTTPS and QUIC, respectively.

(a) HTTPS

Predicted Actual	No Rebuffering	Rebuffering	**Precision**	**Recall**
No Rebuffering	56854	3267	92.3%	94.6%
Rebuffering	4737	14118	74.9%	81.2%
		Weighted Average	89.7%	89.9%

(b) QUIC

Predicted Actual	No Rebuffering	Rebuffering	**Precision**	**Recall**
No Rebuffering	32909	13588	74%	70.8%
Rebuffering	11576	63657	82.4%	84.6%
		Weighted Average	79.2%	79.3%

TABLE V
CONFUSION MATRICES AND PERFORMANCE METRICS FOR CLASSIFICATION OF REBUFFERING EVENTS

Our trained classifier achieves above 85% precision and recall for YouTube over HTTPS and around 72% for YouTube over QUIC. We note that our trained classifiers perform significantly worse for Low quality, which is the rare class, as compared to High quality. For HTTPS, the precision of the Low quality class is 25.1% lower than that for the High quality class. For QUIC, the precision of the High quality class is 6.2% lower than that for the Low quality class. These class-specific differences in classification performance can be explained by class imbalance. For example, we note that class imbalance is roughly 4:1 in favor of High class for HTTPS. Consequently, the classifier's performance is much better for High class.

(a) HTTPS

Predicted Actual	High	Low	**Precision**	**Recall**
High	59291	4492	89.7%	93.0%
Low	6782	8215	64.6%	54.8%
		Weighted Average	85.0%	85.7%

(b) QUIC

Predicted Actual	High	Low	**Precision**	**Recall**
High	37740	16632	68.9%	69.4%
Low	17013	50144	75.1%	74.7%
		Weighted Average	72.3%	72.3%

TABLE VI
CONFUSION MATRICES AND PERFORMANCE METRICS FOR CLASSIFICATION OF VIDEO QUALITY

C. Insights

The trained machine learning models can provide us insights into different network and transport layer features that have significant impact on video QoE. Below, we visualize two example decision tree models.

Figure 8 visualizes a pruned decision tree model that is trained to infer rebuffering events for YouTube streaming on HTTPS. To prune the decision tree, we apply a 10% confidence factor and impose a limit of at least 1800 instances at leaf nodes. The pruned decision tree model is approximately 4% less accurate (in terms of precision and recall) as compared to the unpruned decision tree. The upstream maximum bytes per packet is the decision tree's root node, which represents the most informative feature. The feature split of the root node shows that time windows with at least one upstream packet larger than 1099 bytes are classified as No Rebuffering. Further inspection of upstream packets revealed that this is typical packet size of HTTP GET requests by the video player to download video segments. We also note that time windows with at least 1 upstream TCP SYN flag are likely to be labeled as Rebuffering. Upstream SYN flags may indicate new connection attempts due to disruptions, leading to rebuffering events, in the video download process.

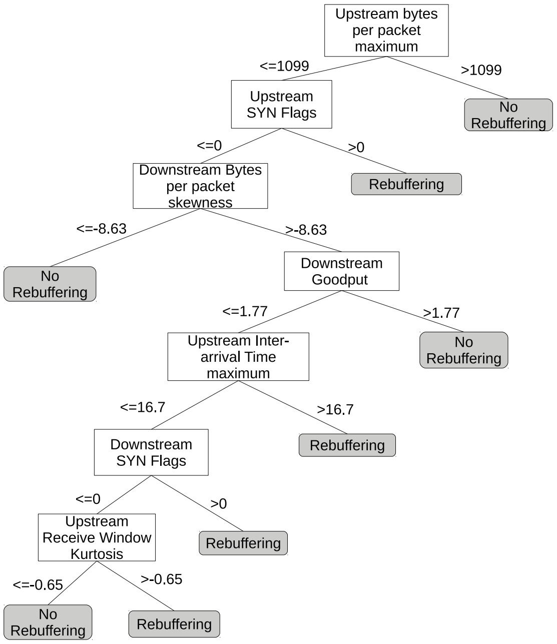


Fig. 8. Decision tree to infer rebuffering events on HTTPS

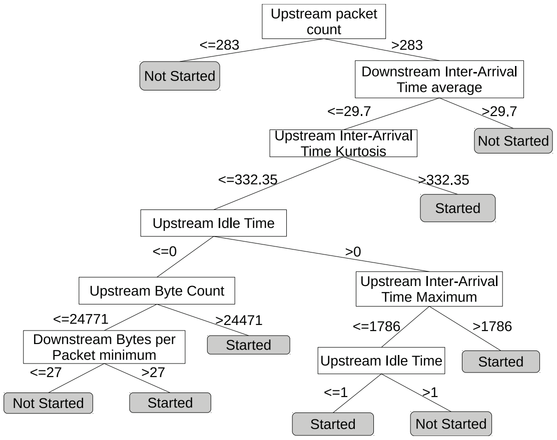


Fig. 9. Decision tree to infer startup delay on QUIC

Figure 9 visualizes a pruned decision tree model that is trained to infer startup delay (at $k = 3.3$ seconds) for YouTube streaming on QUIC. The decision tree was pruned using a confidence factor of 10% with a limit of at least 100 instances at leaf nodes. The pruned decision tree model is approximately 2% less accurate as compared to its unpruned counterpart. The root node of the decision tree shows that fewer upstream packets are indicative of Not Started sessions. Higher downstream average packet inter-arrival time are also indicative of Not Started sessions.

D. Limitations

We note that our results and insights are specific to YouTube and its current video quality adaption logic. We expect the models generated by our approach to differ across not only different video content providers but also over time for the same video content provider. However, our approach is generalizable to any video content provider.

To train supervised machine learning models, network operators can collect ground truth data by streaming videos on test devices. Since network conditions and video quality adaptation logic can change over time, a network operator can collect new ground truth continuously, estimate the effectiveness of the current models, and retrain with new ground truth data if they are inaccurate.

Our approach is primarily geared towards *detection* of QoE impairments in encrypted video traffic. However, network operators need actionable insights to address the detected QoE impairments. Network operators can use the decision tree model to infer potential root causes of QoE impairments.

Some QoE impairments may arise due to issues in the video content provider's network which is beyond the control of the network operator. The network operator can collaborate with the video content provider to mitigate such issues using architectures such as EONA [16], which defines interfaces for network operators and content providers to share information regarding application experience.

VI. Conclusion

We presented a machine learning based approach for real-time inference of encrypted video QoE metrics. We first designed a comprehensive set of features from network and transport layer information. For videos streamed over HTTPS, we utilize both network and transport layer features. For videos streamed over QUIC, we utilize only network layer features. Using the feature sets, we then trained supervised decision tree models on YouTube videos streamed over HTTPS and QUIC. The experimental evaluation showed that our proposed approach achieved up to 90% classification accuracy for HTTPS and up to 85% classification accuracy for QUIC.

VII. Acknowledgements

We thank Sai Kalyan Moguloju and Avinash Talreja for their assistance in data collection. We also thank Yong Ren and Yanzhi Ren of Futurewei Technologies for sharing their insights. This work is supported in part by the National Science Foundation under grant numbers 1464110 and 1617288, and an unrestricted gift by Futurewei Technologies.

References

[1] Cisco Visual Networking Index: Forecast and Methodology, 2016 to 2021. http://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/complete-white-paper-c11-481360.html, June 2017.

[2] 3GPP. Digital cellular telecommunications system (Phase 2+); Universal Mobile Telecommunications System (UMTS); LTE; Policy and charging control architecture.

[3] 3GPP. Evolved Universal Terrestrial Radio Access Network (E-UTRAN); Self-configuring and self-optimizing network (SON) use cases and solutions.

[4] V. Aggarwal, E. Halepovic, J. Pang, S. Venkataraman, and H. Yan. Prometheus: Toward Quality-of-experience Estimation for Mobile Apps from Passive Network Measurements. In *ACM HotMobile*, 2014.

[5] M. Alleven. YouTube driving more QUIC-based traffic on mobile: Vasona. http://www.fiercewireless.com/wireless/youtube-driving-more-quic-based-traffic-mobile-vasona, April 2017.

[6] P. Ameigeiras, J. J. Ramos-Munoz, J. Navarro-Ortiz, and J. Lopez-Soler. Analysis and modelling of YouTube traffic. *Transactions on Emerging Telecommunications Technologies*, 2012.

[7] L. Bernaille, R. Teixeira, I. Akodkenou, A. Soule, and K. Salamatian. Traffic Classification on the Fly. *ACM SIGCOMM*, 2006.

[8] L. Bernaille, R. Teixeira, and K. Salamatian. Early Application Identification. In *ACM CoNEXT*, 2006.

[9] P. Casas, M. Seufert, and R. Schatz. YOUQMON: A System for On-line Monitoring of YouTube QoE in Operational 3G Networks. *ACM SIGMETRICS*, 2013.

[10] G. Dimopoulos, I. Leontiadis, P. Barlet-Ros, and K. Papagiannaki. Measuring Video QoE from Encrypted Traffic. In *ACM IMC*, 2016.

[11] F. Dobrian, V. Sekar, A. Awan, I. Stoica, D. Joseph, A. Ganjam, J. Zhan, and H. Zhang. Understanding the Impact of Video Quality on User Engagement. In *ACM SIGCOMM*, 2011.

[12] H. Ekstrom. QoS control in the 3GPP evolved packet system. *IEEE Communications Magazine*, 2009.

[13] Google. Adding YouTube and Calendar to the HTTPS Transparency Report. https://security.googleblog.com/2016/08/adding-youtube-and-calendar-to-https.html, August 2016.

[14] T.-Y. Huang, R. Johari, and N. McKeown. Downton Abbey Without the Hiccups: Buffer-based Rate Adaptation for HTTP Video Streaming. In *ACM FhMN*, 2013.

[15] ITU-T. Parametric bitstream-based quality assessment of progressive download and adaptive audiovisual streaming services over reliable transport, 2016.

[16] J. Jiang, X. Liu, V. Sekar, I. Stoica, and H. Zhang. EONA: Experience-Oriented Network Architecture. In *ACM HotNets*, 2014.

[17] J. Jiang, V. Sekar, and H. Zhang. Improving Fairness, Efficiency, and Stability in HTTP-based Adaptive Video Streaming with FESTIVE. In *ACM CoNEXT '12*, 2012.

[18] M. Katsarakis, R. C. Teixeira, M. Papadopouli, and V. Christophides. Towards a Causal Analysis of Video QoE from Network and Application QoS. In *ACM Internet-QoE*, 2016.

[19] S. S. Krishnan and R. K. Sitaraman. Video Stream Quality Impacts Viewer Behavior: Inferring Causality Using Quasi-experimental Designs. In *ACM IMC*, 2012.

[20] Z. Li and P. Mohapatra. QRON: QoS-aware routing in overlay networks. *IEEE Journal on Selected Areas in Communications*, 2004.

[21] M. Liberatore and B. N. Levine. Inferring the Source of Encrypted HTTP Connections. In *ACM CCS*, 2006.

[22] A. W. Moore and D. Zuev. Internet Traffic Classification Using Bayesian Analysis Techniques. In *ACM SIGMETRICS*, 2005.

[23] Netflix. Protecting Netflix Viewing Privacy at Scale. https://medium.com/netflix-techblog/protecting-netflix-viewing-privacy-at-scale-39c675d88f45, 2016.

[24] T. T. T. Nguyen and G. Armitage. A survey of techniques for Internet traffic classification using machine learning. *IEEE Communications Surveys Tutorials*, 2008.

[25] I. Orsolic, D. Pevec, M. Suznjevic, and L. Skorin-Kapov. A machine learning approach to classifying YouTube QoE based on encrypted network traffic. *Multimedia Tools and Applications*, 2017.

[26] Sandvine. Encrypted Internet Traffic: A Global Internet Phenomena Spotlight, 2016.

[27] R. Schatz, T. Hofeld, and P. Casas. Passive YouTube QoE Monitoring for ISPs. In *ACM IMIS*, 2012.

[28] R. Schuster, V. Shmatikov, and E. Tromer. Beauty and the Burst: Remote Identification of Encrypted Video Streams. In *USENIX Security*, 2017.

[29] G. Szabo, I. Szabo, and D. Orincsay. Accurate Traffic Classification. In *IEEE WoWMoM*, 2007.

[30] A. M. White, A. R. Matthews, K. Z. Snow, and F. Monrose. Phonotactic Reconstruction of Encrypted VoIP Conversations: Hookt on Fon-iks. In *IEEE Symposium on Security & Privacy*, 2011.

[31] C. V. Wright, L. Ballard, F. Monrose, and G. M. Masson. Language Identification of Encrypted VoIP Traffic: Alejandra y Roberto or Alice and Bob. In *USENIX Security*, 2007.

[32] YouTube. YouTube Player API Reference for iframe Embeds. https://developers.google.com/youtube/iframe_api_reference, July 2017.