Empowering Video Players in Cellular: Throughput Prediction from Radio Network Measurements

Darijo Raca, Ahmed H. Zahran, Cormac J. Sreenan
University College Cork
{d.raca,a.zahran,cjs}@cs.ucc.ie

Rakesh K. Sinha, Emir Halepovic, Rittwik Jana, Vijay Gopalakrishnan, Balagangadhar Bathula
AT&T Labs – Research
{sinha,emir,rjana,gvijay,balab}@research.att.com

Matteo Varvello*
Brave Software
varvello@brave.com

ABSTRACT

Today's HTTP adaptive streaming applications are designed to provide high levels of Quality of Experience (QoE) across a wide range of network conditions. The adaptation logic in these applications typically needs an estimate of the future network bandwidth for quality decisions. This estimation, however, is challenging in cellular networks because of the inherent variability of bandwidth and latency due to factors like signal fading, variable load, and user mobility. In this paper, we exploit machine learning (ML) techniques on a range of radio channel metrics and throughput measurements from a commercial cellular network to improve the estimation accuracy and hence, streaming quality. We propose a novel summarization approach for input raw data samples. This approach reduces the 90th percentile of absolute prediction error from 54% to 13%. We evaluate our prediction engine in a trace-driven controlled lab environment using a popular Android video player (ExoPlayer) running on a stock mobile device and also validate it in the commercial cellular network. Our results show that the three tested adaptation algorithms register improvement across all QoE metrics when using prediction, with stall reduction up to 85% and bitrate switching reduction up to 40%, while maintaining or improving video quality. Finally, prediction improves the video QoE score by up to 33%.

CCS CONCEPTS

• **Information systems** → **Multimedia streaming**; • **Networks** → Public Internet; **Wireless access networks**; **Network measurement**;

KEYWORDS

HAS, 4G, LTE, Mobility, throughput prediction, DASH, adaptive video streaming

*Work done while with AT&T Labs – Research.

Conference'17, July 2017, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

ACM Reference format:
Darijo Raca, Ahmed H. Zahran, Cormac J. Sreenan, Rakesh K. Sinha, Emir Halepovic, Rittwik Jana, Vijay Gopalakrishnan, Balagangadhar Bathula, and Matteo Varvello. 2019. Empowering Video Players in Cellular: Throughput Prediction from Radio Network Measurements. In *Proceedings of ACM Conference, Washington, DC, USA, July 2017 (Conference'17),* 12 pages.
https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Recent reports show that re-buffering ranks ahead of playback failures, slow startup, and poor quality on the list of user complaints about video streaming.[1] Cellular networks are a challenging environment for Adaptive BitRate (ABR) video streaming due to various reasons. Radio channel conditions and cell load are continuously changing. Data transmission to a mobile device is coordinated by protocols that operate at diverse time scales, e.g., radio channel scheduling at millisecond level vs. congestion control at hundreds of milliseconds to seconds level. Furthermore, the base station scheduler allocates the wireless resources based on the bandwidth demand of each device and their channel conditions; this can cause burstiness in the cellular data traffic. State-of-the-art video clients employ adaptation algorithms that use network and application state to determine the quality of the next video chunk to download. There are several recent algorithms based on approaches such as optimization [35, 37], control theory [3], game theory [1], machine learning [15], and other heuristics integrating well-known averaging techniques. While these algorithms differ in the specifics of their decision making, most of them need to know the available network bandwidth to determine the quality of the video to download. Since this information is not readily available to them, clients typically use recent throughput measurements to *estimate* the likely network conditions. This is combined with application state in terms of buffer occupancy and chunk qualities to yield decisions on the quality of future chunks to be selected. The high variability in cellular networks, however, leads to significant throughput estimation errors and in turn results in sub-optimal decisions [21]. While there have been proposals that attempt to use cellular-specific information to aid estimation [5, 33], they are in the minority.

Since statistical estimation is affected by the variability in cellular networks, there has been significant interest in exploring the possibility of accurately predicting throughput instead. Indeed, several

[1] http://connect.mux.com/2017-streaming-perceptions-report

studies based on trace-driven controlled experiments demonstrate that video streaming quality can be improved in cellular networks if throughput can be predicted accurately [13, 21, 40]. The main reason is that stalls are avoided because the player would not overcommit to large chunks of high bitrate when throughput drop is predicted [14]. Additionally, players can avoid streaming the lowest quality after a stall or at startup when high throughput is predicted. Finally, accurate prediction can help reduce the number of quality changes. However, accurate throughput prediction in cellular networks is challenging due to the complex changes and interactions in the network state [39]. It is also unclear whether a typical mobile device can extract appropriate network-level information and leverage it to make accurate throughput predictions.

In this paper, we present a novel ML-based throughput prediction engine for use by mobile devices in cellular networks. This engine predicts average network throughput for a future time window (horizon) based on radio-specific metrics and network throughput observed over a historical window. Our engine uses a novel statistical data summarization method that significantly improves the throughput estimation accuracy in comparison to both ML throughput estimation based on raw data samples and application-level bandwidth estimation commonly used in ABR clients. This result illustrates not only the predictive power of radio metrics but also the effectiveness of our summarization technique. We also show, using an implementation of our engine on Android phones, that state-of-the-art video streaming algorithms benefit from the improved throughput estimation using our prediction engine [25, 38]. All evaluated adaptation algorithms improve their stall (up to 85%) and switching (up to 40%) performance in all tested scenarios. The average quality also improves in most scenarios. Overall, the tested algorithms show improvements of 6%-33% in QoE score.

The novelty of our work derives from two key elements: accurate prediction of *future* throughput using radio metrics on mobile devices and a realistic quantification of the effect of prediction on adaptive video streaming performance. Our contributions are summarized as follows:

- In Section 3 we propose a novel ML-based throughput prediction technique that leverages radio metrics to improve the throughput prediction accuracy in mobile networks significantly. We thoroughly explore the design space (prediction horizon, measurement history, mobility pattern, etc.) to illustrate the impact of different design parameters on the prediction accuracy.
- In Section 4, we build a testbed and evaluate three ABR algorithms with and without our prediction engine. We use publicly available 4G/LTE cellular traces, real video content and an Android client of a broadly-adopted video player (ExoPlayer [2]). Our evaluation considers the impact of important parameters, such as chunk duration and prediction horizon. In all tested scenarios, the performance metrics show a clear improvement, leading to better user QoE.
- In Section 5 we perform a field evaluation in an operational cellular network. This exercise enables us to identify challenges that merit further research and will be of interest to practitioners seeking to implement ML-based prediction in real systems (Section 6).

[2]http://google.github.io/ExoPlayer/

2 MOTIVATION

ABR algorithms commonly consider combinations of network and/or application states for quality selection decision, as illustrated in Figure 1. The *estimator* module captures the network state, while the *application monitoring* module captures the video player state by monitoring playback buffer and streamed video quality. *Estimator* consists of sample processor sub-module (not shown in the figure) responsible for collecting significant statistics from *Chunk loader* module in order to estimate available bandwidth for chunk. For example, depending on algorithm design, it can track the rate sample for each chunk or multiple rate samples per chunk (in the case when a time-based measurement is used). Sample processor feeds sample array to one of the smoothing functions, from which final throughput estimate for next chunk is passed to *adaptation logic* module. Finally, the *adaptation logic* module combines information from *estimator* and *application monitoring* modules to decide on the quality of the chunks to be requested. *Chunk loader* module carries out decision from the *adaptation logic* module and requests next chunk. While most of the efforts focus on enhancing *adaptation logic*, little or no work has been done in improving bandwidth estimation. Most of the algorithms employ one of the standard smoothing techniques (e.g., arithmetic, harmonic or exponential moving average to name a few) applied over n last measured rate samples, where rate samples can be a chunk, time or size based. For example, ARBITER+ [38] combines chunk and time-based sampling with an exponential moving average for bandwidth estimation. The approach we take in this paper is to leverage Android OS, which provides APIs for capturing measurements of radio channel metrics, velocity and throughput samples. We use this capability to build *prediction engine* that captures those metrics at one-second granularity to provide more accurate throughput prediction (see Figure 1, dotted boxes). As a result, this module feeds predicted value to *estimator* module. *Estimator* module may process this value through standard smoothing techniques or feed directly to *adaptation logic* to improve decision-making process for the next chunk.

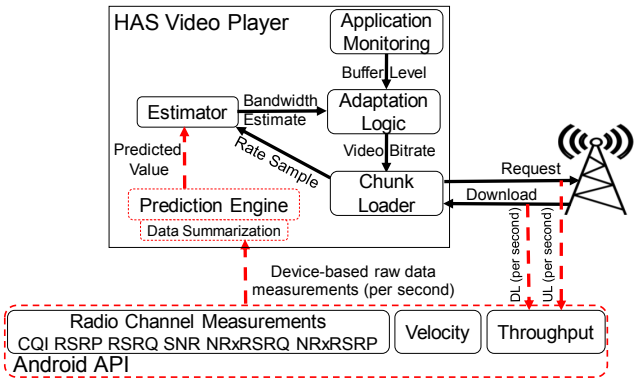


Figure 1: Simplified HAS player architecture and addition of measurement and prediction module

The need for better throughput estimates To further quantify the needs for better throughput prediction we analyze some of the most commonly used bandwidth estimation techniques. We use results from Section 4 for cases with no throughput guidance. Every HAS algorithm in this study takes an estimate from its *estimator* module and uses it to decide bitrate for next chunk. We take that estimate and compare it against a delivery rate of a requested chunk (we use information from logs which allow us to have delivery rates of "future" requested chunks). We label this case as "1-chunk". Furthermore, we extend analysis by looking beyond one chunk. For "2-chunk" and "5-chunk" scenarios, we calculate average value of delivery rates for next two/five chunks requested. Figure 2 depicts accuracy of three classical bandwidth estimation techniques, EWMA, average and median. For accuracy, we analyze the absolute value of the residual error between throughput estimates and the actual average download rate of chunks.

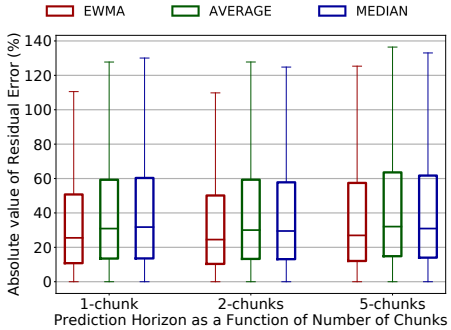


Figure 2: Accuracy of different throughput estimation techniques for different prediction horizon values

The main takeaway from Figure 2 is that none of these techniques exhibit high estimation (not prediction) accuracy regardless of how far into the future we look.

To further illustrate potential gain in improving *estimator* module, we show in Figure 3 one randomly selected video session played using Exoplayer and its default ABR algorithm, with and without accurate throughput guidance. With accurate prediction, the player starts at a higher quality and settles to the optimal rate earlier (i.e., highest) than when relying on classical bandwidth estimation (e.g., median). Furthermore, prediction helps to recognize the level of drop in bandwidth more accurately, forcing the client to switch to the lowest quality quickly. As a result, buffer underrun is cut by 90%, improving the overall user experience.

3 THROUGHPUT PREDICTION VIA ML

In this section, we present our proposed throughput prediction approach and extensively evaluate its performance under different scenarios and configurations.

3.1 Proposed Prediction Technique

In our design, we focus on the prediction of *average* throughput rather than *instantaneous* throughput. The needs of ABR applications motivate this decision. To illustrate, when a streaming client downloads video chunks, throughput fluctuations over the future x seconds is of little concern for the player. What matters is the

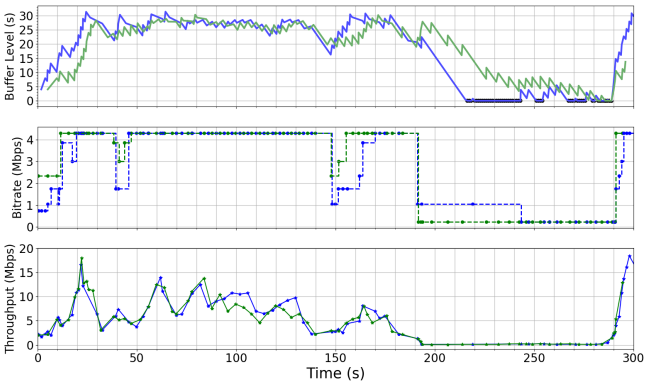


Figure 3: Comparing ExoPlayer sessions with throughput prediction (green, 12 sec. horizon) and without (blue)

average throughput that the video player will observe in the next x seconds, as this will drive the behavior of the video adaptation algorithm. We call x the *prediction horizon* and make it a parameter of the ML algorithm. The data driving this prediction includes both radio metrics and throughput samples collected at the device at arbitrary granularity. We use a granularity of 1 second; this is driven by the sampling period used in capturing radio data that we use in this paper [19]. However, the proposed prediction technique can also be used in conjunction with other sampling time scales. A typical prediction engine input would consist of one or more historic radio metrics and throughput samples. In this study, we investigate several *scenarios* for the combination of prediction horizon and history length. We introduce the notation *PyFx*, where Py denotes past y seconds of historical data, and Fx denotes a prediction horizon of future x seconds. In Section 3.4, we show the impact of varying Py and Fx on the prediction accuracy.

The combination of channel conditions and the current state of a cell largely determines the number of allocated radio resources blocks; this translates to achievable throughput at the device. To capture these dimensions, we use the following radio channel metrics in conjunction with physical mobility speed (in kmh) and historical application throughput:

- *RSRP* - Average power over cell-specific reference symbols carried inside distinct resource element (RE). RSRP is used for measuring cell signal strength/coverage and therefore cell selection (dBm)
- *RSRQ* - defined as the ratio N×RSRP/(RSSI), where N is the number of physical resource blocks (PRB). RSSI is pure wideband power measurement, including intracell power, interference, and noise. RSRQ is measured in dB.
- *SNR* - signal-to-noise ratio (dB)
- *CQI* - CQI is feedback provided by UE to eNodeB. It indicates data rate that could be transmitted over a channel (highest MCS with a BLER probability less than 10%), as the function of SINR and UE's receiver characteristics. Based on UE's prediction of the channel, eNodeB selects an appropriate modulation scheme and coding rate
- *NRxRSRQ, NRxRSRP* - RSRQ and RSRP values for the neighboring cell
- *Downlink Throughput/Uplink Throughput* - download/upload rate measured at the device (kbit/s)

We select the random forest (RF) as our ML algorithm. RF [2] represents an ensemble/boosting learning method for regression and classification tasks. The main reasons for selecting RF are following. First, RF works by growing a collection of decision trees (weak learners) and then making a prediction by taking the mean of individual trees. This approach reduces overfitting because each tree is constructed on a randomly selected subset of features. They are further de-correlated (which minimizes the overfitting) by considering a random subset of features for each split of the decision tree. Second, it can be used for feature ranking, enabling analysis of the importance of each metrics used for training. Finally, it requires minimal tuning (number of trees are the most critical hyperparameter). Also previous studies show that RF outperforms other ML techniques for this problem space [20, 28].

Summarization Techniques: When applying machine learning techniques, the first step includes feeding data without any processing (except normalization methods). For throughput prediction, we use the full *history* of each metric (*raw* in the list below). However, classical ML algorithms usually require high-level features extracted from raw data (i.e., feature engineering) to achieve high accuracy. For example, history may be summarized, e.g., by the average value. However, the average and entire history can be affected by outliers and can react slowly to changes if maintaining a long history. Having outliers in real data is an unavoidable hurdle. For throughput prediction, we are interested in capturing patterns based on historical data. Instead of feeding every sample collected at the arbitrary time interval to the ML algorithm, we only feed key values that best summarize the data. Collected historical data represents a distribution for each metric. To infer unknown distribution from empirical data, we use percentiles (if historical data follows a normal distribution, using mean and standard deviation is enough to explain the whole distribution). We empirically test different combinations of percentiles (including the mean). Finally, we find that the inter-quartile range, mean and 90^{th} percentile give the highest prediction accuracy. Let m_i^n represent n_{th} metric at time i. We consider the following summarization technique: *quantile:* for a $(m_{i-1}^n, m_{i-2}^n, m_{i-3}^n, ..)$ array of values we calculate the following metrics: $25^{th}, 50^{th}, 75^{th}, 90^{th}$ and *mean* of the input array. We compare this summarization technique to the strawman prediction based on *raw* data; i.e., using $(m_{i-1}^n, m_{i-2}^n, m_{i-3}^n, ..)$ for every n as input to the prediction model.

3.2 Evaluation of Prediction Techniques

3.2.1 4G Dataset. Our evaluation is based on publicly available 4G dataset with RAN metrics [19]. The dataset contains over 150 4G traces collected from major Irish and US mobile operators. Traces reflect the use of Android API by providing previously mentioned device-based metrics. The collected trace profiles are a mixture of five different mobility patterns: *static*, *pedestrian*, *bus*, *train*, *car* and *highway*. In [36], authors show 500 records are enough for successfully training ML model. Following their result, we filter out all traces with less than 500 records.

3.2.2 Metrics. We compare the Quantile and raw ML-based throughput prediction techniques described above using two key metrics: the *absolute value of residual error* (ARE) and *coefficient of determination* (R^2).

ARE is the ratio of absolute residual error and actual throughput, where the residual error is the difference between actual and predicted throughput. The following equation defines ARE:

$$ARE = \frac{|max(10, R_i) - max(10, \hat{R}_i)|}{max(10, R_i)} \times 100\,, \tag{1}$$

where R_i and $\hat{R}_i$ is the actual measured throughput and predicted (estimated) throughput (in *Kbps*), respectively. To avoid cases when actual bandwidth drops to zero causing zero division in equation 1 we bound all values less than $10Kbps$ to $10Kbps$.

R^2 score (0-1) is a measure of the goodness of a model compared to a naive model. e.g., a R^2 score of 0.8 (respectively 0.9) implies that the naive model has five (respectively ten) times higher error than the model in question. The following equation defines R^2:

$$R^2 = 1 - \frac{\sum_{i=0}^{N}(R_i - \hat{R}_i)^2}{\sum_{i=0}^{N}(R_i - \bar{R})^2}\,, \tag{2}$$

where N is the number of samples in the test dataset, and $\bar{R}$ the average throughput for the test dataset.

When evaluating performance of RF, it is crucial to analyze generalization error (also called out-of-sample error), which represents the model's ability to represent *unseen* data. Generalization error is a composition of bias, variance, and noise. Bias represents error introduced by approximation of a more complex problem with a simpler model. In other words, if the model assumes a certain relationship for data which does not reflect the actual data generation mechanism, we have bias in our estimate. To capture this, we use training error (i.e., ARE metric for our specific case). However, low training error does not necessarily translate to low bias. Low training error could be a consequence of limited data that do not entirely capture the nature of actual problem. In this case, we have low training error but significant bias. Variance represents the variability of error as we vary the dataset (e.g., re-sampling the training set). We quantify generalization error using cross-validation (CV) data to test a model. If the resulting error is low, then the model is said to have a small generalization error, implying that it can successfully predict new values on unknown data. A high error indicates that the model *overfits* the training data, and is thus only capable of predicting values based on data similar to the training data. Desirable outcome is to have both low training and CV errors.

Unless otherwise noted, we tune RF algorithm and estimate its quality using 10-fold cross-validation[3]. Cross-validation is computationally more expensive than alternative techniques like "holdout", but it guarantees higher accuracy [6].

In our evaluation, we explore a large set of parameters (e.g., horizon, history length, and summarization techniques). Furthermore, a majority of experiments are done in a mobile case (**highway** scenario), as it is the most challenging environment (see section 3.5). Hence, we create a *funnel-based* approach where we progressively fix some parameters after investigating their impact on throughput accuracy as presented in the following subsections.

3.3 Impact of the Summarization Technique

Figure 4 shows the ARE for two approaches: feeding raw input, and applying quantile summarization technique to the input data.

[3]We use *sckit-learn* tool as our main ML framework (https://scikit-learn.org)

These two approaches are compared with various combinations of PxFy; i.e., history and horizon duration.

Regardless of the prediction horizon, quantile summarization technique achieves lower ARE (higher prediction accuracy) than raw technique. This difference gets bigger with a longer metric history. For example, with a prediction history of 20 seconds (P20Fx) the quantile technique lowers the ARE compared to raw by 20% (75^{th} percentile) and 40% (90^{th} percentile). For R^2 score, we observe a similar trend as for ARE, e.g., a value of 0.99 for large history when using the quantile strategy which is a 0.05 boost compared to the raw approach.

To understand the causes of different prediction accuracy across different history lengths, we use the standard approach of analyzing the learning curve. The learning curve represents a ratio/difference between training and cross-validation error metric. The choice of error metric is arbitrary, as more emphasis is on the difference obtained from training and CV data. For the following analysis, we choose CoD for the error metric. Next, we investigate both training and Cross-Validation (CV) error (see Section 3.2.2) for RF. Table 1 shows both training and CV error for the RF algorithm above as a function of the history length, i.e., amount of training data considered. RF does not suffer from high bias for any history and horizon combinations. However, for history lengths shorter than eight seconds, the RF model relatively overfits the training data. This effect is countered as the history length increases.

Table 1: Learning Curves for RF algorithm as a function of history interval and 12-second horizon

Px =	**4s**	**8s**	**12s**	**16s**	**20s**
Train sc.	1.0	1.0	1.0	1.0	1.0
CV sc.	0.93	0.97	0.98	0.99	0.99

3.4 Impact of Prediction Horizon and History Length

Figure 4c also shows that throughput prediction accuracy improves when increasing the prediction horizon. At first, this result appears counter-intuitive as one would expect that predicting throughput for a near future should be easier than for the more distant future. This observation is true for classical estimation techniques, such as EWMA, AVERAGE, and MEDIAN (Figure 2) which relies solely on past throughput values and coarse granularity of history samples to predict future capacity.

However, when predicting the *average* throughput over the next x seconds by using both radio metrics and throughput, overall accuracy improves with the longer horizons. Reason for improved accuracy is that larger horizon results in variance decrease among throughput values. Figure 5a illustrates the effects of averaging over different horizons resulting in a flatter throughput curve.

Similar to the increasing horizon, our analysis also suggests a decrease of ARE as we increase the history length. To confirm this, Figure 5b shows the ARE as a function of increasing history length. The figure shows that increasing history length is beneficial in term of ARE reduction up to a *saturation point* of 20 seconds, beyond which the ARE reduction is marginal. Furthermore, similar observations hold for 4 and 8-second horizons. The same trend can also be seen for the R^2. In [36] authors show that increasing history decreases prediction accuracy countering results of our own. Authors argue that having large history results in predictor having less power in reacting to sudden changes in the wireless channel. This intuition comes from the inability of sudden changes to be reflected when averaging over a large history period. However, we argue that having multiple measures of variability helps in countering this effect. e.g., 25^{th}, percentile can capture small changes in link variability. Based on this result, in the following, we consider history length up to 20 seconds.

3.5 Different Mobility Patterns

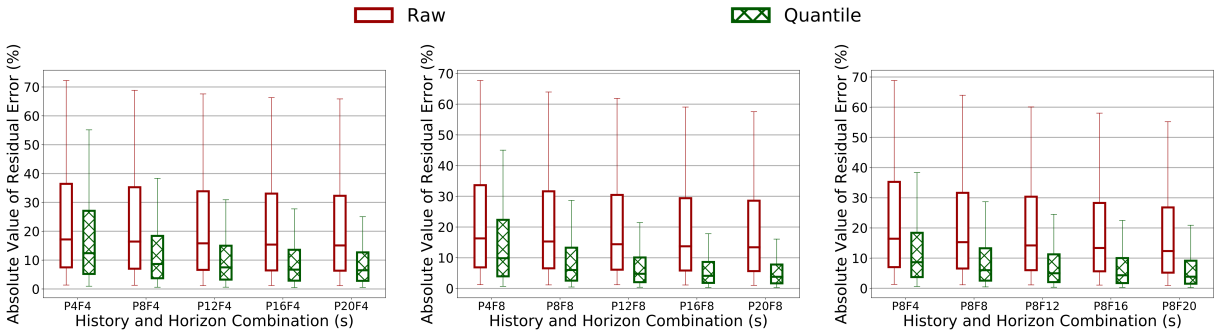
We analyze the performance of our proposed ML technique across different mobility patterns. Table 2 shows average, standard deviation and coefficient of variation (CoV) for all five mobility patterns. Intuitively, a static case has the lowest standard deviation. We expect that static case should yield higher prediction accuracy due to lower variation in throughput values. The larger standard deviation implies a throughput time series that is less "stable" around the mean. The higher variability of the mobile scenarios (we classify bus, pedestrian, car, and highway as mobile cases) is due to *environmental* changes, e.g., channel and cell load. Intuitively, predicting a throughput with lower variation is an easier challenge; we further quantify this observation in the upcoming analysis.

Table 2: Throughput stats for various mobility patterns

Mobility Pattern	Mean (Mbps)	Std (Mbps)	CoV
Static	12.37	5.04	0.4
Pedestrian	10.22	7.59	0.74
Bus	13.97	12.35	0.88
Car	16.29	12.53	0.77
Highway	14.56	12.65	0.87

Figure 5c shows ARE values for static and mobile use-cases. We fix the prediction horizon to 12 seconds but vary the history duration. Overall, the figure shows much better accuracy (lower ARE) in the static scenario. We compare the influence of history length on accuracy for static and mobile cases. With a history length of 4 seconds, 90% of time prediction error is less than 30%, for static case, while for the highway case this increases to 43%. However, extending history to 8 seconds (we use 4 seconds as a threshold to exploit the full benefits of the *quantile* approach), benefits both mobile and static cases, as the 90^{th} percentile of ARE drops by 20% on average for different mobile cases, while in static scenarios this drop is 16%. Increasing history follows the same trend, e.g., 90^{th} of ARE decreases by 40% and 46%, for the mobile and static case, respectively. Nevertheless, the pattern changes for history length beyond 12 seconds, as relative error difference becomes more prominent (e.g., 20-second history lowers 90^{th} percentile by 74% and 71% for static and mobile, respectively). Among mobile cases, overall *highway* scenario shows the highest prediction error. However, the difference between different mobile patterns is negligible.

We observed similar trends for other values of prediction horizon, e.g., for P20F8 the 90^{th} of ARE for static and mobile cases is 12% and 16%, respectively. Similarly, for P20F4 the 90^{th} of ARE for static and mobile cases is 19% and 26%, respectively.

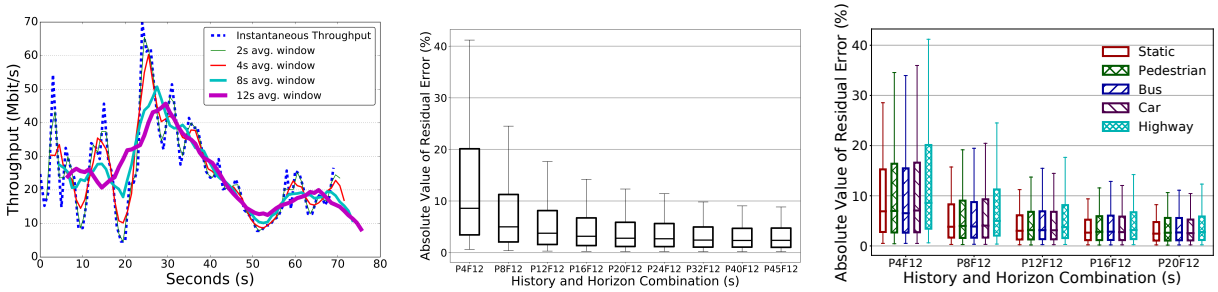


(a) ARE values as a function of history (horizon = 4 seconds)

(b) ARE values as a function of history (horizon = 8 seconds)

(c) ARE values as a function of horizon (history = 8 seconds)

Figure 4: Comparison between different data summarization techniques



(a) Prediction horizon analysis

(b) ARE for 12s horizon and varying history length

(c) Comparison of ARE for various mobility use-cases (PxF12)

Figure 5: Prediction horizon, long history and different mobility patterns

Our main takeaway is that utilization of quantile technique allows high prediction throughput accuracy for longer prediction horizons regardless of mobility environment.

3.6 Metric Contribution to Prediction Accuracy

We investigate the importance of metrics in throughput prediction. Instead of reporting individual metric (the reader is referred to [36] for analysis of each feature individually), we divide them into three groups and report the importance of each group.

We divide metrics into the following groups: *throughput* (which includes the history of both downloads and uploads throughput values), *radio* (which consists of the history of RSRP, RSRQ, SNR, etc.) and device *velocity*.

Table 3a shows how feature importance changes as we vary the history length. For the P4F4 case, historical throughput contributes to 71% of future throughput prediction, and radio metrics and velocity contribute 25% and 4%, respectively.

With an even longer history, the quantile approach can finally be applied, and now we get a greater contribution from radio metrics. As the history increases from 2 to 20 seconds, radio metrics importance increases to 41%, while throughput importance drops to 53%. Table 3b shows that if we fix a history length, then throughput importance goes down with longer horizons. e.g., for P4, the importance of throughput goes down from 71% for F4 to 57% for F12. The drop is significant, and we have observed similar trends for other values of history length as well.

Table 3: Feature Importance for PxF4 and P4Fx cases

(a)

	P4F4	P8F4	P20F4
Radio	25%	31%	41%
Throughput	71%	65%	53%
Velocity	4%	4%	6%

(b)

	P4F4	P4F8	P4F12
Radio	25%	32%	36%
Throughput	71%	62%	57%
Velocity	4%	6%	7%

E.g., in the P20Fx scenario, throughput importance drops from 53% to 48% to 44% for 4-second, 8-second, and 12-second horizon, respectively.

Finally, when predicting for longer horizons, ML model learns more on non-throughput metrics for making a more accurate prediction. This result explains why relying only on throughput is not a good indicator of distant throughput in a highly mobile scenario.

4 VIDEO EXPERIMENTS IN A CONTROLLED ENVIRONMENT

In this section, we evaluate the impact of improved throughput estimation on the streaming performance using a lab testbed. We first present our testbed followed by streaming performance results.

4.1 Testbed Setup

4.1.1 Testbed Overview. Figure 6 depicts our testbed architecture which consists of a mobile device (Nexus 6 running Android OS version 7.1.1), a wireless access point (AP), and a server (PC running Ubuntu 16.04 equipped with 16GB of RAM and Intel i7 CPU). The

mobile device streams video content from the server through AP. The server also acts as a *traffic shaper* by inserting bandwidth profiles from 4G traces between itself and the AP. This is achieved using Linux *traffic control* (tc[4]). Traffic shaper changes link capacity every second, based on a bandwidth trace file. Simultaneously, *Android Debug Bridge* (ADB) is used to feed offline-calculated[5] throughput prediction values to the mobile device every second by saving a value to a file on a mobile device.

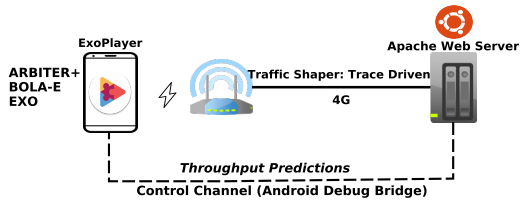


Figure 6: Testbed Architecture

The mobile device runs a video player built using ExoPlayer. ExoPlayer supports the DASH standard via a stand-alone library, and also provides a default adaptation algorithm, which we refer to as EXO. In addition to EXO, we have also ported BOLA-E from dash.js [6] and implemented ARBITER+ [38]. The workload videos comes from the publicly available dataset [18]. Videos are split in 4-second and 8-second chunks and encoded with ten representation rates: 231, 369, 553, 744, 1044, 1748, 2349, 3006, 3856, 4310 *Kbps*.

4.1.2 Video Streaming Algorithms. Many HAS algorithms can be found in the literature [22]. For our evaluation, we select three algorithms: ARBITER+, BOLA-E, and EXO. Selected algorithms use information from both bandwidth estimators, as well as from buffer occupancy when deciding on the rate of the next chunk.

As a bandwidth estimator, ARBITER+ uses the exponential moving average (EWMA) of the last ten chunks rates. Alongside EWMA Arbiter+ employs two additional rate scale factors, to track variation in throughput samples and buffer occupancy. The first factor tracks variation in throughput samples and reduces the estimated rate if bandwidth fluctuation increases. The second one tracks buffer occupancy drain and lowers the rate if the buffer is too low to prevent stalls. However, for higher buffer levels this factor will increase the bandwidth estimate, thus slowing buffer saturation.

BOLA [25] is a buffer based algorithm that relies on Lyapunov optimization to maximize video rate and minimize stall (rebuffering) events. Utility function increases with average bitrate, while the increase in stalls reduces it. However, the algorithm is flexible to allow optimization of different QoE metrics (by defining different utility functions). BOLA-E [24] introduces a throughput estimate to improve startup, seek and low-latency performance of BOLA. The throughput estimate is average of the last five chunk delivery rates.

Finally, EXO algorithm calculates median of a specified number of recent chunk rates (unlike previous two approaches, EXO uses the sum of chunk sizes to decide a number of the last downloaded chunks). Intuitively, median (resistant to outliers) also represents a slightly less conservative estimate compared to the harmonic mean.

For buffer length, we use the recommended values for each algorithm (60-seconds for ARBITER+, 32 seconds for BOLA-E, and 30-seconds for EXO). The initial delay is set to two chunks. After a stall event, a play is resumed after one chunk finishes downloading.

4.1.3 Throughput Prediction Module. Our experiments are based on a subset of the 4G dataset traces to illustrate the benefit of prediction. We filter out traces where average throughput is greater than 6 Mbps for the first five minutes of a trace (streaming is five minutes long). Our rationale being that when bandwidth is so high, any rate adaptation algorithm will be able to stream the highest video quality (4.3 Mbps in the used video dataset) and the benefit of prediction would not be evident. We ended up with 26 traces after this filtering. For characterization of selected traces, we considered standard deviation of throughput within a trace. We sort traces based on standard deviation of BW. Table 4 shows throughput statistics for the top 20% and bottom 20% of sorted traces.

Table 4: Throughput for selected traces

Type	Avg (kbps)	Std (min - max, kbps)
Low-variable traces	1656	591 - 1166
High-variable traces	4451	4010 - 6447

Majority of traces with high BW variability are collected in the highly mobile environment (car), while traces with low BW variability were collected while devices were static or moving at low velocity (pedestrian).

We implement the throughput prediction module using our proposed prediction approach. When testing the streaming performance using one of the selected traces, this trace is eliminated from the training set of the prediction engine used in this experiment. The prediction engine is then used to identify the predicted throughput for every record in the trace. The predicted throughput is stored offline and provided to the video client during the experimentation. In a nutshell, we keep tested trace out of training procedure to ensure unbiased prediction values.

The presented prediction setup is close to reality as it does not imply the knowledge of the transportation mode. Note that the prediction engine is based on traces with mixed mobility patterns. We set the history to 20 seconds and tested different horizon values from 12 seconds-32 seconds. The choice of horizon values was driven by results in [21], where authors show that longer horizons benefit HAS players more than shorter horizons.

Figure 7 shows ARE across five different prediction horizons with history set at 20 seconds. Mixing mobility patterns does not change the performance trends established in Section 3.5. The prediction accuracy increases with the horizon. For the 12-second horizon, 90^{th} percentile ARE is less than 16%. Furthermore, this error drops below 10% for the longest, 32-second horizon.

4.1.4 Integrating Predicted throughput in HAS Algorithm. The predicted throughput can be integrated into the adaptation logic in two different ways. First, the predicted throughput may replace the entire throughput *estimation* in the algorithm (*E-type*). Alternatively, the predicted throughput may be used to replace the estimated throughput *samples* (*S-type*). In [21] authors show that for ideal prediction algorithms with more conservative bandwidth estimation (harmonic, median) have higher improvement with direct estimate while algorithms with more aggressive bandwidth estimation (EWMA) prefer feeding prediction as a sample. As a

[4]https://wiki.debian.org/TrafficControl
[5]Note that the cellular TRX is not used in the controlled experiment.
[6]https://github.com/Dash-Industry-Forum/dash.js/wiki

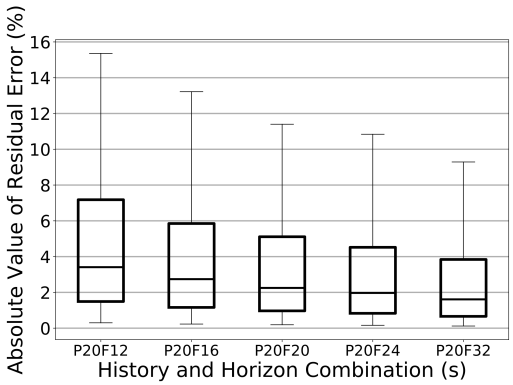


Figure 7: ARE accuracy for five different prediction horizons (mixed-mobility ML model)

result, we use prediction values as a direct estimate for the EXO and BOLA-E, while for the ARBITER+ we feed values through BW estimation module.

4.1.5 Video QoE Models. To evaluate the performance of HAS algorithms, we analyze standardized QoE metrics, such as average video bitrate, switching behavior (e.g., stability), stall frequency and duration. However, to compare algorithms performance these metrics cannot be studied independently. For example, an algorithm achieving the highest average throughput but frequent stalls is inferior to a more cautious algorithm with no stalls, as it provides a better end-user experience. We thus resort to using two video QoE models developed for HAS. These models blend individual QoE metrics to compute a score representing user QoE. Both models are derived from subjective testing of users grading video clips with various induced impairments. The first model (*Yao QoE*) was derived from data collected in a lab environment [11] and is limited to five minutes. Such limitation does not apply to the second model which relies on data crowd-sourced from users watching videos posted on a website [30]. However, in their evaluation, the authors did not consider stall events. As a result, we select an enhanced model [17] which extends the preceding model with stall information (*Clay QoE*). The score derived from these QoE models can be summarized by the following equation:

$$QoE_{score} = \nu \times QoE_{max} - (\kappa_{TQ} \times I_{TQ} + \kappa_{VQ} \times I_{VQ}) + \\ + \Upsilon(I_{TQ}, I_{VQ}) \quad (3)$$

Where I_{TQ}, and I_{VQ}, represent temporal and visual quality impairment factors, respectively. Similarly, κ_{TQ} and κ_{VQ} represent their respective weights. Temporal quality impairments refer to degradation due to initial delay and stall events (stall number and stall duration). Analogously, visual quality impairments take into account average rate and switching behavior. QoE_{max} indicates the maximum value (score) of QoE or growth factor depending on QoE model. Similar to impairment weights, ν is weight for the QoE_{max} score. Finally, $\Upsilon(I_{TQ}, I_{VQ})$ represents a cross-effect function of impairment factors occurring simultaneously. When multiple impairments happen, their cumulative subjective effect is not simply the sum of individual impairment [11]. Function Υ compensates for this effect.

The Yao QoE score starts at 100 and is reduced by impairments, compensated by Υ function of the stall, switching and initial delay impairments. The Clay QoE initial score is based on average rate, reduced by impairments capturing stall and switching impairment. Clay QoE doesn't include cross-effect compensation function Υ.

We analyze both QoE models across different traces allowing us to make the following observations: Both models perceive stall impairments similarly with high correlation (0.9); Models calculate switching impairment differently (Clay uses standard deviation between rates, while Yao relies on difference in Video Quality Metric (VQM) between chunks); Unlike Clay, Yao uses cross-effect compensation function which limits negative impact of multiple impairments; While Clay QoE is calculated over the entire session, Yao QoE calculation is split into 1-min windows, for which QoE score is calculated separately. Total QoE equals arithmetic average of five windows.

Because of the observations mentioned above, we decided to use the geometric mean of the two models to represent an overall QoE score. We chose geometric mean instead of the arithmetic mean because model scores are on a different scale. Table 5 summarizes QoE metrics and their notation.

Table 5: QoE Metrics Notation

Metric	Summary
$Bitrate$	Average bitrate
$Stability$	Average stability, equal to $1 - i$ with i being instability as defined in [9]
$Stalls_{num}$	Average number of stalls
$Stalls_{dur}$	Average stall duration
QoE	Geometric mean of Clay and Yao QoE

4.2 Streaming Performance Results

We evaluate the streaming performance without prediction as a base case and with prediction using different horizon duration. We repeated such evaluation for two scenarios including 4-second and 8-second chunk duration. The shown performance results represent the average of 10 runs.

For the 4-second scenario, Figure 8 plots the relative improvement in performance metrics, for the evaluated algorithms, relative to the no prediction case. Hence, a larger relative improvement in the streaming bitrate means a higher rate while a larger relative improvement in the number of stalls means fewer stalls. The performance metrics of the no prediction case for every algorithm are shown in the white boxes just above the x-axis. Figure 8 shows that integrating our prediction noticeably improves the QoE metrics of different algorithms. Specifically, prediction enables all algorithms to reduce/eliminate stalls. Additionally, prediction allows HAS algorithms to improve switching stability. In particular, average stability can be improved by 15%-47%. Furthermore, improving the accuracy of bandwidth estimation enables the algorithm to enhance their selected chunk quality. For example, integrating prediction fixes throughput underestimation with EXO leading to a higher chunk quality (by 16%). On the other hand, incorporating the prediction with ARBITER+ results in a negligible lower average chunk quality (2%). All these improvements add up leading to boosting the overall user QoE by 8%-27% in the 4-second chunk scenario. It is evident that BOLA-E is the least beneficiary of the compared algorithms.

However, this is expected due to its design relies on a buffer level as the main quality selection decision and only uses throughput estimates in very limited cases as illustrated in Section 4.1.2.

In the 8-second scenario, Figure 9 depicts the relative improvement of the performance metrics for the evaluated algorithms. Similar to the 4-second case, the prediction shows a positive impact on all metrics in the majority of the traces. Overall, QoE can be improved by 19%, 13% and 33% for ARBITER+, BOLA-E, and EXO, respectively. This improvement is higher than that attained in the 4-second scenario. We explain this by the fewer opportunities to change the quality and react to sudden changes to channel capacity in the longer chunk case. In both 4-second and 8-second scenario, EXO features the highest relative improvement in QoE score. This improvement is attributed to the increase in the average bitrate (15% improvement compared to 1-5% for ARBITER+ and BOLA-E).

Identifying the optimal horizon duration is an essential design parameter. For the 4-second scenario, the algorithms show a similar QoE improvement for 20-24 second horizon. In the 8-second chunk duration scenario, algorithms show distinct performance as the prediction horizon increases. ARBITER+ shows the best QoE relative improvement with a 32-second horizon, while BOLA-E and EXO achieve the best performance with a 24-second horizon. Extending the horizon results in averaging over a longer period and thus reducing variability between subsequent prediction values. This leads to an improved switching performance. Additionally, longer horizon enables a client to proactively switch quality and avoid stalls when the throughput drops for a relatively long time that can deplete the buffer. However, increasing horizon will eventually lead to client inability to adapt to sudden changes in channel capacity. This can be seen in case of 8-second chunk and EXO algorithm. For the 32-second horizon, stability improves by 33%. However, this stability leads to a decrease in stall performance (compared to other horizons) and a sharp drop in overall QoE.

Finally, there is a discrepancy between improvement in each QoE metric and overall QoE improvement. This is a direct consequence of QoE model we use in this study. To understand this behavior, we look at each QoE model individually. Let's analyze particular trace where we have high improvement in rebuffering performance (ARBITER+, 4-second chunk duration). Reduction in a number of stalls and total stall duration results in 35x and 1.9x higher stall impairment for Yao and Clay, respectively. For switching, impairment is 1.6x higher for Clay, compared to 1.25x for Yao. While Clay produces similar tradeoff between stall and switching impairment, Yao gives much higher weight on the stall reduction. As a result, the prediction improves Yao QoE by 40%, while for the Clay, lowers QoE by 44%. Overall, QoE improves by 14%. A similar observation holds for BOLA-E.

5 REAL-TIME PREDICTION

Motivated by results of the lab experiments, we implemented our prediction engine inside mobile devices leveraging the Android API and an existing Java ML library. For the ML library, we opt for Weka[7], a software framework that has an extensive collection of state-of-art machine learning algorithms implemented in Java. While not explicitly designed to run on mobile devices, it represents a good starting point for testing the initial prototype.

For the collection of radio metrics and device velocity, we use classes and methods detailed in Table 6. Values are collected periodically, every 1-second in a separate thread inside ExoPlayer. We store all metrics in FIFO queues with size limited to 20 values per queue. We trained an ML model (Random Forest) offline and ported to the mobile devices. For the HAS algorithm, we select EXO using the same parameters as outlined in Section 4.1.2. We use a 20-second prediction horizon (direct estimate). The prediction value is generated in the following way: every time a decision for the next quality needs to be made, the adaptation logic requests a prediction value. This value is generated by creating Quantile statistics based on current state of FIFO queues, followed by a call to the model itself with statistics as input. Finally, the model returns a prediction value for the next 20 seconds.

Table 6: Android API classes used for collecting radio and throughput metrics

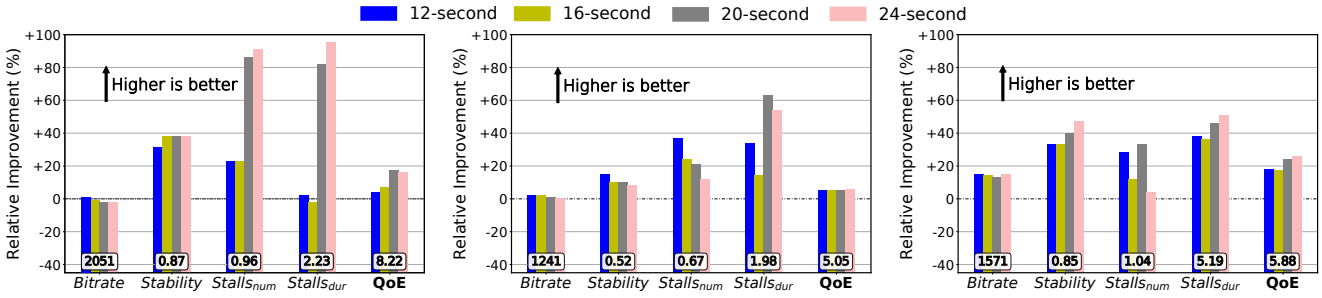
metrics	Class/Method
RSRQ	*CellInfoLte/getCellSignalStrength().getRsrq()*
RSRP	*CellInfoLte/getCellSignalStrength().getDbm()*
CQI	*CellInfoLte/getCellSignalStrength().getCqi()*
SNR	*CellInfoLte/getCellSignalStrength().getRssnr()*
Velocity	*LocationManager*
Throughput	*TrafficStats*

We perform 56 static and mobile field tests in a real cellular network. Each experiment consists of two mobile devices (same model) streaming the same video content side by side. One mobile device stream content with no throughput guidance, while the other one uses our throughput prediction as outlined above. To minimize non-radio related effects, we perform all tests in the early morning while assuming the network is not busy. The following sections explain the device and model limitations we faced while implementing the prediction engine in a mobile device.

Device Limitation: We leverage the standard Android library for capturing channel metrics. However, implementation of these callback functions depends on the manufacturer of the mobile system on a chip (SoC) chipsets. Also, not all parameters are reported for different cellular technologies (2G/3G/4G). We use Samsung J5 mobile devices, as the Exynos chipset implements almost all Android callback methods for reporting channel metrics. Furthermore, loading and running ML model inside a mobile device can be challenging. Due to hardware limitations, loading of the appropriate model can take up to several minutes. e.g., training Random Forest model on all traces results in a large model (400MB) which can not be loaded in the mobile device. As a tradeoff, we limit the number of trees (30) and tree depth to generate a smaller model. This tradeoff results in accuracy decrease of the model. As a result, prediction error (90^{th} percentile) increases from 13% to 21%.

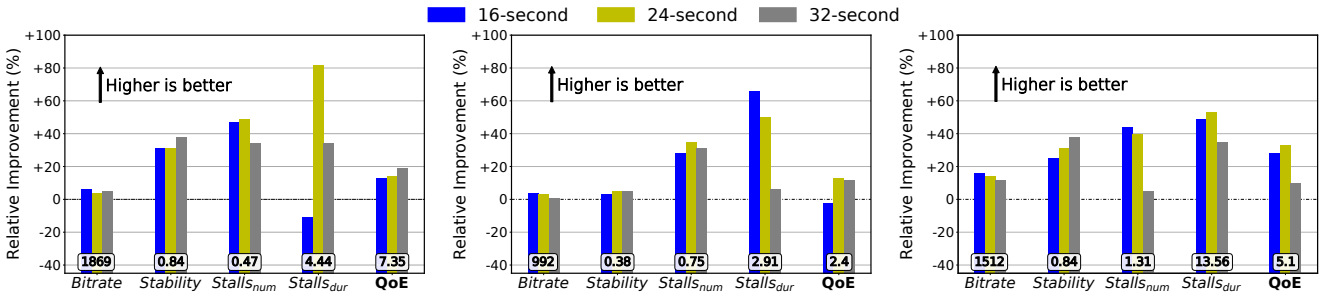
Experiment Limitation: Running two devices side-by-side at the same time does not necessarily mean same channel and environment conditions. There are a couple of limitations we faced during these trials. Firstly, devices do not necessarily report the same values for metrics. Second, even in the static case, phones can be connected to different eNodeBs or same eNodeB but different cell

[7]https://www.cs.waikato.ac.nz/ml/weka/



(a) ARBITER+ (b) BOLA-E (c) EXO

Figure 8: Relative improvement of different QoE metrics across different HAS algorithms for the 4-second chunk duration (*The metrics are normalized to the no-prediction case with numbers in white boxes representing the value of each metric for the no-prediction case*)



(a) ARBITER+ (b) BOLA-E (c) EXO

Figure 9: Relative improvement of different QoE metrics across different HAS algorithms for the 8-second chunk duration (*The metrics are normalized to the no-prediction case with numbers in white boxes representing the value of each metric for the no-prediction case*)

sector. While we tried to minimize these occurrences, we have little or no control over this in mobile cases.

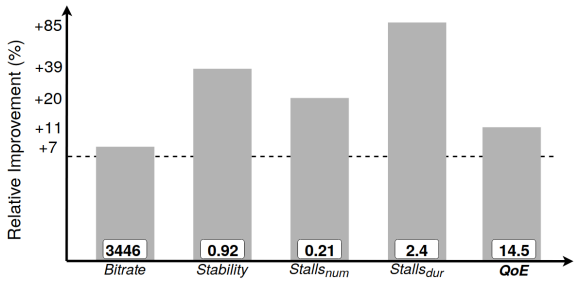


Figure 10: Relative improvement of different QoE metrics in real cellular network with respect to the no-prediction case (EXO algorithm, 4s chunks)

Figure 10 shows similar trend for QoE metrics as in Section 4. However, there exists one major difference between experiments performed in the previous section and experiments in a real cellular network. While we selected a certain subset of traces in controlled experiments (in particular, traces with an average rate close to highest video rate), in a real environment, the majority of tests were conducted in a high channel capacity environment (with the average rate greater than 6 Mbps). This is most evident in the case of bitrates. Average bitrate across all session is 3.4 Mbps (compared to 1.5 Mbps in a controlled environment). As a result, the impact of prediction is limited as the highest rate is 4.3 Mbps, leaving less space for improvement. Still, prediction improves all QoE metrics. In particular, improvement in bitrate is only 7%, capping QoE improvement to 11%.

6 DISCUSSION

Optimal use of prediction: In our experiments we simply replaced the bandwidth estimator with a prediction value and constant horizon. However, the optimal horizon is unknown in advance. Furthermore, none of the tested algorithms is optimized to take full benefit from more accurate predictions. In previous studies [13, 40], authors show that HAS algorithms can be fully tuned based on prediction horizon. Furthermore, some of the algorithms take multiple horizons into account when deciding for the next quality. We argue that having various horizons, i.e., short and long horizon can improve user experience even further than only taking one constant horizon. However, generating multiple horizons in real time is challenging. To overcome this issue following section introduces motivation for a transition towards deep learning algorithms.

Possible model improvement by deriving optimal high-level features: Deep learning frameworks (TensorFlow Lite for Android and Core ML for Apple iOS) are optimized to run on mobile devices. Deep learning architectures may help in obtaining even more accurate predictions. We showed that representing history with multiple measures of variability helps drive down prediction error. However, we use standard statistical measures, which are not necessarily the optimal ones. Also, the same statistics are used across all metrics. However, optimal statistics for one metric may not necessarily be

optimal for other metrics. Instead of "handpicking" high-level features from *raw* input, using neural networks with multiple layers can automate extraction of more significant features from input data. This observation leads us to deep learning algorithms, and in particular recurrent neural networks. Sequence-to-sequence neural networks [27] have a broad reach in language translation. The main power of these networks lies in the ability to summarize a sentence from one language by a couple of critical values (encoder) which then can be used to translate it into another language (decoder). Inspired by this approach, we believe similar architecture can be employed for throughput prediction, where the history of each metric will be summarized by optimal measures of variability independently of each other.

Enhancing model with network metrics: We use device related metrics to improve throughput prediction. However, relying only on device-based metrics is not sufficient in all situations. During busy hours, prediction values can overestimate throughput capacity as a correlation between user channel conditions and available throughput deteriorate. To alleviate this issue, network-related metrics are needed. e.g., cell load would indicate how busy cell is, preventing overestimation and increased probability of rebuffering events. Also, network operators can inflict throttling in a network [10] during busy hours which can skew prediction system completely.

Device vs. in network prediction: In our experiments, we took the approach of making predictions at the device itself. While this approach brings benefits regarding scalability and possibility to retrain model to suit better for local conditions, its prediction power can be limited for reasons outlined in previous sections. Unlike device-based prediction, in-network prediction does not suffer from limited computational power. The device can send its metrics periodically and request forecast when needed. However, this implies having coordination between end-device and network provider/service.

7 RELATED WORK

Cellular networks represent the most challenging environment for throughput estimation leading to the majority of work done in this area. Existing solutions for throughput prediction can be grouped into two general categories: *non-machine learning* and *machine learning* approaches. Furthermore, additional useful categorization can be made regarding prediction horizon (e.g., short, the order of milliseconds, or medium, the order of seconds) and whether solutions are application-specific (e.g., video or voice).

Non-machine learning approaches include different techniques for throughput estimation. Lots of work has been done in order to improve TCP performance over cellular networks [4, 29, 31, 32].

Active measurements were also proposed to estimate throughput, round trip time, and packet losses by sending carefully crafted sequences of short data packets [8]. Other studies rely instead on passive measurements [12, 26] using a device's instantaneous radio channel quality indicator (CQI) and discontinuous transmission ratio (DTX) for throughput estimation.

Applying *machine learning* in throughput forecasting has slowly gained momentum over the years. Most of the work is concerned with improving TCP estimates over shorter horizons [16, 31, 34].

Introduction of adaptive streaming and rise of multimedia streaming consumption over a cellular networks motivated investigation of ability to predict longer horizons. Sayeed et al. use an autoregressive ARIMA based time-series model taking very specific parameters such as Signal-to-Interference and Noise ratio (SINR) and Modulation and Coding Scheme (MCS) as inputs to first predict the number of received bits per physical resource blocks (PRB) and then translate that to effective throughput [23]. Their experiments are evaluated for a stationary device under different channel configurations. Also, some solutions are crafted with a video application in mind. For example, Zou et al. propose an algorithm for HTTP adaptive streaming that relies on an accurate forecast of average throughput [40]. Their solution leads to significant improvements in video Quality of Experience (QoE) compared to other state-of-the-art approaches [7, 9]. In a similar vein, Mangla et al. design an adaptation algorithm that takes prediction errors into account when making a decision for the next chunk [13]. Some solutions look for patterns of similarity between sessions to predict what QoE the new session will have, where similarity is determined through coarse-grained geographic and network features, not precise network performance measurements [26]. Xie et al. propose a framework for HTTP adaptive streaming application where authors leverage LTE resource structure by monitoring available bandwidth based on PRBs utilisation of the cell, enabling more accurate estimation of available bandwidth. Their approach enables HAS client to track changes in available bandwidth more accurately resulting in high video quality while minimising stalling rate [33].

Machine learning has also been used to develop adaptation logic for video streaming algorithms. In [15] authors propose a reinforcement neural network backed algorithm that learns from real traces best strategy for adapting to different network conditions.

In our work, we tackle the throughput prediction problem for video streaming applications using a *machine learning* approach, considering variable prediction horizons and realistic mobility conditions. Similar to our work, Yue et al. [36] also investigate prediction using only device-based metrics. However, they rely on UDP based technique for measuring throughput bandwidth and use the average as their summarization technique. For the horizon, they consider one second. While the framework is based on measurement of radio channel metrics from Android OS they do not quantify its impact on video streaming performance. Further, they compute prediction accuracy using the holdout method while we rely on cross-validation, a more reliable method for estimating model performance. Similar to our evaluation, Xie et al. conducted experiments on real mobile devices in a real cellular network [33]. However, in their approach, they used specialised hardware (Universal Software Radio Peripheral - USRP) for monitoring wireless channel between device and eNodeB and estimating PRB utilisation. Furthermore, all the calculations are done on a laptop which feeds the information back to the device through USB cable. In our real-world experimentation, all the measurements and decision is done at the mobile device.

8 CONCLUSIONS

It is known that cellular radio access networks exhibit highly variable conditions due to a variety of factors. In this paper, we address the problem faced by applications such as video streaming in trying to estimate the available future throughput in the cellular network.

Prior work has focused on the use of a small set of performance metrics gathered by an end-user device to make predictions up to one second. In this paper, we present a thorough quantitative study of throughput prediction in a real cellular network. We combine machine learning techniques with radio channel metrics summarized by a novel quantile abstraction technique to achieve low throughput prediction errors (90% of errors below 13%). By utilizing our abstraction technique, we were able to capture trends and variation in metric data accurately in the environment where metrics are updated/available at fine time granularity. Having more accurate predictions allow us to improve performance of three adaptation algorithms. All tested algorithms improve all QoE metrics when using prediction. Notably, prediction reduces stalls by up to 85%, and bitrate switching by up to 40%, while maintaining or improving video quality. As a result, QoE score improves significantly, by up to 33%.

ACKNOWLEDGEMENTS

The authors acknowledge the support of Science Foundation Ireland (SFI) under Research Grant 13/IA/1892.

REFERENCES

[1] A. Bentaleb, A. C. Begen, S. Harous, and R. Zimmermann. 2018. Want to Play DASH?: A Game Theoretic Approach for Adaptive Streaming over HTTP. In *MMSys*. ACM.
[2] L. Breiman. 2001. Random Forests. *Machine Learning* 45, 1 (2001).
[3] L. De Cicco, V. Caldaralo, V. Palmisano, and S. Mascolo. 2013. ELASTIC: A Client-Side Controller for Dynamic Adaptive Streaming over HTTP (DASH). In *IEEE International Packet Video Workshop*.
[4] S. Ha, I. Rhee, and L. Xu. 2008. CUBIC: A New TCP-friendly High-speed TCP Variant. *SIGOPS Oper. Syst. Rev.* 42, 5 (July 2008).
[5] J. Hao, R. Zimmermann, and H. Ma. 2014. GTube: Geo-predictive Video Streaming over HTTP in Mobile Environments. In *MMSys*. ACM.
[6] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. 2009. *The Elements of Statistical Learning, Data Mining, Interference, and Prediction.* Springer.
[7] T.-Y. 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 *SIGCOMM*. ACM.
[8] Manish Jain and Constantinos Dovrolis. 2003. End-to-end Available Bandwidth: Measurement Methodology, Dynamics, and Relation with TCP Throughput. *IEEE/ACM Trans. Networking* 11, 4 (August 2003).
[9] J. Jiang, V. Sekar, and H. Zhang. 2014. Improving Fairness, Efficiency, and Stability in HTTP-Based Adaptive Video Streaming With Festive. *IEEE/ACM Transactions on Networking* 22, 1 (Feb 2014).
[10] A. M. Kakhki, F. Li, D. Choffnes, E. Katz-Bassett, and A. Mislove. 2016. BingeOn Under the Microscope: Understanding T-Mobiles Zero-Rating Implementation. In *Internet-QoE*. ACM.
[11] Y. Liu, S. Dey, F. Ulupinar, M. Luby, and Y. Mao. 2015. Deriving and Validating User Experience Model for DASH Video Streaming. *IEEE Transactions on Broadcasting* 61, 4 (Dec 2015), 651–665.
[12] F. Lu, H. Du, A. Jain, G. M. Voelker, A. C. Snoeren, and A. Terzis. 2015. CQIC: Revisiting Cross-Layer Congestion Control for Cellular Networks. In *HotMobile*. ACM.
[13] T. Mangla, N. Theera-Ampornpunt, M. Ammar, E. Zegura, and S. Bagchi. 2016. Video Through a Crystal Ball: Effect of Bandwidth Prediction Quality on Adaptive Streaming in Mobile Environments. In *MoVid*. ACM.
[14] T. Mangla, E. Zegura, M. Ammar, E. Halepovic, K.-W. Hwang, R. Jana, and M. Platania. 2018. VideoNOC: Assessing Video QoE for Network Operators Using Passive Measurements. In *MMSys*. ACM.
[15] H. Mao, R. Netravali, and M. Alizadeh. 2017. Neural Adaptive Video Streaming with Pensieve. In *SIGCOMM*. ACM.
[16] M. Mirza, J. Sommers, P. Barford, and X. Zhu. 2010. A Machine Learning Approach to TCP Throughput Prediction. *IEEE/ACM Transactions on Networking* (2010).
[17] S. Petrangeli, J. Famaey, M. Claeys, S. Latré, and F. De Turck. 2015. QoE-Driven Rate Adaptation Heuristic for Fair Adaptive Video Streaming. *ACM Trans. Multimedia Comput. Commun. Appl.* (October 2015).
[18] J. J. Quinlan, A. H. Zahran, and C. J. Sreenan. 2016. Datasets for AVC (H.264) and HEVC (H.265) Evaluation of Dynamic Adaptive Streaming over HTTP (DASH). In *MMSys*. ACM.
[19] D. Raca, J. J. Quinlan, A. H. Zahran, and C. J. Sreenan. 2018. Beyond Throughput: A 4G LTE Dataset with Channel and Context Metrics. In *MMSys*. ACM, 6.
[20] D. Raca, A. H. Zahran, C. J. Sreenan, R. K. Sinha, E. Halepovic, R. Jana, and V. Gopalakrishnan. 2017. Back to the Future: Throughput Prediction For Cellular Networks Using Radio KPIs. In *HotWireless*. ACM.
[21] D. Raca, A. H. Zahran, C. J. Sreenan, R. K. Sinha, E. Halepovic, R. Jana, V. Gopalakrishnan, B. Bathula, and M. Varvello. 2018. Incorporating Prediction into Adaptive Streaming Algorithms: A QoE Perspective. In *NOSSDAV*. ACM.
[22] Y. Sani, A. Mauthe, and C. Edwards. 2017. Adaptive Bitrate Selection: A Survey. *IEEE Communications Surveys Tutorials* 19, 4 (2017).
[23] Z. Sayeed, E. Grinshpun, D. Faucher, and S. Sharma. 2015. Long-term application-level wireless link quality prediction. In *2015 36th IEEE Sarnoff Symposium*.
[24] K. Spiteri, R. Sitaraman, and D. Sparacio. 2018. From Theory to Practice: Improving Bitrate Adaptation in the DASH Reference Player. In *MMSys*. 123–137.
[25] K. Spiteri, R. Urgaonkar, and R. K. Sitaraman. 2016. BOLA: Near-optimal bitrate adaptation for online videos. In *IEEE INFOCOM*.
[26] Y. Sun, X. Yin, J. Jiang, V. Sekar, F. Lin, N. Wang, T. Liu, and B. Sinopoli. 2016. CS2P: Improving Video Bitrate Selection and Adaptation with Data-Driven Throughput Prediction. In *SIGCOMM*. ACM.
[27] I. Sutskever, O. Vinyals, and Q. V Le. 2014. Sequence to Sequence Learning with Neural Networks. In *Advances in Neural Information Processing Systems 27*, Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger (Eds.). Curran Associates, Inc., 3104–3112.
[28] D. Tsilimantos, T. Karagkioules, and S. Valentin. 2018. Classifying Flows and Buffer State for Youtube's HTTP Adaptive Streaming Service in Mobile Networks. In *MMSys*. ACM.
[29] A. Venkataramani, R. Kokku, and M. Dahlin. 2002. TCP Nice: A Mechanism for Background Transfers. *SIGOPS Oper. Syst. Rev.* 36, SI (December 2002), 329–343.
[30] J. De Vriendt, D. De Vleeschauwer, and D. Robinson. 2013. Model for estimating QoE of video delivered using HTTP adaptive streaming. In *IFIP/IEEE IM 2013*.
[31] K. Winstein and H. Balakrishnan. 2013. TCP Ex Machina: Computer-generated Congestion Control. *SIGCOMM Comput. Commun. Rev.* 43, 4 (August 2013), 123–134.
[32] K. Winstein, A. Sivaraman, and H. Balakrishnan. 2013. Stochastic Forecasts Achieve High Throughput and Low Delay over Cellular Networks. In *NSDI*. USENIX, 459–471.
[33] X. Xie, X. Zhang, S. Kumar, and L. E. Li. 2015. piStream: Physical Layer Informed Adaptive Video Streaming over LTE. In *MobiCom*. ACM.
[34] Q. Xu, S. Mehrotra, Z. Mao, and J. Li. 2013. PROTEUS: Network Performance Forecast for Real-time, Interactive Mobile Applications. In *MobiSys*. ACM.
[35] X. Yin, A. Jindal, V. Sekar, and B. Sinopoli. 2015. A Control-Theoretic Approach for Dynamic Adaptive Video Streaming over HTTP. *SIGCOMM Comput. Commun. Rev.* 45, 4 (August 2015), 14.
[36] C. Yue, R. Jin, K. Suh, Y. Qin, B. Wang, and W. Wei. 2017. LinkForecast: Cellular Link Bandwidth Prediction in LTE Networks. *IEEE Transactions on Mobile Computing* (2017).
[37] A. H. Zahran, J. Quinlan, D. Raca, C. J. Sreenan, E. Halepovic, R. K. Sinha, R. Jana, and V. Gopalakrishnan. 2016. OSCAR: An Optimized Stall-cautious Adaptive Bitrate Streaming Algorithm for Mobile Networks. In *MoVid*. ACM.
[38] A. H. Zahran, D. Raca, and C. Sreenan. 2018. ARBITER+: Adaptive Rate-Based InTElligent HTTP StReaming Algorithm for Mobile Networks. *IEEE Transactions on Mobile Computing* (2018).
[39] Y. Zaki, T. Pötsch, J. Chen, L. Subramanian, and C. Görg. 2015. Adaptive Congestion Control for Unpredictable Cellular Networks. In *SIGCOMM*. ACM.
[40] X. K. Zou, J. Erman, V. Gopalakrishnan, E. Halepovic, R. Jana, X. Jin, J. Rexford, and R. K. Sinha. 2015. Can Accurate Predictions Improve Video Streaming in Cellular Networks?. In *HotMobile*. ACM.