Lightweight, General Inference of Streaming Video Quality from Encrypted Traffic

ABSTRACT

Accurately monitoring application performance is becoming more important for Internet Service Providers (ISPs), as users increasingly expect their networks to consistently deliver acceptable application quality. At the same time, the rise of end-to-end encryption makes it difficult for network operators to determine video stream quality—including metrics such as startup delay, resolution, rebuffering, and resolution changes—directly from the traffic stream. This paper develops general methods to infer streaming video quality metrics from encrypted traffic using lightweight features. Our evaluation shows that our models are not only as accurate as previous approaches, but they also generalize across multiple popular video services, including Netflix, YouTube, Amazon Instant Video, and Twitch. The ability of our models to rely on lightweight features points to promising future possibilities for implementing such models at a variety of network locations along the end-to-end network path, from the edge to the core.

1 INTRODUCTION

Video streaming traffic is by far the dominant application traffic on today's Internet, with some projections stating that video streaming will comprise 82% of all Internet traffic in just three years [13]. Video content providers and Internet Service Providers (ISPs) both manage network routing and content delivery to ensure that users experience good video streaming performance; content providers use distributed content delivery [2] to serve video streams, adapting the video bitrate to changing conditions [19]. ISPs deploy proxies, transcode and compress video, and re-route traffic to deliver good video quality with the existing network resources.

Optimizing video delivery depends on the ability to determine the quality of the video stream that a user receives. In contrast to video content providers, who have direct access to video quality from client software, network operators such as ISPs must typically infer video quality from traffic as it passes through the network. End-to-end encryption, which is becoming increasingly common with video streaming over HTTPS and QUIC [28, 32], prevents an ISP from directly observing precise application metrics such as startup delay, rebuffering events, and video resolution from the video streaming protocol [5, 17]. Without access to such metrics, ISPs cannot determine the quality of video sessions or the underlying causes of quality degradation; therefore, they cannot detect when the network may be inducing poor application performance or perform steps to improve the delivery of the application to users.

Several methods to infer video quality from encrypted video traffic have been proposed over the last few years [14, 22, 26], yet these methods are not general, lightweight, or fine-grained. First, previous work has focused mostly on YouTube traffic, but video streaming algorithms and content characteristics can vary significantly across video services (*e.g.*, buffer-based [19] versus throughput-based [36] rate adaption algorithm or fixed-size [19]

versus variable-size video segments [27]). Thus, a model that is tuned for YouTube is unlikely to work more generally across the growing set of video streaming applications. We find, in fact, that these previous methods do not work well for other video streaming services, and thus that more general models are needed.

Second, existing models cannot perform on-path processing; these existing approaches are typically computationally complex, and they often require either deep packet inspection or the computation of stateful features that can be difficult to extract and track for a large number of flows. Third, existing work predicts the quality of experience at a coarse granularity, such as whether quality is "good" or "bad", as opposed to more precise metrics such as resolution or startup delay. Finally, most previous approaches typically classify application granularity for an entire session and do not track these metrics over shorter time intervals within a single session.

In this paper, we develop lightweight, general models that infer four precise, fine-grained video quality characteristics: (1) startup delay, (2) resolution, (3) rebuffering, and (4) resolution changes—from encrypted traffic. Moreover, our new models work well across more video services than any existing method. We demonstrate this generality by showing that our models can predict application quality metrics for four popular video streaming services: Netflix, YouTube, Amazon Instant Video, and Twitch.

The models that we develop lend insight into the types of features in encrypted traffic that are best suited to predict application performance. We present the following contributions:

(1) We develop models that support inference with **lightweight** features. We evaluate the effects that various features have on model accuracy, including how features that require different amounts of state affect prediction accuracy. Our results show that models that rely on lightweight features (*e.g.*, upstream and downstream throughput, inferred video segment sizes) can outperform the accuracy of those that require maintaining more complex per-flow state (*e.g.*, bytes in flight, number of retransmissions). (Section 6)

(2) We demonstrate that our models **generalize** to a wide variety of streaming video applications. In contrast to previous work, which has developed models that are specific to a particular video streaming service, we demonstrate that the models that we have developed can generalize across a wide range of services. (Section 7)

In addition to these two primary contributions, we also show that the predictive models that we develop for the video quality characteristics can produce fine-grained predictions of startup delay, resolution, rebuffering, and resolution changes, in contrast to previous work, whose predictions have generally been binary (*i.e.*, good vs. bad) or otherwise more coarse-grained. Finally, we show that our models can predict video quality on shorter time intervals within a given session, providing more detailed information about how the quality of a video stream evolves over time.

Our results present several avenues for future work. Most importantly, the ability to predict the quality of streaming video applications on lightweight features suggests that our models can be implemented at line rate in software or hardware switch deployments. We have currently deployed software in more than 50 homes to capture streaming statistics that we can use to estimate streaming video quality and are also exploring whether and how these models could be implemented in programmable hardware.

2 INTERNET VIDEO STREAMING

In this section, we provide background on Internet video streaming and define common video quality metrics.

2.1 DASH Protocol

Internet video streaming services typically use Dynamic Adaptive Streaming over HTTP (DASH) [35] to deliver a video stream. DASH divides each video into time slices known as *segments* or chunks (of possibly equal duration), which are then encoded at multiple bitrates and resolutions. These segments are typically stored on multiple web servers so that a client can download segments from a nearby server.

At the beginning of a video session, the client downloads a DASH Media Presentation Description (MPD) file from the server. The MPD file contains all of the information the client needs to retrieve the video (*i.e.*, the audio and video segment file information). The client sequentially issues HTTP requests to retrieve segments at a particular bitrate. An *application-layer Adaptive Bitrate (ABR) algorithm* determines the quality of the next segment that the client should request. ABR algorithms are proprietary, but most video services rely on recently experienced bandwidth [36], current buffer size [19], or a hybrid of the two [42] to guide the selection. The downloaded video segments are stored in a client-side application buffer. The video buffer is meant to ensure continuous playback during a video session. Once the size of the buffer exceeds a predefined threshold, the video starts playing. A video session can typically be classified into two distinct phases: (1) the buffering phase, where the client fetches video segments as quickly as possible to fill the buffer; and (2) the steady-state phase, where a client downloads new video segments at roughly the same rate as playback to maintain a stable buffer level.

2.2 Streaming Video Quality Metrics

A user's experience while watching a video session depends on many factors that range from the user's expectations and context to network and device performance [21]. Our goal is to study techniques that network operators can use to monitor video quality. As a result, we focus on four technical video quality metrics that depend on network performance and have shown to affect user quality of experience [3, 6, 7, 15, 16, 20, 23].

Startup delay. Startup delay is the time elapsed from the moment the player initiates a connection to a video server to the time it starts rendering video frames. Prior work has shown that higher startup delays cause more users to abandon video sessions [23].

High startup delays also reduce the total time that users spend on a video service [15].

Video bitrate. The bitrate of video segments is one metric of the quality of the displayed content. The average bitrate over a video session (computed as the average of the bitrates played weighted by the duration each bitrate is played) is the key quality metric for video content [15]. The relationship between bitrate and quality is complex because the bitrate depends on the resolution, the encoding, and the content type [1].

Bitrate switches. Video players switch the bitrate to adapt to changes in network conditions with the goal to select the best possible bitrate for any given condition. Nevertheless, bitrate switching has a negative effect on quality of experience [6, 18]. Both the frequency of bitrate switches (or switches per second) and the amplitude of switches—the difference between the bitrate before and after the change—affect quality of experience [18].

Rebuffering events. This metric captures the periods of time video playback stalls because the buffer is empty. We can measure the impact of rebuffering with two metrics: the rebuffering time, which is the total time spent rebuffering in a session, or the rate of rebuffering, which is the frequency of rebuffering events. High rebuffering rate and time increase the rate of abandonment and reduce the likelihood that a user will return to the service [15, 23].

3 EXISTING QUALITY INFERENCE METHODS

Network operators must infer video quality from the traffic that traverses their network. Because this traffic is often encrypted [28, 32], several previous approaches have been developed to infer these quality features, since they cannot be observed directly:

- Dimopoulos et al. [14] developed a method that runs on a web proxy in the network to identify YouTube sessions over TCP, collects traffic statistics and HTTP requests to infer rebuffering, bitrate switches, and the average quality of the entire video session.
- Mazhar and Shafiq [26] infer video bitrate, startup delay, and rebuffering events for YouTube (over both TCP and QUIC) using network and transport layer statistics collected per ten-second time slots.
- BUFFEST [22] identifies individual video segments within video traffic to then infer YouTube rebuffering events using a buffer-state emulator.
- eMIMIC [25] relies on the same method to identify video segments and then models the video session as a sequence of segments to infer average bitrate, bitrate switches, rebuffering ratio, and startup delay of a video session. eMIMIC's model must be parameterized for each video service.

Existing models do not generalize. Unfortunately, we find that the models from previous work do not generalize well across multiple services. Notably, most existing methods have only been evaluated for a single service: YouTube. The lone exception is eMIMIC, which was evaluated for two undisclosed video services, but the eMIMIC model has its own set of drawbacks: it assumes fixed-length video segments, making it inapplicable to video services that use variable-length segments such as YouTube. Additionally, existing models are often quite coarse-grained. Both Dimopoulos et al. and
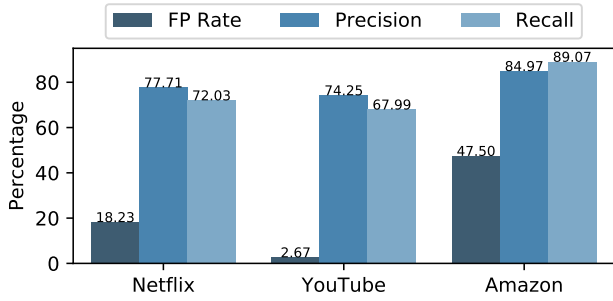
Figure 1: Using a per-service model to detect high (bigger than 5) startup delay using method from Mazhar and Shafiq [26].

eMIMIC output metrics on a per-session granularity, rather than at finer-grained intervals. One reason that general models are challenging to develop is because each video service (and client) has unique buffering behavior, as the service may prioritize different features such as resolution or short startup delays; these unique behaviors mean that models such as BUFFEST that rely on buffer-state estimation cannot generalize to many video services.

We evaluated the state-of-the-art method from Mazhar and Shafiq [26] which is the only existing method that infers multiple video quality metrics at fine-scale granularity (per ten-second time slots). We use over 11,000 videos from four popular video services: Netflix, YouTube, Amazon Prime Video, and Twitch. We used scripts provided by the authors to extract the same features from the collected video sessions. We use Scikit-learn's [29] AdaBoost implementation with one hundred default weak learners. We used the method from Mazhar and Shafiq to train one specific model per service (as in previous work [14, 25, 26]) and evaluate how well we can detect low resolution and long startup delays.

Here, we show results for the startup delay—whether the video has not started within five seconds after the video page has been requested. Figure 1 shows the false positive rate (FPR), precision, and recall when testing the model for each video service.

The model is not effective at detecting high startup delay across a variety of video streaming services. Although we found that the detection rate for YouTube was similar to that reported in previous work [26], false positive rates are around 20% for Netflix and even higher for Amazon. We discovered that 99% of Twitch sessions do not start playing before five seconds, so predicting a startup delay that exceeds five seconds was not a meaningful exercise—it almost always happens. The results from these experiments show that even when we train one model per service using the method from previous work the accuracy is poor.

4 PROBLEM FORMULATION

In this section, we define the problem we address in the rest of the paper: lightweight, general video quality inference from encrypted network traffic. We first explore the set of video quality metrics that a model should predict, as well as the granularity of those predictions. Next, we discuss the space of possible features that

the model could use, discussing not only the features themselves but also identifying *lightweight* features that are most amenable to on-path processing. Finally, we discuss our goal of developing a *general* model for streaming video quality.

4.1 Output: Streaming Video Quality

In this section, we focus on how to set the inference goals for each of the metrics presented in Section 2.2. For example, given a metric, say resolution, one can use a regression model to output a continuous value (*e.g.*, the average resolution of a session) or a classifier (whether resolution was high or low, as in the previous section). For the classification version of the problem, the selection of the classes may depend on the service. For example, in the previous section, a five-second cutoff for startup delay was too short for Twitch.

Startup delay. eMIMIC [25] proposed to infer startup delay as the delay to download the first segment. This metric, however, is misleading as most services download multiple segments prior to startup. For example, we observe in our dataset that Netflix downloads on average 4 video and 1 audio segments before playback begins. Alternatively, Mazhar and Shafiq [26] rely on machine learning to model startup delay. Although it is natural to model startup delay as a regression problem where the goal is to infer the exact delay, they have modeled it as a binary classification problem [26]. The reasoning for this approach comes from prior results, which showed that users tend to ignore startup delay up to a given threshold, after which they report poor experience [23]. This threshold varies depending on the nature of the service, (*e.g.*, roughly 3 seconds for short videos on a service like YouTube and 5 seconds for movies on a service such as Netflix). Therefore, a binary classifier based on a single threshold is unlikely to generalize across services. Instead, our method relies on a regression model to achieve finer inference granularity, which helps generalize across video services.

Resolution (or bitrate). Bitrate and resolution often vary through the course of a video session; therefore, the typical inference target is the average bitrate or resolution. Previous work has predominantly used classification approaches [14, 26]: either binary (good or bad) [26] or with three classes (high definition, standard definition, and low definition) [14]. Such a representation can be misleading as different clients may have different hardware configurations (*e.g.*, a smartphone with a 480p screen compared with a 4K smart television). An alternative is to infer the average bitrate based on the amount of data downloaded during the streaming session [25]. This approach can also lead to confusing results as the relationship between bitrate and quality is complex, since the bitrate depends on not only the resolution, but also the encoding and the content type. Therefore, our models infer resolution, which is easier to analyze and also to generalize across services. One can compute the average resolution of a video session [14, 25] or of a time slot [26]. We opt to infer video quality per time slot.[1] This finer-grained inference can later be used to also recover per-session statistics.

Resolution switches and rebuffering. We can model both resolution switches and rebuffering events as a binary classification (*e.g.*, whether or not the video rebuffered during some window) or a regression problem (*e.g.*, rate of events). Prior work has focused on

---

[1]We use 10-second slots as justified in Section 5.4.

inferring the degree of rebuffering (no stalling, medium stalling, or severe stalling) [14, 25] or on detecting instances when the buffer is running low [22]. For resolution switches, prior work used a similar approach to infer resolution variation (*i.e.*, no, mild, or high variation) [14]. We instead focus on detecting whether there was a rebuffering event in a given time slot as proposed by Mazhar and Shafiq [26]. This approach allows finer-grained inference of the rate and frequency of rebuffering. Similarly, we detect the presence of resolution switches in a given time slot.

4.2 Input: Lightweight Features

The selection of a video quality inference method requires understanding which metrics provide the best tradeoffs between collection overhead, processing complexity, and detection accuracy. Some metrics can easily be computed in the network path, whereas others (*e.g.*, end-to-end latency), require maintaining per-flow state and observations of bidirectional flows—which is often heavyweight and may even be impossible at some vantage points. We discuss features at various levels of the network stack below.

Network Layer. We define network-layer features as metrics that solely rely on lightweight information available from observation of a network flow (identified by the IP/port four-tuple). This includes metrics such as flow throughput, packet/byte counters, or packet arrivals. Being lightweight, one can easily collect these metrics from any monitoring point in the end-to-end path that can observe network flows that belong to a video session. Moreover, most modern routers are capable of gathering this information as the state necessary to do so is relatively minimal.

Transport Layer. Transport-layer features include information such as end-to-end latency and packet retransmissions. These metrics reveal possible network problems, such as presence of a lossy link in the path or a link causing high round-trip latencies between the client and the server. Unfortunately, transport metrics suffer two shortcomings. First, due to encryption, some metrics are only extractable from TCP flows and not flows that use QUIC (which is increasingly prevalent for video delivery). Second, many transport-layer features require maintaining long-running per-flow state, which is prohibitive at scale.

Application Layer. Application-layer metrics include any feature related to the application data, which often provide the greatest insight into the performance of a video session. Encryption, however, makes it impossible to directly extract any application-level information from traffic using deep packet inspection. Fortunately, we can still derive some application-level information from encrypted traffic. For example, BUFFEST [22] showed how to identify individual video segments from the video traffic, using the times of upstream requests (*i.e.*, packets with non-zero payload) to break down the stream of downstream packets into video segments. Our experiments found that this method works well for both to TCP and QUIC video traffic. In the case of QUIC, signaling packets have non-zero payload size, so we use a threshold for QUIC UDP payload size to distinguish upstream and downstream packets. Tracking these features requires recording the times clients issue upstream requests (*i.e.*, upstream packets with payload bigger than zero) to break down the stream of downstream packets into video segments. This process requires a limited amount of state as it needs one

register per flow to keep track of the last upstream request and byte counters for subsequent downstream packets.

Section 6 presents the features we use for our models as well as how we represent them.

4.3 Goal: General Prediction Model

Network providers need to track the performance of an ever-evolving set of video services. An ideal video quality inference technique should perform accurate prediction across a variety of services, without requiring a specific model to be trained for each service.

Per-service models. As discussed in Section 3, most existing methods [14, 22, 26] focused on YouTube. Our results in Section 3 show that simply retraining existing models across different video services can lead to poor prediction accuracy. Moreover, even if effective, applying the same method across different video services to obtain per-service models requires operators to collect a sufficiently diverse set of video sessions for each service and to calibrate the model per service, which is intractable with the increasing number of video services. Complicating matters further, video services often update their client video player logic, which may require network operators to retrain models if they are too tailored to the specific dynamics of the service [22].

General model. A preferable approach is to develop a combined model that works across multiple services. This approach is more appealing, as it allows network operators to train their algorithms periodically for all services, which is desirable considering that the majority of video services have converged to DASH-based algorithms that share similar yet not identical streaming dynamics. This generality may result in lower precision, recall, or both, but may be sufficient as long as the generated models fit some target performance metrics.

Section 7 evaluates the accuracy of video quality detection for both per-service and general models.

5 METHOD

This section first describes our methods to generate labeled video traffic and the resulting dataset used to evaluate video quality inference methods. Then, we present the set of features we compute from the input metrics discussed in the previous section and how we train models to infer the four target metrics.

5.1 Generation of Video Traffic

Inferring video quality from encrypted network traffic requires network traffic of video sessions, where each session is labeled with video quality metrics. To develop this dataset, we instrument a set of laptops to collect network traffic, and we generate video traffic automatically for four popular video services, Netflix, YouTube, Twitch, and Amazon Instant Video (Netflix and YouTube together represent over 50% of fixed Internet traffic in Latin America and North America [34]).

We generate video sessions automatically using `ChromeDriver` [12]. We play each session for 8 to 12 minutes depending on the video length and whether there are ads at the beginning of the session (in contrast to previous work [26],

we do not remove ads from the beginning of sessions in order to recreate the most realistic setting possible). For longer videos (e.g. Netflix movies), we vary the playback starting point to avoid always capturing the first portion of the video. We generate five categories of sessions: Netflix, Amazon, Twitch, YouTube TCP, and YouTube QUIC. Netflix and Amazon movies are randomly selected from the suggestions presented by the service in the catalog page. To avoid biasing our results we select movies from different categories including action, comedies, TV shows, and cartoons. We ultimately select 25 movies and TV shows used in rotation from Netflix and 15 from Amazon Prime Video. Similarly for YouTube, we select 30 videos from different categories. Twitch automatically starts a live video feed when opening the service home page. Thus, we simply collect data from the automatically selected feed.

5.2 Data Collection and Labeling

We collect packet traces using `tcpdump` [39] on the network interface the client uses to stream videos. To label these traffic traces with the appropriate video quality metrics, we develop a Chrome extension that monitors application-level information for the four services. This extension allows us to assign video quality metrics to each stream as seen by the client. [2]

The extension collects browsing history by parsing events available from the Chrome WebRequest APIs [11]. This API exposes all necessary information to identify the start and end of video sessions, as well as the HTTPS requests and responses for video segments. To collect video quality metrics, we first use the Chrome browser API to inspect the URL of every page.

The extension identifies flows that correspond to each of these video services in the packet traces and collects the appropriate statistics for each flow. Because video sessions are typically encrypted, we rely on the QNAME in client DNS queries to identify video flows. We inspect the hostname in DNS queries and match these lookups against well-known domains of video services using regular expressions. If the DNS domain name corresponds to one of the four video services that we study, we then track all video quality metrics defined in Section 2.2 as follows:

Netflix: Parsing overlay text. Netflix reports video quality statistics as an overlay text on the video if the user provides a specific keystroke combination. We inject this keystroke combination but render the text invisible, which allows us to parse the reported statistics without impacting the playback experience. This information is updated once per second, so we adjust our collection period accordingly. Netflix reports a variety of statistics. We focus on the player and buffer state information; including whether the player is playing or not, buffer levels (*i.e.*, length of video present in the buffer), and the buffering bitrate and resolution. The one-second refresh cycle limits the precision for detecting resolution switches. Fortunately, this precision is sufficient for our analysis as Netflix uses four-second video segments, hence resolution switches rarely happen more than once per second.

YouTube: iframe API. We use the YouTube iframe API [43] to periodically extract player status information, including current video resolution, available playback buffer (in seconds) and current

Service	Total Runs	% Home	% Lab
Netflix	3,498	59	41
YouTube TCP	4,232	18	72
YouTube QUIC	1,101	68	32
Twitch	2,231	17	83
Amazon	1,852	10	90

Table 1: Summary of the dataset.

playing position. Additionally, we collect events reported by the *<video> HTML5* tag, which exposes the times that the player starts or stops the video playback due to both user interaction (*e.g.*, pressing pause) or due to lack of available content in the buffer. The YouTube API reports the resolution of the video being played, but not the actual average bitrate. We then use the URLs of the downloaded video segments to extract the nominal bitrate of the video segment using *pytube* [30], an external python library.

Twitch and Amazon Video: HTML 5 tag parsing. As the two services expose no proprietary interface, we generalize the module developed for YouTube to solely rely on the *<video> HTML5* tag to collect all the required data. This approach allows us to collect all the events described above as well as player status information, including current video resolution, available playback buffer (in seconds), and current playing position. The only limitation is that the real bitrate is not available, so we focus on inferring resolution instead of bitrate.

5.3 Experiment Setup

We instrumented 11 machines to generate video traffic and collect packet traces together with the data from the Chrome extension: six laptops in residences connected to the home WiFi network (three homes in a European metropolitan area with download speeds of 100 Mbps, 18 Mbps, and 6 Mbps, respectively; one room in an European student residence; one apartment in a campus town in the US; and one home from a rural area in the US), four laptops located in our lab connected via the local WiFi network, and one desktop connected via Ethernet to our lab network.

In the lab environment, we manually varied the network conditions in the experiments using `tc` [9] to ensure that our datasets capture a wide range of network conditions. These conditions can either be stable for the entire video session or vary at random time intervals. We varied capacity from 50kbps to 30mbps, and introduce loss rates between 0% and 1% and additional latency between 0 ms and 30 ms. All experiments within homes ran with no other modifications of network conditions, to emulate realistic home network conditions.

We collected data from November 20, 2017 to June 18, 2018. We filtered any session that experienced playing errors during the execution. For example, we noticed that when encountering particularly challenging video conditions (*e.g.*, 50 kbps download speeds), Netflix's player simply stops and shows an error instead of stalling and waiting for the connectivity to return to playable conditions. The resulting dataset contains 11,767 video sessions. Table 1 shows the number of the runs per video service and under the different network conditions. We will release this dataset with the publication of this paper.

---

[2]The extension supports any HTML5 based video. It will be made available after paper acceptance.

Network Layer		Transport Layer		Application Layer	
throughput up/down (total, video, non-video)	○	# flags up/down (ack/syn/rst/push/urgent)	○	segment sizes (all previous, last-10, cumulative)	◐
throughput down difference	○	receive window size up/down	○	segment requests inter arrivals	◐
packet count up/down	○	idle time up/down	○	segment completions inter arrivals	◐
byte count up/down	○	goodput up/down	○	# of pending request	◐
packet inter arrivals up/down	○	bytes per packet up/down	○	# of downloaded segments	◐
# of parallel flows	○	round trip time	●	# of requested segments	◐
		bytes in flight up/down	●		
		# retransmissions up/down	●		
		# packets out of order up/down	●		

Table 2: Summary of the extracted features from traffic. Amount of state required: Simple state (○), Simple application-layer state (◐), Complex state (●).

5.4 Feature Description

For each video session, we computed a set of features from the captured traffic and associated metrics, as summarized in Table 2. We also loosely characterize the cost of computing these features, which is roughly correlated with the amount of state required to collect and compute the associated feature.

Network Layer. Flows to video services fall into three categories: flows that carry video traffic, flows that belong to the service but transport other type of information (*e.g.*, the structure of the web page), and all remaining traffic flows traversing to-and-from the end-host network. For each network flow corresponding to video traffic, we compute the upstream and downstream average throughput as well as average packet and byte counts per second. We also compute the difference of the average downstream throughput of video traffic between consecutive time slots. This metric captures temporal variations in video content retrieval rate. Finally, we compute the upstream and downstream average throughput for service flows not carrying video and for the total traffic. Modern hardware can compute each of these features in-line with a single register (*e.g.*, to store counters for throughput or timestamps for packet time arrivals).

Transport Layer. We extract transport-layer features from information in TCP/UDP headers, computing each feature separately for upstream and downstream traffic, as in previous work [26]. We calculate the features by analyzing individual packets, and their relative state in the lifetime of the flow. For TCP, we extract retransmissions, window sizes, out-of-order packets, and TCP flag information. The rest of the features are computed for UDP, as well. We compute summary statistics such as mean, median, maximum, minimum, standard deviation, kurtosis, and skewness. The cost of computing each feature depends on whether the full state of the flow is required: Features such as goodput and TCP flag counters (in the upper part of the table) require simple state and can be computed in line similarly to the network-layer ones. The remaining features require more complex mechanisms to inspect each packet and track state across packets belonging to the flow, as shown in the bottom of the table.

Application Layer. We use sequences of inferred video segment downloads to build up the feature set for the application layer. In particular, we compute the following features: the sizes of the downloaded segments (both for the entire flow history and the last ten segments); this provides insight into the amount of data per second of video delivered. Features such as the interarrival times for segment requests, completed downloads, and the number of outstanding downloads and active flows provide information on how many requests the player is capable of issuing (and complete) at any specific point in time. For each one of these features we compute the following statistics: minimum, mean, maximum, standard deviation and 50th, 75th, 85th, and 90th percentile. These features require a small amount of state to track, consisting of per flow streaming registers to track upstream packets and byte counters to infer segment sizes.

We compute these features at fixed time intervals and label all time slots with the video quality (in particular, startup delay, resolution, number of rebufferings and resolution switches) extracted using the Chrome extension. We consider bins of 5 and 10 seconds or at the end of each downloaded video segments. To decide on the bin size, we study the number of switches and rebufferings per time bin using our ground truth. The overwhelming majority of time bins have no bitrate switches nor rebuffering events. For example, at 5 seconds, more than 93% of all bins for Netflix have no quality switch. As the bin size increases, we have more switches per bin and we lose on the precision of switch detection. At the same time, if the time bin is too short it may only capture the partial download of a video segment, in particular when the network conditions are poor. We select 10-second bins, which represent the best tradeoff between the precision of detecting events and ensuring downloads of full segments per bin.

5.5 Model Generation

To generate our models, we experimented with different methods: Adaboost (as in prior work [26]), logistic regression, decision trees, and random forest. We evaluate methods based on precision, recall and false positive rate. Our analysis, omitted for brevity, finds that random forest achieves the highest accuracy for all our inference goals; hence, the rest of this paper focuses on the results of random forest models.

Training and testing sets. We generate different training sets: network-layer features (*Net*), transport-layer features (*Tran*), application-layer features (*App*), as well as a combination of features from different layers: Net+Tran, Net+App and all layers combined (*All*). For each target quality metric, we train 30 models in total: (1) varying across these six features sets and (2) using five different datasets, splitting the dataset with sessions from each of the
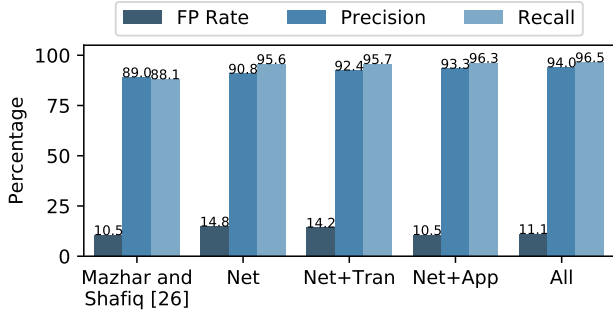
Figure 2: Startup delay inference error based different features sets (aggregated into two-second bins based on real startup delay).

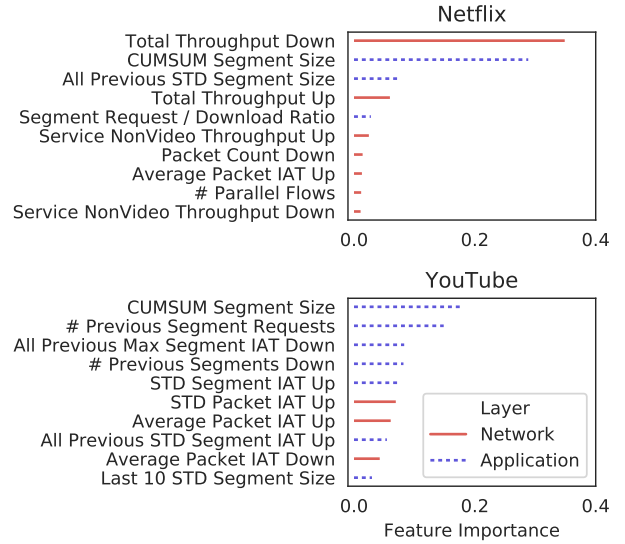Figure 3: Startup delay feature importance for Netflix and YouTube models.

four video services—Netflix, YouTube, Amazon Prime Video, and Twitch—plus a combined dataset with sessions from all services. For models that rely on transport-layer features, we omit YouTube sessions over UDP as we cannot compute all features. For each target quality metric, we evaluate models using 10-fold cross-validation. We do not present the results for models based only on transport-layer or application-layer features, because the additional cost of collecting lightweight network-layer features is minimal compared to the more stateful features in the other layers, and these models are less accurate, in any case.

6 MAKING INFERENCE LIGHTWEIGHT

In this section, we evaluate the accuracy of various models for predicting streaming video quality, particularly evaluating models that incorporate only lightweight features against models that use more comprehensive feature sets. We find that the models that rely on only lightweight features (*e.g.*, lightweight network-layer and application-layer features) perform nearly as well as models that rely on the complete set of features.

6.1 Startup Delay

For predicting startup delay, we train the models on the first ten seconds of each video session. We train one random forest regressor for each service feature set and service combination. We present the average results across the four video services. To allow direct comparison with Mazhar and Shafiq [26], we convert the output of our (more fine-grained) regression models into a binary classification problem to predict whether the video failed to start within the first five seconds. Figure 2 presents the precision, recall, and false positive rate of our models using different feature sets together with that of Mazhar and Shafiq's model. The results show that the precision and recall of all of our models improve upon the state of the art. Interestingly, our results also show that the model using a combination of features from network and application layer yields the highest precision and recall with tolerable false positive rate. These results also show that we can exclude Net+Tran models, because transport-layer features are the costliest to track and Net+App models consistently achieve higher precision/recall and lower false positive rates in any case. Additionally, we find that

solely using network-layer features achieves higher precision and recall compared to Mazhar and Shafiq [26], at the cost of higher false positive rate. Overall, the model that relies on network and application-layer features (Net+App) provides the best tradeoff between accuracy and state required, but network operators may choose a model that uses only network-layer features, which result in the lowest monitoring overhead with a small sacrifice in the inference quality.

To further understand the effect of different types of features on the models, we study feature importance (based on the Gini index [8]) across the different services. Figure 3 ranks the ten most important features for both Netflix and YouTube for the Net+App model. Overall, application-layer features dominate YouTube's chart with predominantly features that indicate how fast the client downloads segments, such as the cumulative segment size and their inter-arrival times. We observe the same trend for Amazon and Twitch. In contrast, the total amount of downstream traffic (a network-layer feature) dominates Netflix's list. Although this result seems to suggest a possible difference across services, this feature is also indicative of the number of segments that are being downloaded, meaning that ultimately this feature is indeed a key factor on how fast the client can begin video playback. In fact, all of these features are aligned with general DASH video streaming behavior where the video startup delay is associated with a buffer-level threshold, because once the client-side buffer reaches the threshold, playback begins. Hence, the faster the client reaches this threshold (by downloading segments quickly), the lower the startup delay will be.

In contrast to models in previous work which make discrete predictions about startup delay (*i.e.*, good vs. bad), our model predicts startup delay as a continuous value. Thus, we can also study the magnitude of errors in our model prediction. Figure 4 shows the distribution of the relative errors for inferred startup delays split
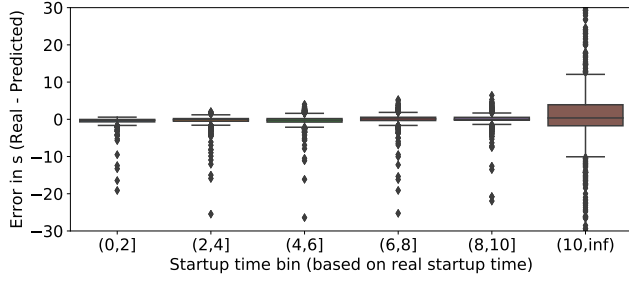
Figure 4: Relative error in startup delay inference (by levels) using Net+App features cut at [-30, 30] on the y axis (For all four video services).

(a) ROC.

(b) Precision-recall.

Figure 5: Resolution inference using different feature sets (For all four video services).

Figure 6: Feature importance for Netflix and YouTube service-specific models, for inferring video resolution.

into two-second bins. Our models perform particularly well for startup delays of less than ten seconds with errors mostly within 1 second. Above this threshold we see that the precision degrades. This behavior is due to two reasons: (1) 91.2% of all sessions in our data have startup delays of less than ten seconds, meaning that the number of samples in the greater than ten-second range is small; and (2) as startup delays increase beyond ten seconds, it becomes more difficult for the model to be as precise. In particular, we see in our dataset instances with values ranging from ten seconds to multiple minutes. While this eventuality should be taken into account in the design of any such model, also note that previous work has shown that startup delays above ten seconds can cause a rate of abandonment above 50% [23]. Hence, detecting that the startup delay has passed such threshold should be sufficient for an operator to identify a video session as poor quality.
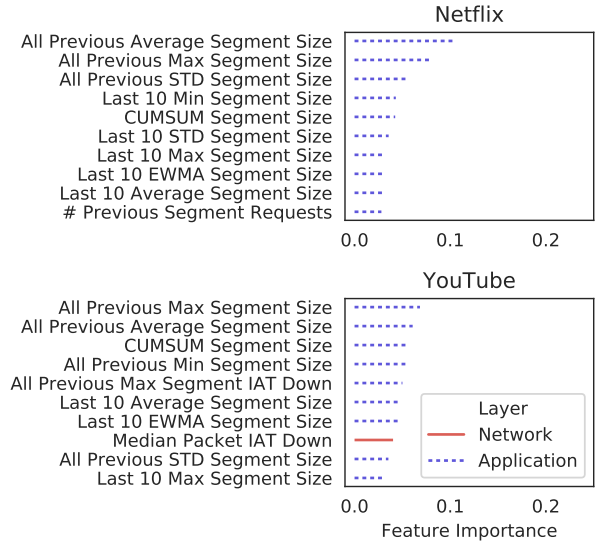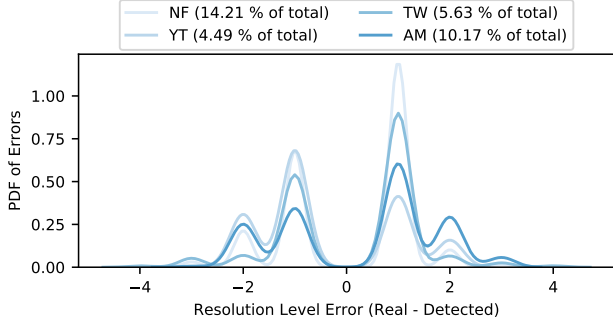
6.2 Resolution

Next, we explore resolution inference. We divide each video session into ten-second time intervals and conduct the inference on each time bin. We train one random forest multi-class classifier for each service feature set and service combination. Each class of the classifier corresponds to one of the following resolution values: 240p, 360p, 480p, 720p, and 1080p. Similarly to the previous section, we present aggregated results for the different layers averaging across the studied services. We report the receiver operating characteristic (Figure 5a) and precision-recall (Figure 5b) curves for the models weighted average across the different labels to illustrate the performance of the classifier for different values of the detection threshold. The models trained with application layer features consistently achieve the best performance with both precision and recall reaching 91% for a 4% false positive rate. Any model not including application features reduces precision by at least 8%, while also doubling the false-positive rate.

We next investigate the relative importance of features in Net+App model. Figure 6 reports the results for YouTube and Netflix, which shows that most features are related to segment-size statistics. The same applies to the other services. This result confirms the general intuition, since higher resolution means higher pixels-per-inch, thereby requiring more data to be delivered per video segment. Models trained with network and transport layer features infer resolution by relying on attributes related to byte
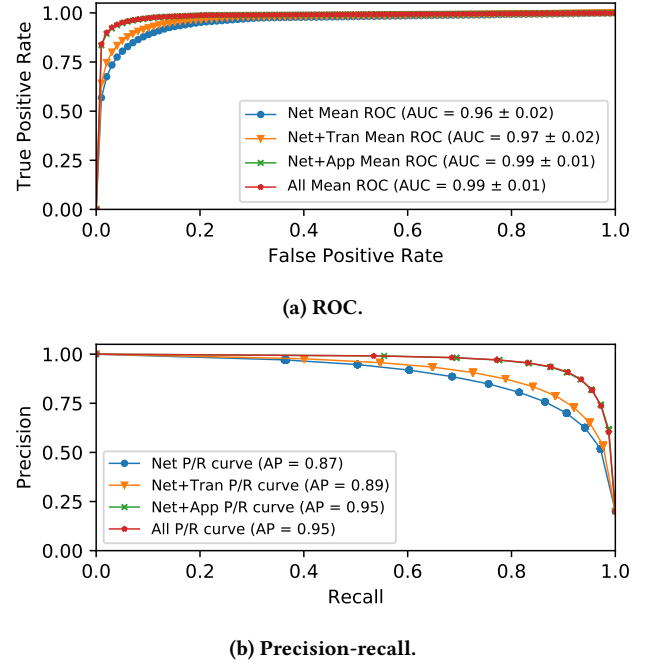
count, packet count, bytes-per-packet, bytes-in-flight, and throughput. Without segment-related features, these models achieve comparatively lower precision and recall.

We also quantify the error rate for resolution inference. Figure 7 shows the kernel density estimation obtained plotting relative

Figure 7: Kernel Density Estimation function for the relative error in resolution inference (normalized to levels).

errors for the time slots for which the model infers an incorrect resolution. Recall that we model resolution using a multi-class classifier, resulting in discrete resolution "steps." Hence, the error corresponds to distance in terms of the number of steps away from the correct resolution. As with the startup delay, errors tend to be centered around the real value, with the vast majority of errors falling within one or two resolution steps away the ground truth. For example, Netflix is the service with the higher number of mistakingly inferred time slots at 14%, but 83.9% of these errors are only one step away. Even given these error rates, this model still offers a better approximation of actual video resolution than previous models that only reflect coarse quality (*i.e.*, good vs. bad).

6.3 Rebuffering

When inferring rebuffering events, our model aims to infer, in every time bin, whether at least one rebuffering event took place. As we conduct our experiments, we observe that only YouTube video sessions experienced rebuffering in our dataset. Amazon, Twitch, and Netflix appear to avoid rebuffering by employing a much larger client-side application buffer. Hence, the results reported in Figures 8a and 8b are for YouTube sessions only. We observe that the model trained with Net+App features achieves the best performance. False positive rate and precision are comparable or better to the other metrics (0.7% and 84.9% respectively), yet recall is only 60%. This result is still better than BUFFEST [22], which achieved 31% precision and 10% recall. This improvement results from the use of segment-level information, as opposed to packet-level information used in BUFFEST [22]. Mazhar and Shafiq [26] report higher recall than our model, but we believe that this discrepancy is due to an artifact of their dataset, which includes an abnormally high frequency of rebuffering events: one out of four slots in their dataset experiences rebuffering, which is a much higher prior probability for rebuffering than would occur in real-world settings. We do not have access to their dataset. In reviewing their experiment setup, we observed that the network conditions in these experiments used both higher throughputs and higher packet loss rates (uniformly distributed between 0 and 100%). Such settings would lead YouTube to more easily move to HD quality while subsequently experiencing an abnormally high frequency of rebuffering events.
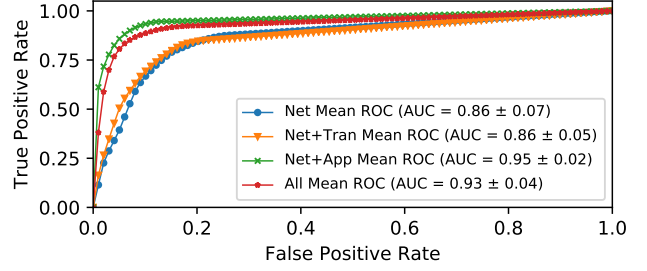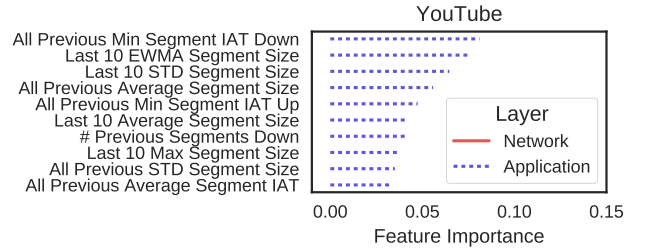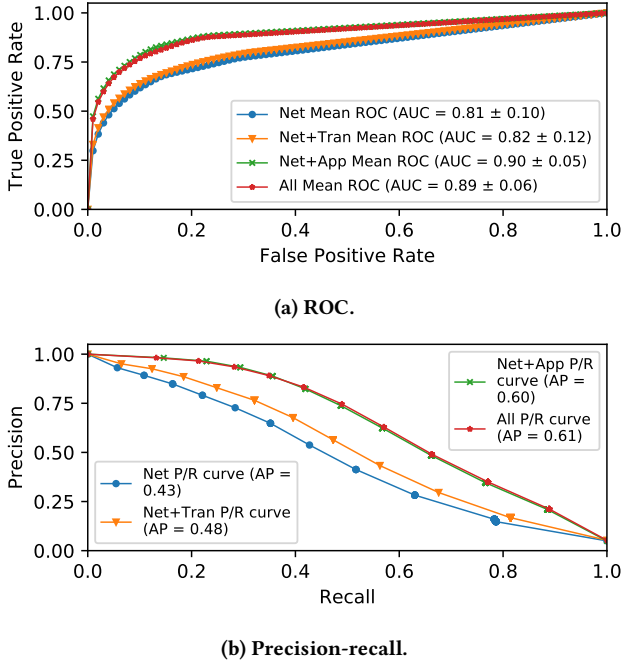

(a) ROC.
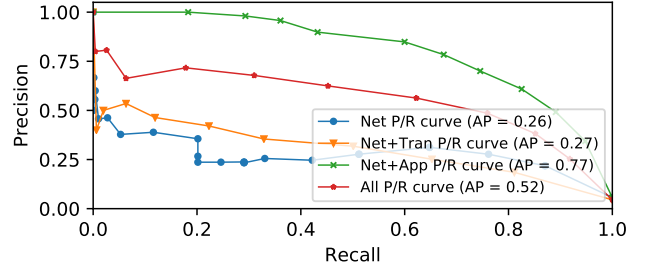
(b) Precision-recall.

Figure 8: Inferring rebuffering.

Figure 9: Rebuffering presence feature importance for YouTube service-specific model.

Figure 9 shows that the most important features are those associated with segment size, interarrival times, and total number of downloads. Intuitively, these features capture the draining of the buffer. As the interarrival time increases and the number and size of downloaded segments decreases, the client-side buffer is drained at a rate that is faster than the buffer filling rate.

Finally, we seek to address whether the low recall achieved by the produced model could excessively impact its usage in real-world scenarios. In particular, we aim to understand if false negatives could prevent the user of the model from detecting presence of rebuffering in video sessions. We then evaluate how far false negatives are from a real rebuffering event; *i.e.* for every time slot where the model does not detect an existing rebuffering event, we examine the nearby time slots until we find a predicted rebuffering event. We

(a) ROC.

(b) Precision-recall.

Figure 10: Inferring resolution switches (For all four video services).

Figure 11: Feature importance for predicting resolution switches in the service-specific Netflix and YouTube models.
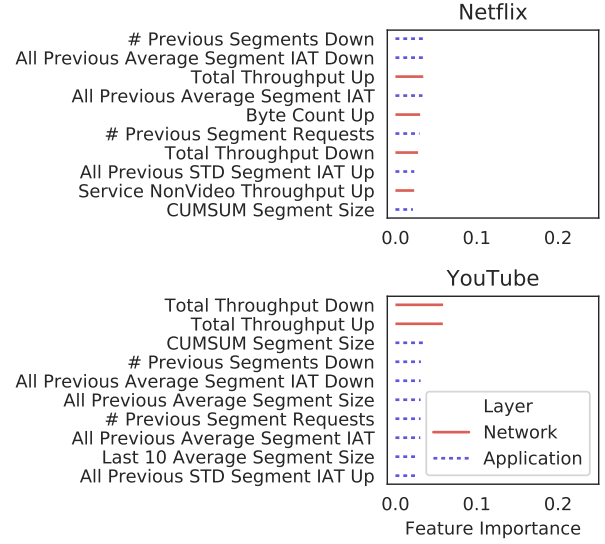
measure the time that elapses between these two slots. Our results (not shown for lack of space) show that 26% of false-negatives occur a single time slot away from a real rebuffering event (*i.e.* at most 10 seconds away). Further, 41.6% of them are at most 30 seconds away. Allowing these relative errors would produce recall values of 78.9% and 91.9%, respectively.

6.4 Resolution Switches

Finally, we explore models to infer resolution switches. We divide video sessions into ten-second time intervals, and train random forest models to infer if at least one resolution switch occurred in each time bin. A resolution switch can either refer to a switch from a lower to a higher video resolution or vice-versa. Figures 10a and 10b show the results. As in previous sections, the Net+App model and the model that uses all features (All) are the most accurate. We see that the average precision to detect resolution switches, however, is only 61%, even when using all features.

Figure 11 studies the ten most important features in the Net+App model for Netflix and YouTube. In contrast to the previous metrics, these models lack the presence of more dominant features that naturally reflect behavior associated with resolution switches. We believe that this lack of dominant features for resolution switch inference reflects the intricacy of ABR algorithms that rely on a mix of throughput estimate, buffer occupancy, and various parameters which may even be re-tuned as network condition changes[4].

Overall, we see that our inference methods for resolution switches performs relatively poorly compared with the other quality metrics. Given that we find features are generally similarly

important, and that resolution switches can be caused by conditions that are more nuanced compared with the other metrics, we argue that resolution switches may call for a different approach to reliably infer. We leave this for future exploration. Although, we plan to continue the investigation to achieve higher accuracy, we believe that by relying on the four target quality metrics together, we can improve the overall precision and recall to identify instances of poor video quality.

7 MAKING INFERENCE GENERAL

We now develop and evaluate a general model that infers video quality metrics across four popular video services: YouTube, Netflix, Amazon Instant Video, and Twitch. We present results for startup delay, resolution, and resolution switches. Because the only video service that experienced rebuffering events in our dataset is YouTube, we do not consider rebuffering in this section. For each service, we compare the accuracy of the Net+App model trained with data using only that service against its accuracy when trained with data from all services. We find that combined models perform nearly as well as models that rely only on sessions from a single service across all quality metrics. This result suggests that our lightweight models generalize to a wide variety of video streaming services.

Startup Delay We first evaluate a general model for inferring startup delay. Figure 12 reports startup delay inference errors for Netflix split into two-second intervals for the model trained using solely Netflix sessions (Same-NF) and the one obtained by using our entire dataset (Combined-NF). The general model performs almost as well as the one obtained from tailored training of a regressor that is specific to Netflix. In particular, for startup delays between two and ten seconds, both models achieve errors mostly within one second. The biggest discrepancies occur in the remaining two
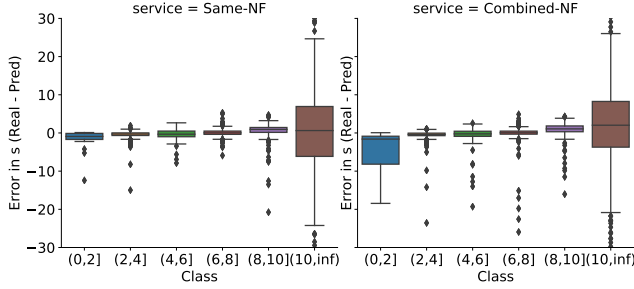
Figure 12: Startup delay inference error difference for Netflix, general versus Netflix model (aggregated into two-second time intervals, based on real startup delay).
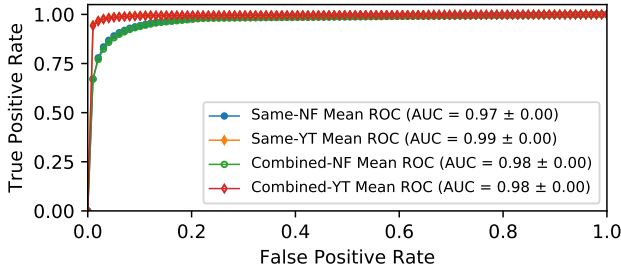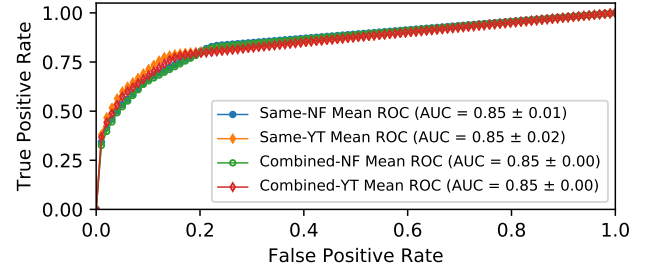

(a) ROC.
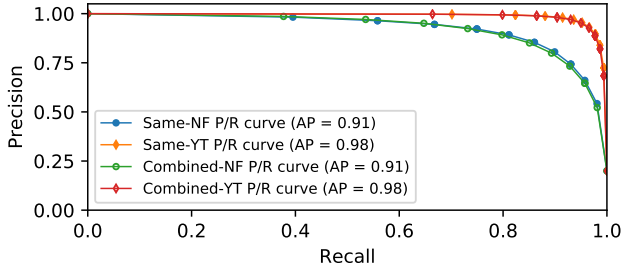
(b) Precision-recall.

Figure 13: Inferring resolution, general model.

classes, which have the lowest number of startup delay instances for Netflix, whereas other services experience more startup delays in those groups. In particular, YouTube has twice as many instances of less than two seconds startup delays (27% of them compared to 14% for all the other services). The most important features for the produced model include the cumulative segment sizes and their size distribution, two metrics that reflect the interarrival times for segment completions, and the average throughput achieved. These results are consistent with our service-specific analysis in Section 6.1. In summary, these results suggest it is possible to predict startup delay with a general model across multiple services.

Resolution. Figures 13a and 13b show results for Netflix and YouTube comparing general and service-specific models for inferring resolution. The general model performs nearly as well as the

(a) ROC.
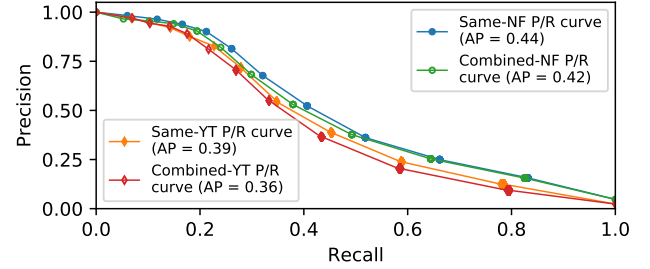
(b) Precision-recall.

Figure 14: Inferring resolution switches, general model.

per-service models across all of the video services. The four most important features are all related to segment size. The weighted average and standard deviation of the sizes of the last ten downloaded segments along with the maximum and average segment sizes over all past observed segments are most important. Recall from Section 6.2 that the top features for each service-specific model reflect the same pattern. In summary, the results reflect a general observation that segment sizes tend to vary across different resolutions, providing a solid basis for a general prediction model based on features related to segments.

Resolution Switches. Figures 14a and 14b show that a general model achieves similar accuracy as per-service models for resolution switches. The precision decreases by less than 1%, while the recall decreases by no more than 2.5%, with an associated increase in the false positive rate of less than 1%. Amazon and Twitch show similar results, although the general models are slightly less accurate in these cases, likely due to the general challenges in inferring resolution switches for YouTube and Netflix, as described earlier Section 6.4. Nevertheless, even for resolution switches, which are difficult to infer, a general model is not significantly less accurate than the service-specific models.

8 RELATED WORK

Previous work has thoroughly explored video quality, as well as how different video quality metrics relate to user quality of experience [21]. The methods to monitor these metrics, however, vary according to the vantage point. Streaming providers can directly

11

instrument the player to report video quality [3, 7, 15, 23], but one cannot directly measure quality from within the network. Therefore, ISPs must infer video quality based on the observation of the packets traversing the network. A variety of approaches have been developed for modeling one or more video quality metrics or quality of experience from network characteristics [10, 31, 37, 38, 41]. These models pre-date the wide-adoption of DASH. Hence, in practice, ISPs mostly rely on deep-packet inspection to identify video sessions within network traffic and then extract video quality [24, 33, 40], which is becoming infeasible due to encryption of video sessions [5, 17]. This is the main challenge we address in this paper.

A number of prior studies also address the problem of inferring video quality from encrypted video traffic as we do in this paper. We already discussed the most relevant methods in Section 3. Dimopoulos et al. [14] developed a method that runs on a web proxy in the network to infer quality of YouTube sessions. Their study only focuses on YouTube over TCP, so it is unclear whether the methods generalize to other video services or even to YouTube over QUIC (which is soaring at astonishing pace [28]). Second, they infer video quality metrics of full sessions, which is too coarse for some use cases such as online monitoring and optimization. Finally, the use of a web proxy raises privacy concerns. eMIMIC [25] improves on the accuracy of Dimopoulos et al. [14]. It relies on video segment detection to build parametric models of video session properties that translate into quality metrics. eMIMIC was evaluated for two undisclosed video services, where the model parameters had to be fine-tuned per service. Our work provides finer-grained inference targets and work per time slot.

Our approach of inferring video quality per time slots is most similar to the works by Mazhar and Shafiq [26] and BUFFEST [22]. BUFFEST [22] focuses only on rebuffering detection for YouTube. It introduces the method to identify video segments from encrypted traffic that we leverage to compute application-layer features. BUFFEST implements a buffer level estimation emulator. It combines the estimated buffer levels with network layer metrics to feed a machine learning model and predict the presence of rebuffering at periodic time slots. Our rebuffering models achieve higher accuracy than BUFFEST's. Mazhar and Shafiq [26] use decision trees to detect rebuffering events in real-time, whether video quality is high or low, and whether the video session has started or not. They only evaluate their model for YouTube (although both for HTTPS and QUIC), and our evaluation in Section 3 shows that it does not generalize to the other video services we tested. Their models take as input network- and transport-level features, which add a level of indirection to video quality. Our work demonstrates that the use of input features computed over video segments can outperform transport features reducing the collection cost for the operator. Moreover, our reliance on video segments helps generalize the method for different transport protocols and video services because we are removing transport overhead and any differences in the mapping of segments into packets. Video segments are the atomic unit of DASH video streaming and hence we can model video quality at finer granularity. In contrast to any of the existing work, our work evaluates a wide range of input features and shows that we can obtain accurate, general models to compute target quality metrics online with lightweight features.

9 CONCLUSION

In this paper, we developed and evaluated lightweight, general models to infer streaming video quality from encrypted traffic. We first demonstrated that existing models for inferring video quality from encrypted traffic that are trained and evaluated only for YouTube are inaccurate when applied to a broader class of video services and not practical for on-path operation in real-world network deployments. Using these findings as motivation, we then developed models that: (1) rely on only lightweight features of the network traffic, towards facilitating on-path, real-time inference of video quality; (2) can accurately infer video quality metrics across four popular Internet streaming video services: Netflix, YouTube, Amazon Instant Video, and Twitch. We develop and evaluate our models using more than 11,000 video sessions gathered from both laboratory settings and real-world home network deployments. Ultimately, we develop and present a model that accurately infers video quality metrics across all four of these video services, while relying on only lightweight features. In addition to simply reporting model accuracy, we also explore the relative feature importance in each of our models, lending intuition for *why* certain models perform—and generalize—better than others.

Our results suggest many immediate avenues for future work. First, the ability for a general model to accurately predict many video quality metrics based only on lightweight features in encrypted traffic suggests that it may indeed be possible to implement and deploy such a model at various vantage points in the network, from the home network to the backbone. A clear avenue for future work involves operationalizing models that can operate at line rate, leveraging programmable network devices as needed to measure, extract, and compute model features. Second, it is clear that more work is needed to develop a general model to detect resolution switches in video. Finally, our models all initially identify video traffic from the DNS QNAME; currently, DNS queries are transmitted in cleartext, but with possible migration to DNS-over-TLS and DNS-over-HTTPS, another possible future avenue could involve coupling our work with future work to identify video traffic flows without the benefit of cleartext DNS queries.

REFERENCES

[1] Anne Aaron, Z Li, M Manohara, J De Cock, and D Ronca. 2015. Per-title encode optimization. *The Netflix Techblog* (2015).

[2] Vijay Kumar Adhikari, Yang Guo, Fang Hao, Matteo Varvello, Volker Hilt, Moritz Steiner, and Zhi-Li Zhang. 2012. Unreeling Netflix: Understanding and improving multi-CDN movie delivery. In *INFOCOM*. Orlando, FL.

[3] Adnan Ahmed, Zubair Shafiq, Harkeerat Bedi, and Amir Khakpour. 2017. Suffering from buffering? Detecting QoE impairments in live video streams. In *International Conference on Network Protocols (ICNP)*. Toronto, Canada.

[4] Zahaib Akhtar, Yun Seong Nam, Ramesh Govindan, Sanjay Rao, Jessica Chen, Ethan Katz-Bassett, Bruno Ribeiro, Jibin Zhan, and Hui Zhang. 2018. Oboe: auto-tuning video ABR algorithms to network conditions. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*. ACM, 44–58.

[5] GSM Association. 2015. Network Management of Encrypted Traffic: Version 1.0. https://www.gsma.com/newsroom/wp-content/uploads/WWG-04-v1-0.pdf.

[6] A. Balachandran, V. Sekar, A. Akella, S. Seshan, I. Stoica, and H. Zhang. 2012. A quest for an internet video quality-of-experience metric. In *HotNets*. Redmond, WA.

[7] Athula Balachandran, Vyas Sekar, Aditya Akella, Srinivasan Seshan, Ion Stoica, and Hui Zhang. 2013. Developing a predictive model of quality of experience for internet video. *ACM SIGCOMM Computer Communication Review* 43, 4 (2013), 339–350.

[8] Leo Breiman. 2017. *Classification and regression trees*. Routledge.

[9] Martin Brown. 2006. Traffic Control HOWTO. http://tldp.org/HOWTO/Traffic-Control-HOWTO/index.html.

[10] K-T Chen, C-C Tu, and W-C Xiao. 2009. Oneclick: A framework for measuring network quality of experience. In *INFOCOM*. Rio de Janeiro, Brazil.

[11] Chrome web 2018. Chrome webRequest API. https://developer.chrome.com/extensions/webRequest.

[12] Chromedriver 2018. ChromeDriver - WebDriver for Chrome. https://sites.google.com/a/chromium.org/chromedriver/.

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

[14] Giorgos Dimopoulos, Ilias Leontiadis, Pere Barlet-Ros, and Konstantina Papagiannaki. 2016. Measuring video QoE from encrypted traffic. In *Proceedings of the 2016 Internet Measurement Conference*. ACM, 513–526.

[15] Florin Dobrian, Vyas Sekar, Asad Awan, Ion Stoica, Dilip Joseph, Aditya Ganjam, Jibin Zhan, and Hui Zhang. 2011. Understanding the impact of video quality on user engagement. *ACM SIGCOMM Computer Communication Review* 41, 4 (2011), 362–373.

[16] Driving Engagement 2017. Driving Engagement for Online Video. http://events.digitallyspeaking.com/akamai/mddec10/post.html?hash=ZDlBSGhsMXBidnJ3RXNWSW5mSE1HZz09.

[17] Keith Dyer. 2015. How encryption threatens mobile operators, and what they can do about it. http://the-mobile-network.com/2015/01/how-encryption-threatens-mobile-operators-and-what-they-can-do-about-it/.

[18] Tobias Hoßfeld, Michael Seufert, Christian Sieber, and Thomas Zinner. 2014. Assessing effect sizes of influence factors towards a QoE model for HTTP adaptive streaming. In *Quality of Multimedia Experience (QoMEX)*. Singapore.

[19] T. Huang, R. Johari, N. McKeown, M. Trunnell, and M. Watson. 2014. A buffer-based approach to rate adaptation: Evidence from a large video streaming service. In *ACM SIGCOMM*. Chicago, IL.

[20] ITU 2012. ITU Methodology for the subjective assessment of the quality of television pictures. Recommendation ITU-R BT.500-13.

[21] Parikshit Juluri, Venkatesh Tamarapalli, and Deep Medhi. 2016. Measurement of Quality of Experience of Video-on-Demand Services: A Survey. *IEEE Communications Surveys & Tutorials* 18, 1 (2016).

[22] Vengatanathan Krishnamoorthi, Niklas Carlsson, Emir Halepovic, and Eric Petajan. 2017. BUFFEST: Predicting Buffer Conditions and Real-time Requirements of HTTP(S) Adaptive Streaming Clients. In *MMSys'17*. Taipei, Taiwan.

[23] S Shunmuga Krishnan and Ramesh K Sitaraman. 2013. Video stream quality impacts viewer behavior: inferring causality using quasi-experimental designs. *IEEE/ACM Transactions on Networking* 21, 6 (2013), 2001–2014.

[24] Tarun Mangla, Emir Halepovic, Mostafa Ammar, and Ellen Zegura. 2017. MIMIC: Using passive network measurements to estimate HTTP-based adaptive video QoE metrics. In *Network Traffic Measurement and Analysis Conference (TMA)*. Dublin, Ireland.

[25] Tarun Mangla, Emir Halepovic, Mostafa Ammar, and Ellen Zegura. 2018. eMIMIC: Estimating HTTP-based Video QoE Metrics from Encrypted Network Traffic. In *Network Traffic Measurement and Analysis Conference (TMA)*. Vienna, Austria.

[26] M. Hammad Mazhar and Zubair Shafiq. 2018. Real-time Video Quality of Experience Monitoring for HTTPS and QUIC. In *INFOCOM*. Honolulu, HI.

[27] Abhijit Mondal, Satadal Sengupta, Bachu Rikith Reddy, MJV Koundinya, Chander Govindarajan, Pradipta De, Niloy Ganguly, and Sandip Chakraborty. 2017. Candid with YouTube: Adaptive Streaming Behavior and Implications on Data Consumption. In *NOSSDAV'17*. Taipei, Taiwan.

[28] Openwave Mobility. 2018. Mobile Video Index. https://landing.owmobility.com/mobile-video-index/.

[29] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. 2011. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research* 12 (2011), 2825–2830.

[30] pytube 2018. pytube - A lightweight, dependency-free Python library for downloading YouTube Videos. https://github.com/nficano/pytube.

[31] Amy R Reibman, Vinay A Vaishampayan, and Yegnaswamy Sermadevi. 2004. Quality monitoring of video over a packet network. *IEEE Transactions on Multimedia* 6, 2 (2004), 327–334.

[32] Sandvine. 2015. Global Internet Phenomena Spotlight: Encrypted Internet Traffic. https://www.sandvine.com/hubfs/downloads/archive/global-internet-phenomena-spotlight-encrypted-internet-traffic.pdf.

[33] Sandvine. 2015. Internet Traffic Classification: A Sandvine Technology Showcase. https://www.sandvine.com/hubfs/downloads/archive/technology-showcase-internet-traffic-classification.pdf.

[34] Sandvine. 2016. 2016 Global Internet Phenomena—Latin America & North America. https://www.sandvine.com/hubfs/downloads/archive/2016-global-internet-phenomena-report-latin-america-and-north-america.pdf.

[35] Iraj Sodagar. 2011. The MPEG-DASH standard for multimedia streaming over the internet. *IEEE MultiMedia* 18, 4 (2011), 62–67.

[36] T. Stockhammer. 2011. Dynamic adaptive streaming over HTTP: standards and design principles. In *ACM Conference on Multimedia Systems (MMSys '11)*. San Jose, CA.

[37] Klaus Stuhlmuller, Niko Farber, Michael Link, and Bernd Girod. 2000. Analysis of video transmission over lossy channels. *IEEE Journal on Selected Areas in communications* 18, 6 (2000), 1012–1032.

[38] Shu Tao, John Apostolopoulos, and Roch Guérin. 2008. Real-time monitoring of video quality in IP networks. *IEEE/ACM Transactions on networking* 16, 5 (2008), 1052–1065.

[39] Tcpdump 2017. tcpdump - dump traffic on a network. https://www.tcpdump.org/manpages/tcpdump.1.html.

[40] Martino Trevisan, Idilio Drago, and Marco Mellia. 2016. Impact of Access Speed on Adaptive Video Streaming Quality: A Passive Perspective. In *Proceedings of the 2016 Workshop on QoE-based Analysis and Management of Data Communication Networks (Internet-QoE '16)*. Florianopolis, Brazil.

[41] Mukundan Venkataraman and Mainak Chatterjee. 2011. Inferring video QoE in real time. *IEEE network* 25, 1 (January 2011), 4–13.

[42] Xiaoqi Yin, Abhishek Jindal, Vyas Sekar, and Bruno Sinopoli. 2015. A control-theoretic approach for dynamic adaptive video streaming over HTTP. *ACM SIGCOMM Computer Communication Review* 45, 4 (2015), 325–338.

[43] Youtube API 2018. YouTube Player API Reference for iframe Embeds. https://developers.google.com/youtube/iframe_api_reference.