# Inferring Streaming Video Quality from Encrypted Traffic: Practical Models and Deployment Experience

FRANCESCO BRONZINO*†, Inria, France and Nokia Bell Labs, France
PAUL SCHMITT*, Princeton University, USA
SARA AYOUBI, Inria, France
GUILHERME MARTINS, University of Chicago, USA
RENATA TEIXEIRA, Inria, France
NICK FEAMSTER, University of Chicago, USA

Inferring the quality of streaming video applications is important for Internet service providers, but the fact that most video streams are encrypted makes it difficult to do so. We develop models that infer quality metrics (*i.e.*, startup delay and resolution) for encrypted streaming video services. Our paper builds on previous work, but extends it in several ways. First, the models work in deployment settings where the video sessions and segments must be identified from a mix of traffic and the time precision of the collected traffic statistics is more coarse (*e.g.*, due to aggregation). Second, we develop a single composite model that works for a range of different services (*i.e.*, Netflix, YouTube, Amazon, and Twitch), as opposed to just a single service. Third, unlike many previous models, our models perform predictions at finer granularity (*e.g.*, the precise startup delay instead of just detecting short versus long delays) allowing to draw better conclusions on the ongoing streaming quality. Fourth, we demonstrate the models are practical through a 16-month deployment in 66 homes and provide new insights about the relationships between Internet "speed" and the quality of the corresponding video streams, for a variety of services; we find that higher speeds provide only minimal improvements to startup delay and resolution.

CCS Concepts: • **Information systems** → **Multimedia streaming**; • **Networks** → **Network measurement**; *Network management*; • **Computing methodologies** → *Classification and regression trees.*

Additional Key Words and Phrases: DASH; Quality Inference; Encrypted Traffic; Network Measurements

**ACM Reference Format:**
Francesco Bronzino, Paul Schmitt, Sara Ayoubi, Guilherme Martins, Renata Teixeira, and Nick Feamster. 2019. Inferring Streaming Video Quality from Encrypted Traffic: Practical Models and Deployment Experience. *Proc. ACM Meas. Anal. Comput. Syst.* 3, 3, Article 56 (December 2019), 25 pages. https://doi.org/10.1145/3366704

*Both authors contributed equally to this research.
†Work done while at Inria.

Authors' addresses: Francesco Bronzino, francesco.bronzino@nokia-bell-labs.com, Inria, Paris, France, Nokia Bell Labs, Paris-Saclay, France; Paul Schmitt, pschmitt@cs.princeton.edu, Princeton University, Princeton, New Jersey, USA; Sara Ayoubi, sara.ayoubi@inria.fr, Inria, Paris, France; Guilherme Martins, gmartins@uchicago.edu, University of Chicago, Chicago, Illinois, USA; Renata Teixeira, renata.teixeira@inria.fr, Inria, Paris, France; Nick Feamster, feamster@uchicago.edu, University of Chicago, Chicago, Illinois, USA.


2476-1249/2019/12-ART56 $15.00
https://doi.org/10.1145/3366704

## 1 Introduction

Video streaming traffic is by far the dominant application traffic on today's Internet, with some projections forecasting that video streaming will comprise 82% of all Internet traffic in just two years [14]. 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, Internet Service Providers (ISPs) must infer video quality from traffic as it passes through the network. ISPs need to measure video streaming quality because it represents a more direct metric of customer experience than performance metrics typically extracted from network flows such as data rates. ISPs monitor video quality metrics to detect network issues that affect customer experience and mitigate problems before they generate user complaints. On longer timescales, trends in video quality can facilitate capacity planning. For example, when all clients in a neighborhood experience poor video quality, the ISP can plan to upgrade capacity for those users. Yet, monitoring video quality is not straightforward for ISPs: With end-to-end encryption becoming more common, as a result of increased video streaming content over HTTPS and QUIC [29, 32], ISPs cannot directly observe video quality metrics such as startup delay and video resolution from the video streaming protocol [5, 18]. The end-to-end encryption of the video streams thus presents ISPs with the challenge of inferring video quality metrics solely from properties of the network traffic that are directly observable.

Previous approaches infer the quality of a specific video service, typically using modeling and prediction that is based on an offline trace in a controlled laboratory setting [15, 23, 27]. Unfortunately, these models are often not directly applicable in practice because real-world networks (1) have other traffic besides the video streams themselves, creating the need to identify video services and sessions; (2) have multiple video services, as opposed to just a single one. Transferring the existing models to practice turns out to introduce new challenges due to these factors.

First, inference models must take into account the fact that real network traffic traces contain a mix of traffic. Often gathered at coarse temporal granularities due to monitoring constraints in production networks, video session traffic is mixed with non-video cross-traffic. In a real deployment, the models must then identify the video sessions accurately, especially given that errors in identifying applications can propagate to the quality of the prediction models. Second, the prediction models should apply to a range of services, which existing models tend not to do. A model that can predict quality across multiple services is hard to devise because both video streaming algorithms and content characteristics can vary significantly across video services (*e.g.*, buffer-based [21] versus throughput-based [35] rate adaption algorithm, fixed-size [21] versus variable-size video segments [28]).

This work takes a step towards making video inference models practical, tackling the challenges that arise when the models must operate on real network traffic traces and across a broad range of services. As a proof of concept that a general model that applies across a range of services in a real deployment can be designed and implemented, we studied four major streaming services—Netflix, YouTube, Amazon, and Twitch—across a 16-month period, in 66 home networks in the United States and France, comprising a total of 216,173 video sessions. To our knowledge, this deployment study is the largest public study of its kind for video quality inference.

We find that models that are trained across all four services (*composite* models) perform almost as well as the service-specific models that were designed in previous work, provided that the training data contains traffic from each of the services for which quality is being predicted. On the other hand, we fall short of developing a truly general model that can predict video quality for services that are not in the training set. An important challenge for future work will be to devise such a model. Towards the goal of developing such a model, we release our training data to the

community [1], which contains over 13,000 video sessions labeled with ground truth video quality, as a benchmark for video quality inference, so that others can compare against our work and build on it.

Beyond the practical models themselves, the deployment study that we performed as part of this work has important broader implications for both ISPs and consumers at large. We conducted this study in collaboration with *The Wall Street Journal* to understand the effect of home internet speeds on video streaming quality [38]. This deployment study found that the speeds that consumers purchase from their ISPs have considerably diminishing returns with respect to video quality. Specifically, Internet speeds higher than about 100 Mbps of downstream throughput offer negligible improvements to video quality metrics such as startup delay and resolution. This new result raises important questions for operators and consumers. Operators may focus on other aspects of their networks to optimize video delivery; at the same time, consumers can be more informed about what downstream throughput they actually need from their ISPs to achieve acceptable application quality.

The rest of the paper proceeds as follows. Section 2 provides background on streaming video quality inference and details the state of the art in this area; in this section, we also demonstrate that previous models do not generalize across a range of services. Section 3 details the video quality metrics we aim to predict, and the different classes of features that we use as input to the prediction models. Section 4 explains how we selected the appropriate prediction model for each video quality metric and how we train the models. Section 5 discusses the model validation. Section 6 characterizes the 16-month, 66-home deployment and the challenges we had to overcome to apply the models in practice. Section 7 presents the results obtained from the deployment study and Section 8 concludes with a summary and discussion of open problems and possible future directions.

## 2 Background and Related Work: Streaming Video Quality Inference

In this section, we first provide background on DASH video streaming. Then, we discuss the problem of streaming video quality inference that ISPs face, and the current state of the art in video quality inference.

Internet video streaming services typically use Dynamic Adaptive Streaming over HTTP (DASH) [34] to deliver a video stream. DASH divides each video into time intervals 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 servers (*e.g.*, a content delivery network) to allow a video client to download segments from a nearby server. At the beginning of a video session, a 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 [35], current buffer size [21], or a hybrid of the two [40] 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.

ISPs must infer video quality based on the observation of the packets traversing the network. The following video quality metrics affect a user's quality of experience [4, 6, 7, 16, 17, 22, 24]:

- *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.
- The *resolution* of a video is the number of pixels in each dimension of video frame.

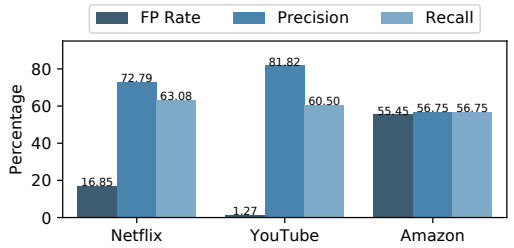



**Fig. 1.** *Using a per-service model to detect high (bigger than 5 seconds) startup delay using method from Mazhar and Shafiq [27].*

- The *bitrate* of a segment measures the number of bits transmitted over time.
- *Resolution switching* has a negative effect on quality of experience [6, 20]; both the amplitude and frequency of switches affect quality of experience [20].
- *Rebuffering* also increases rate of abandonment and reduces the likelihood that a user will return to the service [16, 24].

Resolution and bitrate can vary over the course of the video stream. Players may switch the bitrate to adapt to changes in network conditions with the goal to select the best possible bitrate for any given condition.

In the past, ISPs have relied on deep-packet inspection to identify video sessions within network traffic and subsequently infer video quality [25, 33, 39]. Yet, most video traffic is now encrypted, which makes it more challenging to infer video traffic [5, 18].

Previous work has attempted to infer video quality from encrypted traffic streams, yet has typically done so (1) in controlled settings, where the video traffic is the only traffic in the experiment; (2) for single services, as opposed to a range of services. It turns out that these existing methods do not apply well to the general case, or in deployment, for two reasons. First, in real deployments, video streaming traffic is mixed with other application traffic, and thus simply *identifying the video sessions* becomes a challenging problem in its own right. Any errors in identifying the sessions will propagate and create further errors when inferring any quality of experience features. Second, different video streaming services behave in different ways—video on demand services incur a higher startup delay for defense against subsequent rebuffering, for example. As such, a model that is trained for a specific service will not automatically perform accurate predictions for subsequent services.

Previous work has developed models that operate in controlled settings for specific services that often do not apply in deployment scenarios or for a wide range of services. Mazhar and Shafiq [27], BUFFEST [23], and Requet [19] infer video quality over short time intervals, which is closest to our deployment scenario. The method from Mazhar and Shafiq [27] is the most promising for our goals because they infer three video quality metrics: rebuffering events, video quality, and startup delay. In addition, their models are based on decision trees, so we can retrain them for the four services we study using our labeled data for each service, without the need to reverse engineering each individual service. BUFFEST [23] focused only on rebuffering detection for YouTube, but introduces a method for identifying video segments that we build on in this work. We observe in our dataset that rebuffering occurs in only a small fraction of video sessions, even under poor network conditions. Hence, we concentrate our work on the inference of resolution and startup delay, as they are meaningful to all video sessions.

The shortcomings of these previous approaches are apparent when we attempt to apply them to a wider range of video services. When we apply the models from Mazhar and Shafiq to the labeled datasets we collected with the four video services (described in Section 4.1) the accuracy is low. The models from Mazhar and Shafiq infer quality metrics at a coarse granularity: whether there was rebuffering, whether video quality is high or low, and whether the video session has started or not before a given time threshold. Here, we show results for the startup delay—whether the video started in more than five seconds after the video page has been requested. We choose five seconds based on previous findings that most users consider such startup time acceptable across different video service types [24]. We use Scikit-learn's [30] AdaBoost implementation with one hundred default weak learners as in the original paper. We train one model per service and evaluate how well we can detect long startup delays. Figure 1 shows the false positive rate (FPR), precision, and recall when testing the model for each video service. We omit results for Twitch as 99% of Twitch sessions start playing after five seconds, so predicting a startup delay that exceeds five seconds was not a meaningful exercise—it almost always happens.

Mazhar and Shafiq's method was ineffective at detecting high startup delay for the other services. False positive rates are around 20% for Netflix and even higher for Amazon. While perhaps a threshold different than five seconds would be more appropriate for Netflix and Amazon, this result highlights the difficulty in choosing such thresholds across services with different streaming logics. Instead of focusing on fine tuning the parameters of their model with the exact same set of features and inference goals from the original paper, we apply Mazhar and Shafiq's general approach but revisit the design space. In particular, we define inference goals that are more fine-grained (*e.g.*, inference of the exact startup delay instead of whether the delay is above a certain threshold) and we leverage BUFFEST's method to identify video segments from encrypted traffic to consider a larger set of input features.

In addition to problems with accuracy, many previous models have problems with *granularity*. Specifically, many methods infer video quality metrics of *entire video sessions* as opposed to continuous estimates of video sessions over shorter time intervals, as we do in this work. Continuous estimates of video quality are often more useful for troubleshooting, or other optimizations that could be made in real-time. Dimopoulos *et al.* [15] relied on a web proxy in the network to inspect traffic and infer quality of YouTube sessions; it is apparent that they may have performed man-in-the-middle attacks on the encrypted traffic to infer quality. eMIMIC [26] relies on the method in BUFFEST [23] to identify video segments and build parametric models of video session properties that translate into quality metrics. This model assumes that video segments have a fixed length, which means that it cannot apply to streaming services with variable-length segments such as YouTube. Requet [19] extended BUFFEST's video segment detection method to infer potential buffer anomalies and resolution at each segment download for YouTube. These models rely on buffer-state estimation and are difficult to apply to other video services, because each video service (and client) has unique buffering behavior, as the service may prioritize different features such as higher resolution or short startup delays.

## 3 Metrics and Features

In this section, we define the problem of video quality inference from encrypted network traffic. We first discuss the set of video quality metrics that a model should predict, as well as the granularity of those predictions. Then, we discuss the space of possible features that the model could use.

### 3.1 Target Quality Metrics

We focus on building models to infer startup delay and resolution. Prior work has focused on bitrate as a way to approximate the video resolution, but the relationship between bitrate and resolution

| 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 1.** *Summary of the extracted features from traffic.*

is complex because the bitrate also depends on the encoding and the content type [3]. Resolution switches can be inferred later from the resolution per time slot. We omit the models for rebuffering as our dataset only contains rebuferring for YouTube. We present rebuffering models for YouTube in a technical report [9].

**Startup delay.** Prior work presents different models of startup delay. eMIMIC [26] defines startup delay as the delay to download the first segment. This definition can be misleading, however, because most services download multiple segments before starting playback. For example, we observe in our dataset that Netflix downloads, on average, four video segments and one audio segment before playback begins. Alternatively, Mazhar and Shafiq [27] rely on machine learning to predict startup delay as a binary prediction [27]. The threshold to classify a startup delay as short or long, however, varies depending on the nature of the service, (*e.g.*, roughly three seconds for short videos on a service like YouTube and five seconds for movies on a service such as Netflix). Therefore, we instead use a regression model to achieve finer inference granularity and generalize across video services.

**Resolution.** Resolution often varies through the course of a video session; therefore, the typical inference target is the average resolution. Previous work has predominantly used classification approaches [15, 27]: either binary (good or bad) [27] or with three classes (high definition, standard definition, and low definition) [15]. Such a representation can be misleading, as different clients may have different hardware configurations (*e.g.*, a smartphone with a 480p screen compared to a 4K smart television). Therefore, our models infer resolution as a multi-class classifier, which is easier to analyze. Each class of the classifier corresponds to one of the following resolution values: 240p, 360p, 480p, 720p, and 1080p.

One can compute the average resolution of a video session [15, 26] or of a time slot [27]. We opt to infer resolution over time as this fine-grained inference works for online monitoring plus it can later be used to recover per-session statistics as well as to infer the frequency and amplitude of resolution switches. We consider bins of five and ten seconds or at the end of each downloaded video segment. To determine the bin size, we study the number of resolution switches per time bin using our ground truth. The overwhelming majority of time bins have no bitrate switches. For example, at five seconds, more than 93% of all bins for Netflix have no quality switch. As the bin size increases, more switches occur per bin and resolution inference occurs on a less precise timescale. 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 ten-second bins, which represents a good tradeoff between the precision of inferring resolution and the likelihood that each time bin contains complete video segment downloads.

### 3.2 Features

For each video session, we compute a set of features from the captured traffic at various levels of the network stack, as summarized in Table 1. We consider a super-set of the features used in prior models [15, 26, 27] to evaluate the sub-set of input features that provides the best inference accuracy.

**Network Layer.** We define network-layer features as metrics that solely rely on information available from observation of a network flow (identified by the IP/port four-tuple).

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.

**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. For the metrics in Table 1, we compute summary statistics such as mean, median, maximum, minimum, standard deviation, kurtosis, and skewness.

**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 [23] 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 TCP and QUIC video traffic. Section 6.2.2 provides an in depth analysis of the methodology and the accuracy of our implementation of such techniques.

We use sequences of inferred video segment downloads to build up the feature set for the application layer. For each one of the features in Table 1 we compute the following statistics: minimum, mean, maximum, standard deviation and 50th, 75th, 85th, and 90th percentile.

## 4 Model Selection and Training

We describe how we gathered the inputs for the prediction model, how we selected the prediction model for startup delay and resolution, and the process for training the final regressor and classifier.

### 4.1 Inputs and Labeled Data

**Gathering input features.** We train models considering different sets of input features: 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 32 models in total: (1) varying across these six feature sets and (2) using six different datasets, splitting the dataset with sessions from each of the four video services—Netflix, YouTube, Amazon, and Twitch—plus two combined datasets, one with

sessions from all services (which we call *composite*) and one with sessions from three out of four services (which we call *excluded*). 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, and these models are less accurate, in any case.

**Labeling: Chrome Extension.** To label these traffic traces with the appropriate video quality metrics, we developed a Chrome extension that monitors application-level information for the four services. This extension, which supports any HTML 5-based video, allowed us to assign video quality metrics to each stream as seen by the client. We make the extension and the tools to generate and label video sessions available as open source software [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 used the Chrome browser API to inspect the URL of every page and identify pages reproducing video for each of the video services of interest. After the extension identifies the page, collection is tailored for each service:

- *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 injected 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 adjusted our collection period accordingly. Netflix reports a variety of statistics. We focused 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 resolution.
- *YouTube: iframe API.* We used the YouTube iframe API [41] to periodically extract player status information, including current video resolution, available playback buffer (in seconds) and current 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.
- *Twitch and Amazon: HTML 5 tag parsing.* As the two services expose no proprietary interface, we generalized the module developed for YouTube to solely rely on the *<video> HTML5* tag to collect all the required data. This approach allowed 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.

### 4.2 Training

**Startup delay.** We trained on features computed from the first ten seconds of each video session. We experimented with different regression methods, including: linear, ridge, SVR, decision tree regressor, and random forest regressor. We evaluate methods based on the average absolute error and conclude that random forest leads to lowest errors. We select the hyper parameters of the random forest models on the validation set using the R2 score to evaluate all combinations with exhaustive grid search.

**Resolution.** We trained a classifier with five classes: 240p, 360p, 480p, 720p, and 1080p. We evaluated Adaboost (as in prior work [27]), logistic regression, decision trees, and random forest. We select random forests because it again gives higher precision and recall with lower false positive rates.

| Service | Total Runs | % Home | % Lab |
|---|---|---|---|
| **Netflix** | 3,861 | 53% | 47% |
| **YouTube TCP** | 4,511 | 16% | 74% |
| **YouTube QUIC** | 1,310 | 58% | 42% |
| **Twitch** | 2,231 | 17% | 83% |
| **Amazon** | 1,852 | 10% | 90% |

**Table 2.** *Summary of the labeled dataset.*

Similarly to the model for startup delay, we select hyper parameters using exhaustive grid search and select the parameters that maximize the F1 score.

**Generating and labeling traffic traces.** 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 large European city with download speeds of 100 Mbps, 18 Mbps, and 6 Mbps, respectively; one room in a student residence in the same city; one apartment in a university campus 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. We generated video sessions automatically using `ChromeDriver` [13]. We played each session for 8 to 12 minutes depending on the video length and whether there were ads at the beginning of the session (in contrast to previous work [27], we did *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 varied 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. We randomly selected Netflix and Amazon movies from the suggestions presented by each service in the catalog page. To avoid bias, we selected movies from different categories including action, comedies, TV shows, and cartoons. We ultimately selected 25 movies and TV shows used in rotation from Netflix and 15 from Amazon. 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. To allow Netflix to stream 1080p content on Chrome, we installed a dedicated browser extension [12]. We collected packet traces using `tcpdump` [37] on the network interface the client uses and compute traffic features presented in the previous section. Finally, for each session we also collect ground truth browsing history through a browser plugin and build time series of downloaded video/audio segments during each experiment. We isolate individual segment downloads by inspecting the URLs of the web requests.

**Emulating Network Conditions.** In the lab environment, we manually varied the network conditions in the experiments using `tc` [10] to ensure that our training datasets captured 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 50 kbps to 30 Mbps, and introduced 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.

**Measurement period.** We collected data from November 20, 2017 to May 3, 2019. 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. Hence, we removed all sessions that required more than a high threshold, *i.e.*, 30 seconds, to start reproducing the video. The resulting dataset contains 13,765 video sessions

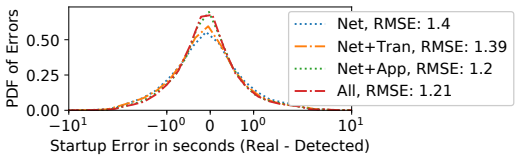



**Fig. 2.** *Startup delay inference error across different feature sets.*

from Netflix, Amazon, Twitch, and YouTube, that we use to train and test our inference models. [1] Table 2 shows the number of runs per video service and under the different network conditions.

## 5 Model Validation

We evaluate the accuracy of models relying on different feature sets for predicting startup delay and resolution. Our results show that models that rely on network- and application-layer features outperform models that rely on network- and transport-layer features across all services. This result is in contrast with prior work [15, 27], which provided models that rely on transport-layer features.

We also evaluate whether our models are general. A composite model—where we train the model with data from multiple services and later predict quality of any video service—is ideal as it removes the requirement to collect data with ground truth for a large number of services. Our evaluation of a *composite model* trained with data from the four video services shows that it performs nearly as well as *specific models* that rely only on sessions from a single service across both quality metrics. This result raises hopes that the composite model can generalize to a wide variety of video streaming services. When we train models using a subset of the services and evaluate it against the left out one (*excluded models*), however, the accuracy of both startup delay and resolution models degrades significantly, rendering the models unusable. This result highlights that although our modeling method is general in that it achieves good accuracy across four video services, the training set used to infer quality metrics should include all services that one aims to do prediction for.

### 5.1 Startup Delay

To predict startup delay, we train one random forest regressor for each feature set and service combination. We study the magnitude of the inference errors in our model prediction. Figure 2 presents the error of startup delay inference across the four services for different feature sets. Interestingly, our results show that the model using a combination of features from network and application layer yields the highest precision, minimizing the root mean square error (RMSE) across the four services. These results also show that we can exclude Net+Tran models, because Net+App models have consistently smaller errors. ISPs may ultimately choose a model that uses only network-layer features, which rely on features that are readily available in many monitoring systems for a small decrease in inference accuracy.

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 ranking with predominantly features that indicate how fast the client downloads segments, such as the cumulative segment size and their interarrival 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

[1]Dataset available for download at [1].

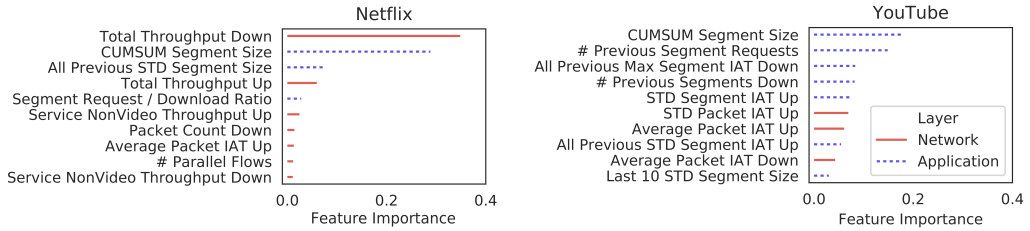
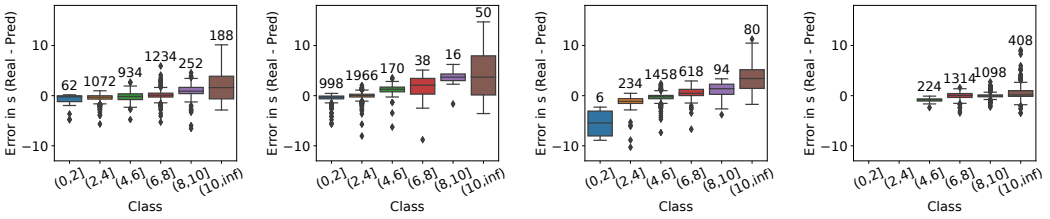



**Fig. 3.** *Startup delay feature importance for Netflix and YouTube Net+App models.*




**(a)** *Netflix.* **(b)** *YouTube.* **(c)** *Amazon.* **(d)** *Twitch.*

**Fig. 4.** *Startup delay inference error across different services for Net+App specific models.*

difference across services, this feature also indicates the number of segments being downloaded. In fact, all of these features align with the expected DASH video streaming behavior: the video startup delay represents the time for filling the client-side buffer to a certain threshold for the playback to begin. Hence, the faster the client reaches this threshold (by downloading segments quickly), the lower the startup delay will be.

Finally, we aim to understand how much the larger errors observed in Figure 2 could negatively impact real world applications of the model for the different services. Figure 4 shows for each service the distribution of the relative errors for inferred startup delays split into two-second bins (using the Net+App feature set). We see that the errors depend heavily on the number of samples in each bin (displayed at the top of each box): the larger the number of samples, the smaller the errors. This result is expected as the model better learns the behavior of video services with more data points, and is also reassuring as the model is more accurate for the cases that one will more likely encounter in practice. The bins with more samples have less than one second error most of the time. For example, the vast majority of startup delays for YouTube occur in the zero to four seconds bins (91% of total sample) and errors are within one second for the majority of these instances. Overall, our models perform particularly well for startup delays of less than ten seconds with errors mostly within one second (with the exception of Amazon, which only has six samples in the (0,2] range of startup delay). Above ten seconds the precision degrades due to the lower number of samples.

**Composite models.** We evaluate a composite model for inferring startup delay across multiple services. Figure 5 reports startup delay inference errors for Netflix for the model trained using solely Netflix sessions (specific model), the one using data from all four services (composite model), and finally the one using only the other three services (excluded model). The composite model is helpful in that we can perform training and parameter tuning once across all the services and deploy a single model online for a small loss in accuracy. We present results for Netflix, but the conclusions were similar for the other three services. The composite model performs almost as well as the specific model obtained from tailored training of a regressor for Netflix; the RMSE of the

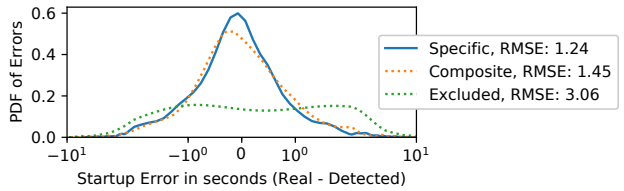
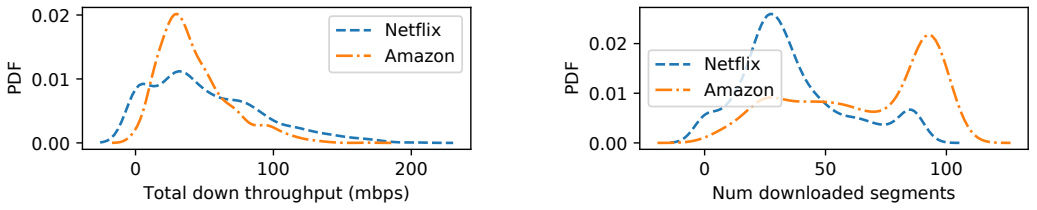



**Fig. 5.** *Relative error in startup delay inference with same or composite Net+App models.*

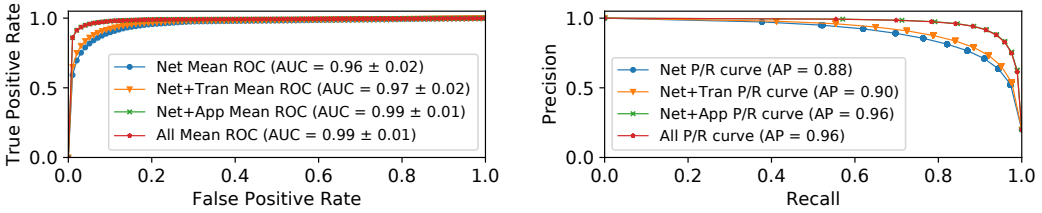



**(a)** *Total down throughput.* **(b)** *Number of downloaded segments.*

**Fig. 6.** *Total down throughput is similar across services; number of downaloded segments is not.*

specific model is 1.24 and that of the composite model is 1.45. Our analysis of the data shows that errors are mostly within one second for startup delays between two and ten seconds, the biggest discrepancies occur for delays below two and above ten seconds due to the low number of samples. In summary, these results suggest it is possible to predict startup delay with a composite model across multiple services.

**Applying a composite model to new services.** Given that all the video streaming services use DASH, our hope was to train the model with data from a few services and later use it for other services not present in the training set. To test the generality of the model, we evaluate a model for inferring startup delay, where we train with Amazon, YouTube, and Twitch data and test with Netflix (excluded model). Unfortunately, the results in Figure 5 show that the excluded model performs very poorly compared to both the specific and composite models more than doubling the RMSE to 2.91. To better understand this result, we analyze the differences and similarities of the values of input features among the four different services. Although Netflix's behavior is often similar to that of Amazon, the similarities are concentrated on only a sub-set of the features in our models. For example, Figure 6a shows the distribution of the total downstream throughput in the first ten seconds and Figure 6b shows the number of downloaded segments—two of the most important features for Netflix's inference. We observe that the distribution of the downstream throughput is fairly similar between Netflix and Amazon. On the other hand, the distributions of the number of downloaded segments present significant differences peaking respectively at 25 and 100 segments. Given these discrepancies, the models have little basis to predict startup delay for Netflix when trained with only the other three services. Whether it is possible to build a general model to infer startup delay across services is an interesting question that deserves further investigation. In the rest of this paper, we rely on the composite models based on the Net+App features to infer startup delay across services.




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

**Fig. 7.** *Resolution inference using different feature sets (For all four video services).*

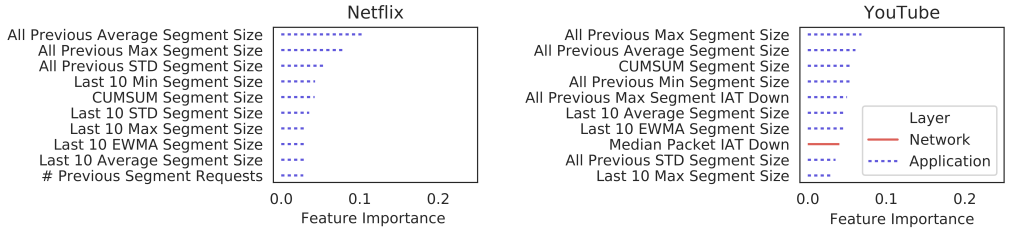


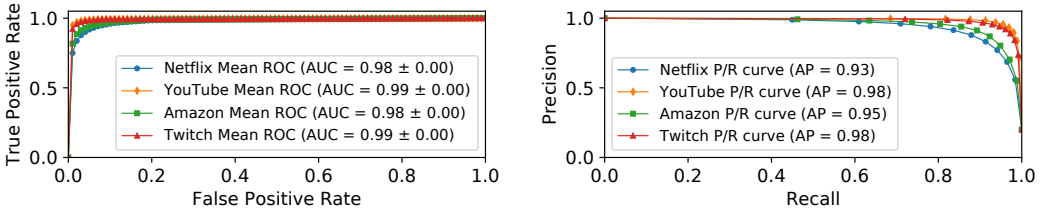

**Fig. 8.** *Resolution inference feature importance for Netflix and YouTube Net+App models.*

## 5.2 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. Similarly to the previous section, we present aggregated results for the different layers across the studied services. We report the receiver operating characteristic (Figure 7a) and precision-recall (Figure 7b) curves for the models weighted average across the different resolution 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 the Net+App model. Figure 8 reports the results for Netflix and YouTube, which show that most features are related to segment size. The same applies to the other services. This result confirms the general intuition: given similar content, higher resolution implies more 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.

Figure 9 presents the accuracy for each video service using the specific model. We see that overall the precision and recall are above 81% and false positive rate below 12% across all services. The accuracy of the resolution inference model is particularly high for YouTube and Twitch with average precision of 0.98 for both services. The accuracy is slightly lower for Amazon and Netflix, because these services change the resolution more frequently to adapt to playing conditions, whereas YouTube and Twitch often stick to a single, often lower, resolution.




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

**Fig. 9.** *Resolution inference across services (with Net+App feature set).*

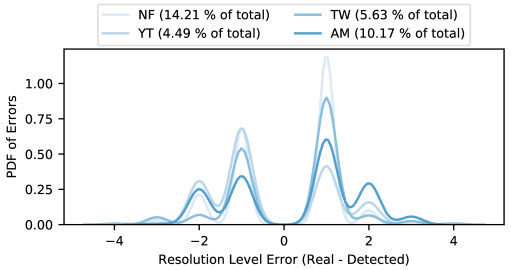
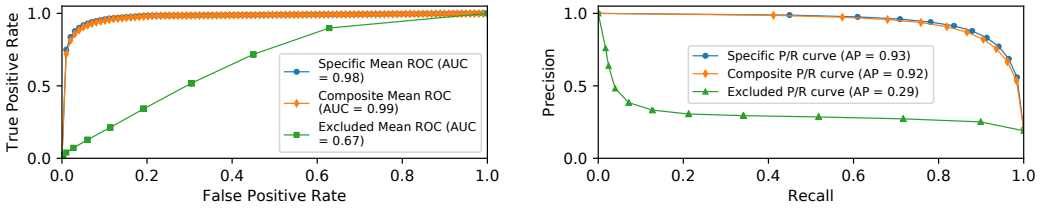



**Fig. 10.** *Kernel Density Estimation function for the relative error in resolution inference for Net+App models (normalized to levels).*

We also quantify the error rate for resolution inference. Figure 10 shows the kernel density estimation obtained plotting relative 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 from the ground truth. For example, Netflix is the service with the higher number of incorrectly 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 (*e.g.*, good vs. bad).

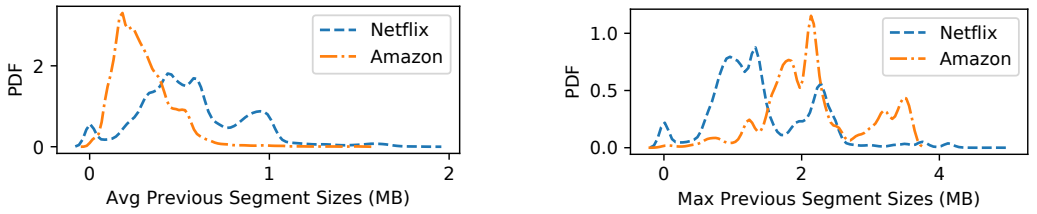
**Composite models.** Figure 11 shows model accuracy when testing with Netflix using the specific model, the composite model, and the excluded model. We illustrate with results for Netflix, but the conclusions are similar when doing this analysis for the other three services. The composite model performs nearly as well as the specific model (with average precision at 92% compared to 93% for the specific model). These results are aligned with the general correlation between segment sizes and resolution that we observed for all services. Indeed, the four most important features for the composite model are all related to segment size: weighted average and standard deviation of the sizes of the last ten downloaded segments, and the maximum and average segment sizes. This observation thus provides a solid basis for a composite model based on features related to segments.

**Applying a composite model to new services.** Figure 11 also presents the accuracy of the excluded model, which is trained with data from Amazon, YouTube, and Twitch and tested with Netflix. As with the startup delay model, we observe a strong degradation of model accuracy in




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

**Fig. 11.** *Composite vs. specific Net+App models performance for resolution inference.*




**(a)** *Average previous segment sizes.* **(b)** *Max size previous segments.*

**Fig. 12.** *Both max previous segment size and average previous segment size differ across services.*

this case. The average precision is only 20%. Our analysis of the distributions of values of the most important input features for the different services reveals even worse trends than discussed for the startup delay. Although Netflix's behavior is most similar to Amazon's, there are sharp differences across all the most important features. For example, Figure 12 shows the distribution for the two most important features for Netflix, the average segment size of previously downloaded segments and their max size. As a result, the accuracy of the excluded model suffers. Our comprehensive analysis of the distributions of feature values across all services indicates that achieving a general model for resolution seems even more challenging than for startup delay as services' behavior vary substantially. As for startup delay, the rest of this paper will rely on the composite model with Net+App features to infer resolution across services.

## 6 Video Quality Inference in Practice

In this section we present the challenges we discovered when applying the methods in practice. Testing our models in a long-running deployment raises a new set of challenges that are not faced by offline models that operate on curated traces in controlled lab settings. Two factors, in particular, affected the accuracy of the models: (1) the challenge of accurately detecting the start and end of a video session in the presence of unrelated cross-traffic and attributing the video session traffic to a particular service; and (2) the granularity of training data versus what is practical to collect in an operational system. We first characterize the deployment and the collected data and then present how we tackle the two challenges.

### 6.1 In-Home Deployment Dataset

**Embedded home network monitoring system.** We have developed a network monitoring system that collects the features shown in Table 1. Our current deployment has been tested extensively on both Raspberry Pi and Odroid platforms connected within volunteers' homes. The

| Speed [mbps] | Homes | Devices | # Video Sessions | | | |
|---|---|---|---|---|---|---|
| | | | Netflix | YouTube | Amazon | Twitch |
| (0,50] | 20 | 329 | 21,774 | 65,448 | 2,612 | 2,584 |
| (50,100] | 23 | 288 | 11,788 | 50,178 | 3,273 | 3,345 |
| (100,500] | 19 | 277 | 13,201 | 38,691 | 2,030 | 197 |
| (500,1000] | 4 | 38 | 523 | 442 | 86 | 1 |
| Total | 66 | 932 | 47,286 | 154,759 | 8,001 | 6,127 |

**Table 3.** *Number of video sessions per speed tier.*

system is flexible and can operate in a variety of configurations: our current setup involves either setting the home gateway into bridge mode and placing the device in between the gateway and the user's access point, or deploying the device as a passive monitoring device on a span port on the user's home network gateway.

The system collects network and application features aggregated across five-second intervals. For each interval, we report average statistics for network features divided per flow, together with the list of downloaded video segments. Every hour, the system uploads the collected statistics to a remote server. To validate the results produced by the model, we instrument the extension presented in Section 4.1 in five homes. Through the extension we collect ground truth data for 2,347 streaming sessions for the four services used to train the models.

**Video sessions from 66 homes.** We analyze data collected between January 23, 2018, and March 12, 2019. We concluded the data collection in May 2019, at which point we had 60 devices in homes in the United States participating in our study, and an additional 6 devices deployed in France. Downstream throughputs from these networks ranged from 18 Mbps to 1 Gbps. During the duration of the deployment, we have recorded a total of 216,173 video sessions from four major video service providers: Netflix, YouTube, Amazon, and Twitch. Table 3 presents an overview of the deployment, including the total number of video sessions collected for each video service provider grouped by the homes' Internet speed (as per the users' contract with their ISP) and the number of unique device MAC addresses seen in the home networks.

Additionally, we periodically (four times per day) record the Internet capacity using active throughput measurements (*i.e.*, speed tests) from the embedded system. We collect this information to understand relationships between access link capacity and video QoE metrics.

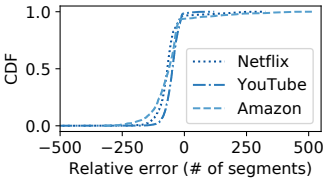
**Ethics considerations.** Our deployment was conducted in collaboration with *The Wall Street Journal*; it went through an extensive evaluation and approval process. The data collection was approved by Princeton University's IRB and *The Wall Street Journal*'s internal ethics and standards board. The users had to personally install the collection devices into their homes with a clear understanding of the data that would be collected and how the data would be used.

### 6.2 Practical Challenges for Robust Models: Session Detection

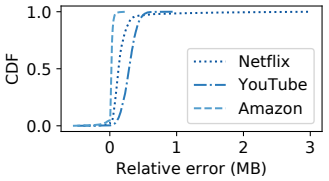
The inference of video quality from within the network without access to the application data requires tailoring three techniques to efficiently extract and process features from network traffic:

(1) Identifying traffic corresponding to video streaming sessions. (Section 6.2.1)
(2) Within this traffic, dividing the traffic into video segments, an application feature that is necessary for the inference models. (Section 6.2.2)
(3) Finally, based on the resulting collected features, infer the start and end of video sessions. (Section 6.2.3)

The rest of this section describes these three steps.




**(a)** *Number of segments.*




**(b)** *Segment sizes.*

**Fig. 13.** *Relative error in segment detection.* [2]

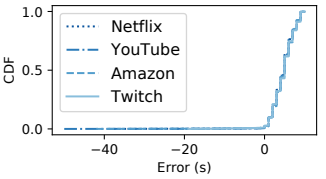



**Fig. 14.** *Absolute error in segment start time estimation.*

*6.2.1 Flow Labeling.* We isolate traffic to and from video services by matching DNS lookups against known signatures for each video service. Because video sessions are often encrypted, we rely on a client's 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. For example, `(.+?\.)?nflxvideo\.net` captures all subdomains corresponding to Netflix's video caches. Some video providers also use external hosting services, so we must also capture traffic to these third-party services. In order to build the signatures dataset, we use our training data of roughly 13,000 sessions, mapping URLs to DNS signatures for the four services. Additionally, as some video players might initiate flows without the initial DNS resolution (*e.g.*, with native implementations of video players or a cached DNS resolution), we assemble a list of known prefixes of video caches and servers we use to identify flows to video services even without the initial DNS resolution.

*6.2.2 Segment Detection.* The models we use operate on the features included in Table 1. The features list includes some application-layer features, such as video segment download inter-arrival times, which have to be inferred from the encrypted traffic. Krishnamoorthi *et al.*. [23] mentioned, without formalizing, that individual video segment information can be extracted by observing patterns in upstream traffic. The key idea behind this method is to use the times of upstream requests to break down the stream of downstream packets into video segments. Although newer versions of HTTP permit pipelining of multiple requests over a single connection, adaptive video plays one video segment after the other and hence sequential request-response is natural. The key challenge is how to identify the times of requests for video segments within the stream of packets from the client to the server without breaking encryption. Fortunately, upstream packets are either requests for video segments or acknowledgments, so we can identify requests based on packet size.

We implement this segment identification mechanism in the deployed system. Our implementation tracks the last seen upstream packet with a non-zero payload and counts the subsequent downstream traffic to determine segment sizes. The method is similar for QUIC, except that QUIC signaling packets have non-zero payload size. We determine that a QUIC packet contains a request if the UDP payload size is greater than a threshold. We set this threshold to 150 bytes based on the current QUIC specification [31] and the empirical distribution of QUIC upstream packet sizes in the dataset of QUIC YouTube sessions we collect.

**Validation.** We validate the technique used to detect segment boundaries and estimate their size. To do so, we use the collected ground truth dataset containing time series history for all segment downloads over recorded video sessions and compare it with the system output over the same sessions.

[2]We were unable to obtain ground truth for segment sizes for Twitch because the application uses chunked transfer encoding, where the size of the segment is inline with the HTTP payload, as opposed to in a separate header.

We characterize video segments in terms of both the number of segments and segment sizes in a given video session. Figures 13a and 13b show the accuracy of this approach. The method works similarly across the services we tested; it slightly underestimates the number of segments in a video session. We suspect two possible reasons for this underestimation: (1) in order to determine whether a video session is ongoing, we infer such events in real-time. This process is described in detail in the next section. As the session detection may be subject to errors due to lack of visibility in the encrypted traffic, *i.e.* indicate that a session start time is slightly later than the actual start time, this could cause the segment detection to miss a few segments at the beginning of a session; and (2) some services generate a number of empty requests at the beginning of a session, possibly to test the path to their video caches, which may cause the small error shown in Figure 13a. Evaluation of segment size estimation (Figure 13b) shows that we accurately infer segment size for all three services in the general case, with an error within 250KB of the actual segment size in the majority of cases. Overestimates may be caused by double-counting retransmissions and the fact that some services periodically issue empty requests.

*6.2.3 Identifying Video Sessions.* Identifying a video session from encrypted network traffic is a challenge as network traffic is noisy. To detect session start and end times, we extend the method from Dimopoulos *et al.* [15], which identifies a spike in traffic to specific YouTube domains to determine the start of a video session and a silent period to indicate the end of a video session. We build on this approach to design a session identification method that generalizes this intuition across video services.

Our analysis of video sessions across different services confirms that at the beginning of each video session, there is a spike in the volume of traffic that comes from servers that are associated with the video services but are distinct from the servers that deliver video traffic. This activity can correspond to, for example, the download of the player code or thumbnail image downloads as users browse the catalog.

Two other features are useful for identifying session boundaries. First, in addition to generating more traffic to other servers, video players also generate new network flows to download audio and video content when moving from one video to another. Second, considering the buffer-based approach of video services, most sessions have no video traffic at the end, when the player is exclusively rendering the buffered content without having to further retrieve new segments.

Hence, we idenfity a new session under the following conditions:

(1) The amount of non-video service traffic exceeds a threshold $\alpha$ or its rate over the entirety of the service traffic exceeds a threshold $\beta$.
(2) The time elapsed since the last video traffic activity exceeds a threshold $\gamma$ or the rate of new flows initiated exceeds a threshold $\delta$.

We set the thresholds based on the general behavior of the studied protocols. In particular, we set $\alpha$ to 50 kbps and $\beta$ to 5% following the distribution of sessions collected in our dataset; both are relatively conservative, but we rely on the second condition of our detection method to further increase the accuracy. $\gamma$ is set to 0.5; this value offers a good trade-off between YouTube, that for most sessions uses only a single flow dedicated to video retrieval, and Netflix, which, in contrast, uses a large number of parallel flows. Finally, $\delta$ is set to 30 seconds which corresponds to a small playback buffer with respect to the max buffer size of these services (for example, Netflix uses up to 220 seconds worth of buffered video). We anticipate that in cases where other video services have drastically different traffic profiles, per-service tuning of the thresholds may be necessary.

**Validation.** Figure 14 shows the obtained cumulative distribution function of the error in video session start time; specifically, the figure shows the time between the detected session start minus the ground truth session start as seen from the web history (taking the timestamp of the first video

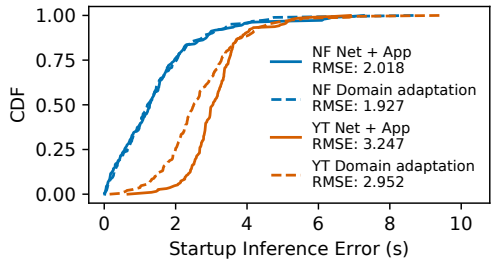



**Fig. 15.** *Effect of domain adaptation on startup delay inference.*

| | Net + App | | | Domain adaptation | | |
|---|---|---|---|---|---|---|
| | Precision | Recall | FPR | Precision | Recall | FPR |
| Netflix | 86% | 86% | 3.4% | 90% | 90% | 2.5% |
| YouTube | 82% | 82% | 4.4% | 78% | 78% | 5.5% |
| Twitch | 82% | 82% | 4.6% | 91% | 91% | 2.2% |
| Amazon | 76% | 76% | 6% | 78% | 78% | 5.6% |

**Table 4.** *Effect of domain adaptation on resolution inference.*

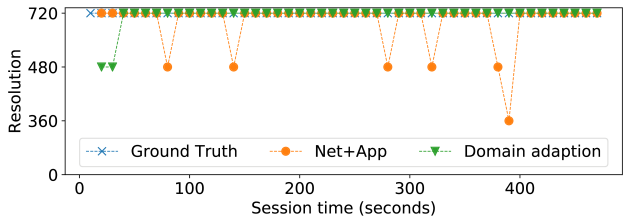



**Fig. 16.** *Sample session for errors with/without domain adaptation.*

or audio segment request) for all video sessions. The results show that the estimate for session start time is within a few seconds of the actual time for most sessions; in some cases, the technique infers an early start time, perhaps as a result of non-video traffic mistakenly attributed to video traffic. Additionally, our system extracts and reports features at a fixed period (currently every five seconds), which makes it impractical to identify an exact start time.

### 6.3 Practical Challenges for Robust Models: Data Granularity

An operational monitoring system cannot export information about each individual packet. It is hence common to report traffic statistics in fixed time intervals or time bins (*e.g.*, SNMP or Netflow polling intervals). Our system follows this behavior reporting statistics every five seconds. The training data that we and prior work collect has a precise session start time, whereas the data collected from a deployed system will only have data collected in time intervals, where the session start time might be *anywhere* in that interval, as demonstrated in the previous section. This corresponding mismatch in granularity creates a challenge for the inference models. Furthermore, any error in estimating the start time propagates to the time bins used for inference across the

entire session, resulting in a situation where the time bins in the training and deployment data sets do not correspond to one another at all.

**Domain adaptation.** Intuitively, our approach to address these challenges involves accounting for additional noise that the practical monitoring introduces that is not present in a lab setting or in the training data. To do so, we introduce noise into the training data so that it more closely resembles the data collected from the deployment. The techniques that we apply are grounded in the general theory of domain adaptation [36]; they work as follows: Because the actual start time can fall anywhere within the five-second interval, we pre-process our training data and artificially adjust each session start time over a window of -5 to +5 seconds from the actual start value in increments of 0.5 seconds. For each new artificial start time, we recalculate all metrics based on this value for the entire session. This technique has two benefits: it makes the model more robust to noise, and it increases the volume of training data.

**Validation.** We validated the domain-adapted models against 2,347 sessions collected across five homes of the deployment. We compared each session collected from the extension with the closest session detected in the deployment data. We present results for both startup delay and resolution using the Net+App unified model (as presented in Section 4) and the model after domain adaptation.

Figure 15 shows the resulting improvement in startup inference when we apply domain adaption based models to Netflix and YouTube: In both cases, the root mean square error improves—quite significantly for YouTube. Table 4 shows that we obtain similar improvements for resolution inference, except for YouTube. We posit that this result is attributable to the ground truth dataset collected; the YouTube data is heavily biased towards 360p resolutions (90+%), whereas all other services operated at higher (and more diverse) resolutions. While domain adaptation increases balance across classes, it may slightly impact classes with more prevalence in the training dataset. We leave investigation into mitigating such issues for future exploration. Finally, Figure 16 illustrates how domain adaption helps correct errors on resolution inference through an example session. These results suggest that domain adaptation is a promising approach for bridging the gap between lab-trained models and real network deployments. We expect that results could be further improved by applying domain adaptation with smaller time intervals.

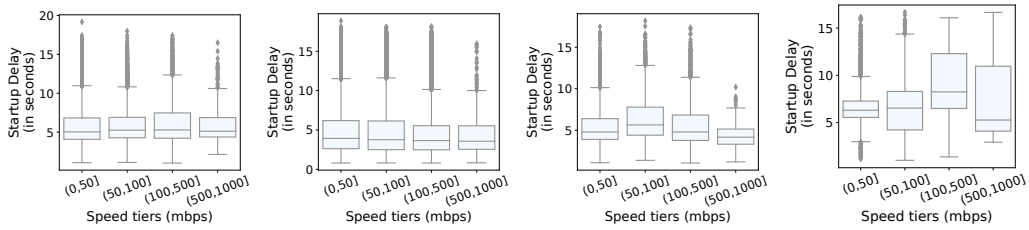
## 7 Inference Results

In this section we present our findings on the inference results gathered from the collected deployment data. We use the Net+App unified models from Section 4 to infer startup delay and resolution on all video sessions collected throughout the lifetime of the deployment (Table 3).

### 7.1 Startup Delay

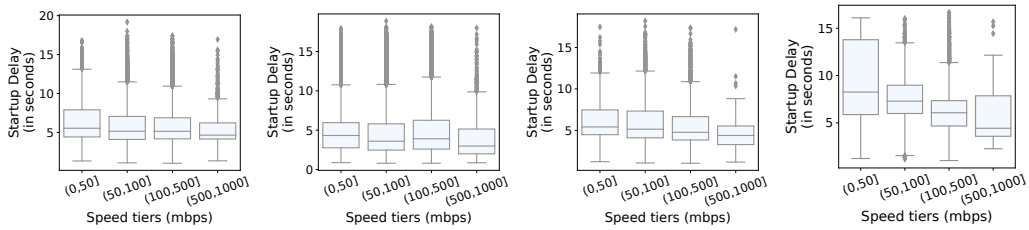
We infer the startup delay for each video session in the deployment dataset in order to pose a question that both ISPs and customers may ask: how does access link capacity relate to startup delay? Answering this question allows us to understand the tangible benefits of paying for higher access capacity with regard to video streaming. We employ two metrics of "capacity": the nominal download subscription speed that users reported for each home, and the 95th percentile of active throughput measurements that we collected from the embedded in-home devices. Figure 17 presents box plots of startup delay versus nominal speeds, whereas Figure 18 presents startup delay versus the 95th percentile of throughput measurements. Note that the number of samples for the highest tier for both Amazon and Twitch is too small to draw conclusions (see Table 3), so we focus on results in the first three tiers for these services.

Figure 17 shows our unexpected result. We see that median startup delays for each service tend to be similar across the subscription tiers. For example, YouTube consistently achieves a median of <5 second startup delay across all tiers and Amazon a constant median startup delay of <6 seconds.




**(a)** *Netflix.* **(b)** *YouTube.* **(c)** *Amazon.* **(d)** *Twitch.*

**Fig. 17.** *Startup Delay Inference vs. Nominal Speed Tier.*



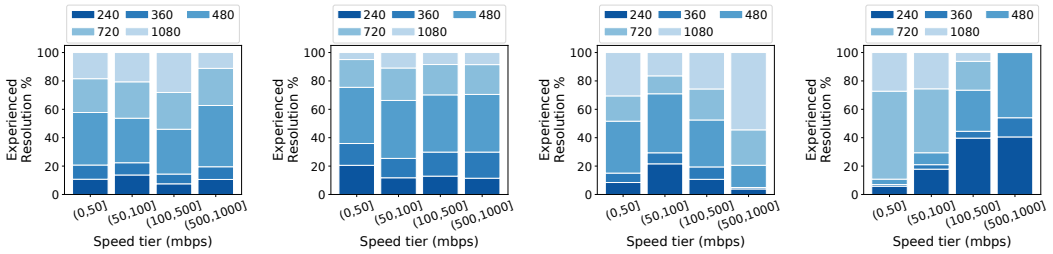

**(a)** *Netflix.* **(b)** *YouTube.* **(c)** *Amazon.* **(d)** *Twitch.*

**Fig. 18.** *Startup Delay Inference vs. Active Throughput Measurements (95th Percentile).*

Netflix and Twitch's startup delay differs by at most ± 2 seconds across tiers, but these plots exhibit no trend of decreasing startup delay as nominal speeds increase as one would expect. There are several possible explanations for this result. One possibility is that actual speeds vary considerably over time due to variations in available capacity along end-to-end paths due (for example) to diurnal traffic demands.

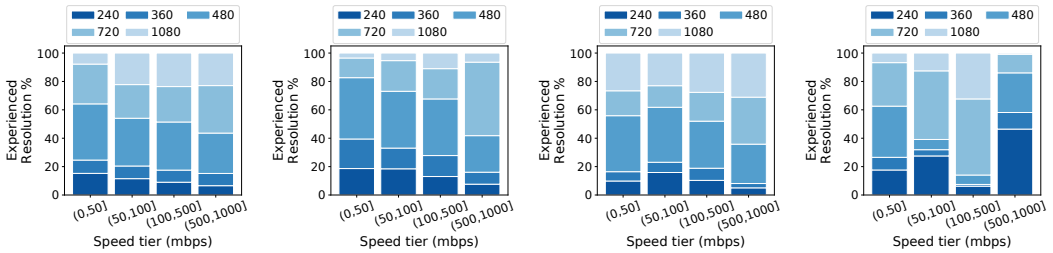
Startup delays follow a more expected pattern when we consider capacity as defined by active throughput measurements. Figure 18 shows that startup delay decreases as measured throughput increases for Netflix, YouTube, and Twitch. For Netflix, the difference in startup delay between the highest and lowest tier is four seconds. This difference is three seconds for YouTube. This figure also highlights the difference in startup delay across services, given similar network conditions. These differences reflect the fact that different services select different design tradeoffs that best fit their business model and content. For example, Netflix has the highest startup delay of all the services. Because Netflix content is mostly movies and long-form television shows—which are relatively long—having higher startup delays may be more acceptable. As we see in the next section, our inference reflects the expected trends that Netflix trades off higher startup delay to achieve higher resolution video streams.

To further clarify the observed trends, we analyze whether operators delivered the expected speeds to their customers, offering an explanation to the results just presented. From our analysis, we observed that, in the majority of cases, operators actually over provision their networks. This was reflected in our active measurements whereas results exceeded the expected speeds. This result confirms our intuition that video quality is less impacted by the access capacity offered by the operators rather than by other factors such as variations in available capacity along end-to-end path.




**(a)** *Netflix.* **(b)** *YouTube.* **(c)** *Amazon.* **(d)** *Twitch.*

**Fig. 19.** *Resolution vs Nominal Speed Tier*




**(a)** *Netflix.* **(b)** *YouTube.* **(c)** *Amazon.* **(d)** *Twitch.*

**Fig. 20.** *Resolution vs. Active Throughput Measurements (95th Percentile).*

## 7.2 Resolution

Next, we infer the resolution for each ten-second bin of a video session. As with our startup delay analysis, we study the correlation between resolution and access capacity. For this analysis, we omit the first 60 seconds of video sessions, since many ABR algorithms begin with a low resolution to ramp up as they obtain a better estimate of the end-to-end throughput. Recall that our video quality inference model outputs one of five resolution classes: 240, 360, 480, 720, or 1080p.

Figure 19 presents resolution versus the nominal speed tier, whereas Figure 20 presents resolution versus the 95th percentile of active throughput measurements. As with startup delay, we observe expected results (*i.e.*, higher resolutions with higher capacities) with measured throughput (Figure 20); the trend is less clear for nominal speed tiers (Figure 19). Recall that the number of samples in the highest speed bar for Amazon and Twitch is small; focusing on the other three tiers shows a clear trend of increased percentage of bins with higher resolutions as capacity increases. For example, Netflix and YouTube in the highest speed tier achieve about 40% more 1080p than in the lowest speed tier. We also observe that, in general, YouTube streams at lower resolution for the same network conditions than other services.

Video resolution is dependent on more factors than simply the network conditions. First, some videos are only available in SD, and thus, will stream at 480p regardless of network conditions. Second, services tailor to the device type as a higher resolution may not be playable on all devices. We can identify the device type for some of the devices in the deployment based on the MAC address. Of 1,290,130 ten-second bins of YouTube data for which we have device types, 616,042 are associated with smartphones, indicating that device type may be a confounder.

## 8 Conclusion

Internet service providers increasingly need ways to infer the quality of video streams from encrypted traffic, a process that involves both identifying the video sessions and segments and processing the resulting traffic to infer quality across a range of services. We build on previous work that infers video quality for specific services or in controlled settings, extending the state of the art in several ways. First, we infer startup delay and resolution delay more precisely, attempting to infer the specific values of these metrics instead of coarse-grained indicators. Second, we design a model that is robust in a deployment setting, applying techniques such as domain adaptation to make the trained models more robust to the noise that appears in deployment, and developing new techniques to identify video sessions and segments among a mix of traffic that occurs in deployment. Third, we develop a composite model that can predict quality for a range of services: YouTube, Netflix, Amazon, and Twitch. Our results show that prediction models that use a combination of network- and application-layer features outperform models that rely on network- and transport-layer features, for both startup delay and resolution. Models of startup delay achieve less than one second error for most video sessions; the average precision of resolution models is above 0.93. As part of our work, we generated a comprehensive labeled set with more than 13,000 video sessions for the four video services. Finally, we applied these models to 16 months of traffic from 66 homes to demonstrate the applicability of our models in practice, and to study the relationship between access link capacity and video quality. We found, surprisingly, that higher access speeds provide only marginal improvements to video quality, especially at higher speeds.

Our work points to several avenues for future work. First, our composite models perform poorly for services that are not in the training set; a truly general model that can predict video quality for arbitrary services remains an open problem. Second, more work remains to operationalize video quality prediction models to operate in real time.

## Acknowledgments

We thank our shepherd Zubair Shafiq and the anonymous reviewers for their helpful comments. We also thank Israel Marquez Salinas, Srikanth Sundaresan, and Sarah Wassermann for their help with early stages of this project and Mark Crovella for the valuable discussions on domain adaptation. This work was supported by the ANR Project N$^o$ ANR-15-CE25-0013-01 (BottleNet), NSF Award CNS-1704077, the Google Faculty Research Award program, the Comcast Tech Research Grant program, and Inria through the IPL BetterNet and the associate team HOMENET. Special thanks to our collaborators at *The Wall Street Journal*, especially Shalini Ramachandran, Tom Gryta, Kara Dapena, and Patrick Thomas, for their efforts in managing the deployment and recruiting participants.

## References

[1] 2019. Labeled video sessions dataset. https://nm-public-data.s3.us-east-2.amazonaws.com/dataset/all_traffic_time_10.pkl.

[2] 2019. Video Collection Tools. https://github.com/inria-muse/video_collection.

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

[4] 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.

[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] Francesco Bronzino, Paul Schmitt, Sara Ayoubi, Nick Feamster, Renata Teixeira, Sarah Wasserman, and Srikanth Sundaresan. 2019. Lightweight, General Inference of Streaming Video Quality from Encrypted Traffic. *arXiv preprint arXiv:1901.05800v1* (2019).
[10] Martin Brown. 2006. Traffic Control HOWTO. http://tldp.org/HOWTO/Traffic-Control-HOWTO/index.html.
[11] Chrome web 2018. Chrome webRequest API. https://developer.chrome.com/extensions/webRequest.
[12] Chrome1080 2018. netflix-1080p - Chrome extension to play Netflix in 1080p and 5.1. https://github.com/truedread/netflix-1080p.
[13] Chromedriver 2018. ChromeDriver - WebDriver for Chrome. https://sites.google.com/a/chromium.org/chromedriver/.
[14] 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.
[15] 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.
[16] 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.
[17] Driving Engagement 2017. Driving Engagement for Online Video. http://events.digitallyspeaking.com/akamai/mddec10/post.html?hash=ZDlBSGhsMXBidnJ3RXNWSW5mSE1HZz09.
[18] 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/.
[19] Craig Gutterman, Katherine Guo, Sarthak Arora, Xiaoyang Wang, Les Wu, Ethan Katz-Bassett, and Gil Zussman. 2019. Requet: Real-Time QoE Detection for Encrypted YouTube Traffic. In *ACM Conference on Multimedia Systems (MMSys '19)*. Amherst, MA, USA.
[20] 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.
[21] 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.
[22] ITU 2012. ITU Methodology for the subjective assessment of the quality of television pictures. Recommendation ITU-R BT.500-13.
[23] 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.
[24] 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.
[25] 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.
[26] 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.
[27] M. Hammad Mazhar and Zubair Shafiq. 2018. Real-time Video Quality of Experience Monitoring for HTTPS and QUIC. In *INFOCOM*. Honolulu, HI.
[28] 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.
[29] Openwave Mobility. 2018. Mobile Video Index. https://landing.owmobility.com/mobile-video-index/.
[30] 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.
[31] QUIC Draft 2018. QUIC: A UDP-Based Multiplexed and Secure Transport. https://www.ietf.org/id/draft-ietf-quic-transport-09.txt.
[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] Iraj Sodagar. 2011. The MPEG-DASH standard for multimedia streaming over the internet. *IEEE MultiMedia* 18, 4 (2011), 62–67.

[35] T. Stockhammer. 2011. Dynamic adaptive streaming over HTTP: standards and design principles. In *ACM Conference on Multimedia Systems (MMSys '11)*. San Jose, CA.
[36] Baochen Sun, Jiashi Feng, and Kate Saenko. 2016. Return of frustratingly easy domain adaptation. In *Thirtieth AAAI Conference on Artificial Intelligence*.
[37] Tcpdump 2017. tcpdump - dump traffic on a network. https://www.tcpdump.org/manpages/tcpdump.1.html.
[38] The Wall Street Journal 2019. The Truth About Faster Internet: It's Not Worth It. https://www.wsj.com/graphics/faster-internet-not-worth-it/.
[39] 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.
[40] 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.
[41] Youtube API 2018. YouTube Player API Reference for iframe Embeds. https://developers.google.com/youtube/iframe_api_reference.

Received August 2019; revised September 2019; accepted October 2019