Lumos: towards Better Video Streaming QoE through Accurate Throughput Prediction

Gerui Lv[1,2,*], Qinghua Wu[1,2,3,*], Weiran Wang[1,2], Zhenyu Li[1,2,3] and Gaogang Xie[2,4]
[1]*Institute of Computing Technology, Chinese Academy of Sciences, China*
[2]*University of Chinese Academy of Sciences, China* [3]*Purple Mountain Laboratories, China*
[4]*Computer Network Information Center, Chinese Academy of Sciences, China*
Email: {lvgerui, wuqinghua, wangweiran, zyli}@ict.ac.cn, xie@cnic.cn

IEEE INFOCOM 2022 - IEEE Conference on Computer Communications | 978-1-6654-5822-1/22/$31.00 ©2022 IEEE | DOI: 10.1109/INFOCOM48880.2022.9796948

***Abstract*—ABR algorithms dynamically select the bitrate of chunks based on the network capacity. To estimate the network capacity, most ABR algorithms use throughput prediction while recent works start to leverage delivery time prediction. We in this paper examine all components of the predictor for ABR algorithms, i.e., input features, mapping function and output target. We build an automated video streaming measurement platform, and collect extensive dataset under various network environments, containing 2500+ video sessions. Through analysis, we find that most of previous works failed to achieve accurate prediction due to ignoring how application behavior influences application throughput, e.g., the strong correlation between chunk size and throughput. Then we identify underlying factors affecting this correlation, and consider them as features for more accurate prediction. Moreover, we show that throughput is a better target for data-driven predictors than delivery time in terms of prediction error, due to the long tail distribution of delivery time. Based on those above, we propose a decision-tree-based throughput predictor, named Lumos, which acts as a plug-in for ABR algorithms. Extensive experiments in real-world Internet demonstrate that Lumos achieves high prediction accuracy and improves the QoE of ABR algorithms when integrated into them.**

I. Introduction

HTTP-based video streaming (standardized as DASH [1]) currently accounts for the majority of the Internet traffic [2]. In DASH systems, each video is encoded into multiple versions with the same content but different average bitrates (i.e., quality). Each version of video is then segmented into chunks with equal duration (usually 2-10 seconds). Video client runs adaptive bitrate (ABR) algorithms to select bitrate of each chunk based on network capacity and client buffer occupancy, in order to maximize Quality of Experience (QoE), including maximizing video quality and minimizing rebuffering time and quality switch.

Most ABR algorithms use throughput prediction to estimate network capacity [3]–[10]. As an alternative, recent work [11] advocates to predict delivery time of chunks for better QoE. Naturally, two key problems about prediction in ABR algorithms emerge:

1) Input features: what factors assist to achieve better prediction? Take throughput for example, the throughput perceived by application is affected by both network conditions and application behavior [12]–[14]. Traditional throughput predictors for video streaming, whether history-based [3], [10], [15] or learning-based [5], consider throughput fluctuation only as change of network conditions. Recent works [8], [11] start to consider chunk size into prediction. However, chunk size could not represent all application behaviors, such as ON-OFF period [12], [13]. Moreover, none of prior works has investigated how all possible impacting factors exactly affect throughput.

2) Output target: which one of throughput and delivery time is a better target to predict? Since throughput and delivery time can be converted to each other with chunk size given, the two indicators are regarded as the same in representing network capacity. However, while throughput is corresponding to bitrate selection, delivery time is directly used to calculate QoE, it is intuitive to regard delivery time as a more effective indicator for ABR algorithms. There has been no previous work quantitatively comparing these two indicators.

This paper aims to design a better predictor for ABR algorithms by tackling these two problems, and achieves the following contributions:

- We conceptually summarize (§II) and quantitatively verify (§III) all the impacting factors in predicting throughput and delivery time. By analyzing an extensive dataset collected in real-world Internet using our video streaming measurement platform, we make the following observations. Available bandwidth of network is different from application throughput of chunks due to the behavior of video streaming. Specifically, strong correlation exists between chunk size and throughput, which most of previous works overlooked, leading to inaccurate prediction of throughput. In fact, this correlation is deeply affected by the state of client player, the chunk index, and the signal strength of the mobile client platform, all of which should be considered for more accurate prediction of throughput.
- We found that throughput is a better prediction target than delivery time in terms of prediction error (§IV-A). By considering all the considered impacting factors, we construct two data-driven predictors (decision trees and multiple linear regression), to predict the two targets. Experimental results show that predictors for delivery time has relatively larger prediction error, due to the long tail distribution of delivery

*Co-first authors

time.

- We propose Lumos, a decision-tree-based accurate predictor of throughput (§IV), which could be integrated into existing ABR algorithms to improve the prediction accuracy of throughput for better QoE. Three Lumos-assisted ABR algorithms (RB [3], MPC [4], BBA [16]) are evaluated in our real-world video streaming platform (§V). Experimental results indicate that Lumos achieves much better prediction accuracy and Lumos-assisted ABR algorithms outperform the original algorithms in QoE. Moreover, MPC+Lumos improves average QoE by 6.3% over original MPC, and even 19.2% over Pensieve [17], a state-of-the-art learning-based ABR algorithm.

II. Background and Motivation

Background: ABR algorithms determine the bitrate of each video chunk according to the information of network and client obtained when retrieving previous chunks. Existing ABR algorithms can be classified into four categories: rate-based (e.g., [3], [5]), buffer-based (e.g., [16], [18]), mixed (e.g., [4]) and learning-based (e.g., [17]). *Rate-based* approaches and *buffer-based* approaches select the bitrate according to the predicted throughput and the buffer occupancy of the video player, respectively. *Mixed* approaches select the bitrate according to both throughput and buffer level, which are also taken as input in *learning-based* approaches. Note that rate-based and mixed approaches originally requires explicit throughput prediction. Besides, even buffer-based and learning-based approaches tend to rely on throughput prediction when they are deployed in real-world environments, e.g., the change from BOLA [18] to DYNAMIC [7] and from Pensieve [17] to ABRL [9]. Under this circumstance, accurate prediction of throughput (as well as delivery time) is vital to improve the QoE of video streaming.

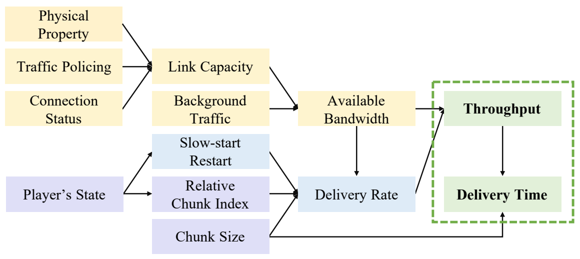
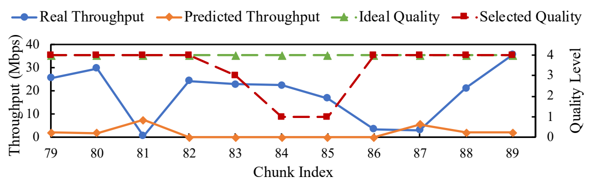
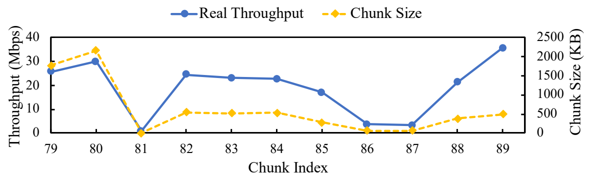


Fig. 1: Factors impacting chunk throughput and delivery time

Fig. 1 shows all factors impacting throughput and thus delivery time of chunks, as well as their relationships. As delivery time can be calculated from application throughput with chunk size given[1], the factors impacting throughput necessarily impact delivery time. Therefore, we focus on all the factors directly impacting application throughput. Application throughput is determined by both available bandwidth of the bottleneck link and delivery rate of the application [14]. Available bandwidth is equal to link capacity minus background traffic, and link capacity is further determined by physical property of the link, traffic policing [19] and connection status between the server and the video player (e.g., ISP, AS [5], CDN layer [10], connection type and signal strength [20]). As for delivery rate, which is controlled by $cwnd$ of the server, essentially depends on both available bandwidth and application behavior. The ON-OFF period is the unique behavior of video streaming [12], [13], indicating that the player requests for chunks periodically rather than continuously after the start-up phase. During the inactivity time of the player (i.e., the OFF period), no data is transferred between the player and the server. If the inactivity time exceeds a timeout (200ms in Linux), the server resets the $cwnd$ to the initial size (e.g., 10MSS), and returns to the slow-start phase [13], which is named slow-start restart [17], [21]. Due to the application behavior and transport mechanism, the request pattern of video player can be divided into two states [12]: the *Buffering-State*, in which $cwnd$ is adjusted continuously in subsequent chunks and the *Steady-State* in which slow-start restart occurs when retrieving each chunk.



(a) Inaccurate throughput prediction decreases video quality



(b) Chunk size and real throughput

Fig. 2: A case of RobustMPC

Motivation: For both existing ABR algorithms and wildly deployed video service, a large gap exists between the bitrate that ABR selects and the available bandwidth [22], [23]. As an example, a real case of a classical ABR algorithm (RobustMPC [4]) is present to show how inaccurate prediction of throughput does harm to QoE. Fig. 2a shows that if the real throughput of one chunk suddenly falls (at the 81st chunk), the predicted throughput of next chunk by RobustMPC will fall as well; but when the real throughput of chunks rises in the future (from the 82nd chunk), the predicted throughput can not react to the change in time. Although the real throughput is high enough for the highest bitrate (not shown in the figure), RobustMPC keeps selecting lower bitrate for 3 consecutive chunks, resulting in unnecessary video quality reduction and fluctuation, and thus degenerated QoE.

One will usually attribute throughput plummeting in Fig. 2a

[1]The application throughput of a chunk is calculated as the chunk size divided by the delivery time. The delivery time is the duration from the time when the first byte of the request is made to the time when the last byte of the response is received [10].

only to fluctuation of network bandwidth, and believes that throughput is difficult to predict [16]. This is partly caused by confusing application throughput with available bandwidth, as many prior works do [3], [5], [17], [24]. However, as illustrated in Fig. 1, many factors that influence throughput are ignored in most exiting prediction schemes. For instance, as shown in Fig. 2b, it seems that throughput changes in the same trend as chunk size does, which was also mentioned in prior works [11], [13]. This indicates that variation of application perceived throughput may be not dominated by network conditions as considered before. Although throughput prediction of video streaming has already been widely investigated [5], [10]–[13], [15], [22], there still lacks of a deep understanding of this issue, which motivates our work in this paper.

III. Prediction Factors

In this section, we investigate the factors by which better prediction is achieved, to select input features for the predictor. We first describe our automated video streaming measurement platform, and the extensive dataset of video streaming collected through this platform in real-world Internet (§III-A). By analyzing the collected dataset, we identify the impacting factors that affect the throughput of video chunks, and quantitatively characterize how they assist in throughput prediction (§III-B). After that, we reveal the correlation between throughput and chunk size, and the impact of various factors on this correlation (§III-C).

A. Collecting Dataset

To collect dataset for analysis, we build an automated video streaming measurement platform following DASH specification. The platform consists of the client which runs the video player, and the server which stores video contents.

Video content: We select Elephant Dream and Big Buck Bunny, 2 raw videos from [25]. Each of them is around 10 minutes. We use FFmpeg [26] to encode them by the H.264/MPEG-4 codec at bitrates in [300, 750, 1200, 1850, 2850, 4300] Kbps, which correspond to resolutions [144p, 240p, 360p, 480p, 720p, 1080p] (consistent with [8], [17]). For the video at each bitrate, we encode it into two versions, with a 2-second or 4-second chunk duration respectively. Each version of the video is indexed by a Media Presentation Description (MPD) file generated by MP4Box [27].

Video client: We use Google Chrome browser running dash.js as the video player, and develop Python scripts based on Selenium [28] to automatically control the browser to imitate the requesting and playing behavior of users. The scripts run on Windows laptops, which are connected with APs by WiFi (2.4GHz or 5GHz band), or 4G hotspots by USB.

Video server: We use Nginx server [29] to host video contents and dash.js player, which runs in 3 distinct cloud servers located in 3 cities running Ubuntu 16.04, with varied downstream bandwidth (5Mbps or 50Mbps). We also deploy scripts on the server to interact with the client. Note that the measurement platform is deployed in real-world environment. Thus, we run tests in the wild Internet, instead of in the emulator with constant throughput traces.

Methodology: A test contains several sessions. (1) Before a test starts, we record connection type and signal strength of the client. Since we only aim to conduct tests rather than to implement a complete production system, we simply input the connection type manually into the scripts[2]. As for signal strength, RSSI for WiFi is automatically obtained by pywifi [31] in the scripts, while RSRP and SINR for 4G are manually fetched from Cellular-Z [32][3]. To eliminate the impact of mobility, we conduct tests with the client staying still. (2) To obtain data of chunks at all bitrates in one test, we implement a custom rule in dash.js, which selects a constant bitrate in the whole video session[4]. Before each session starts, the output bitrate in the custom rule will be increased to the next available level by scripts on the server. Then, the client requests for codes of dash.js player from the server, and starts the video session. In this way, all the six bitrates can be traversed in a test with six sessions. (3) During each session, we collect information from both the player on the client and the TCP/IP stack on the server. The client records playback information of each chunk via the custom rule, including chunk size, buffer level, delivery time, inactivity time, rebuffering time and application throughput. The server captures information of TCP connection via tcp_probe in Linux kernel, including $cwnd$ size, $ssthresh$, smoothed RTT and inflight size. We conduct extensive tests for each combination at different time (morning, noon, afternoon or evening), in different locations (4 cities for the client) and with different access network types (4G or WiFi), to cover different usage scenarios as many as possible. In all, we collected data of 2500+ video sessions (800+ with constant bitrate and 1700+ with ABR algorithms), containing 300,000+ video chunks.

B. Factors that Impact Throughput Prediction

To figure out how factors listed in Fig. 1 contribute to throughput prediction of video chunks, we evaluate mutual information [35], [36] between throughput and these factors. Since physical property and background traffic of the link can not be obtained accurately, we use past throughput samples (e.g., throughput of the last chunk) to roughly estimate their effects, which are widely used to predict throughput in ABR algorithms. We use downstream bandwidth of the server to indicate traffic policing (§III-A). As for connection status, prior works focus on features of the whole network, e.g., information of ISP, AS [5] and CDN [10]. However, these information can not directly describe the network environment of the client, but also is hard for the client to obtain, especially under DASH specification. Hence we select respective factors of the client side, i.e., connection type and signal strength [20],

[2]Connection type can be accessed in dash.js through Network Information API [30]. However, this experimental API is not supported by desktop browsers when we build this platform.

[3]In practice, signal strength of WiFi and 4G can be accessed through specific Android APIs, e.g., [33], [34].

[4]We also implement several existing ABR algorithms for training and testing, see §IV-B and §V.

to characterize connection status (§III-A). For comparison, we classify signal strength into three categories (strong, middle and weak), according to RSSI of WiFi and RSRP of 4G. Besides, we set the inactivity time to 200ms to distinguish the two states of the player, which are indicated by the relative index. The relative index of each chunk is increased by chunk in the Buffering-State; otherwise it stays 0.

The mutual information between throughput and each considered factor is shown in Fig. 3. Note that mutual information does not have an upper bound.

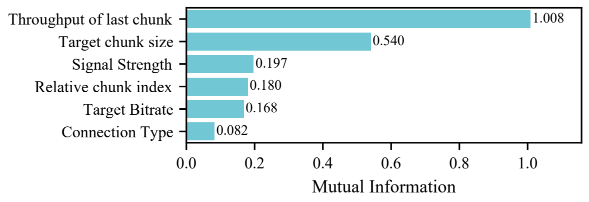


Fig. 3: Mutual information between throughput and its impacting factors

Observation 1: ***Throughput of last chunk and target chunk size are the two most important factors in throughput prediction.***

Previous works which only consider past throughput samples, e.g., [3], fail to achieve accurate throughput prediction due to their ignorance of target chunk size. This result distinguishes application throughput from available bandwidth since the former varies with different chunk size. Although other factors contribute less to predict throughput directly, we argue that integrating them leads to better prediction accuracy, which is investigated in next subsection.

C. Correlation between Throughput and Chunk Size

Prior works have reported that application throughput is related to chunk size [12], [13]. To investigate this relationship, we use maximal information coefficient (MIC) [37] to calculate the correlation between throughput and chunk size, and find this correlation affected by client player's state, bitrate level, and network condition.

Observation 2: ***Correlation between throughput and chunk size is deeply affected by player's state, relative chunk index, and signal strength of the client.***

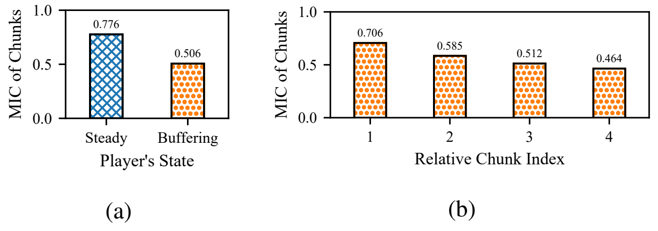


Fig. 4: Correlation of throughput and chunk size varies in different player's states

1) Player's state: To investigate the impact of player's state on the correlation, we only focus on data from the best network conditions, i.e., under strong signal strength with 50Mbps downstream bandwidth, where network capacity is sufficient for delivery with the highest bitrate. Fig. 4a shows that the correlation in the Steady-State is higher than that in the Buffering-State. This is because $cwnd$ in the Steady-State increases from the initial value independently for each chunk (§ II), before congestion is triggered, larger chunk size leads to larger $cwnd$ size, and thus higher throughput.

We further analyze how the correlation is affected by the relative chunk index in the Buffering-State. Fig. 4b shows that correlation decreases as relative chunk index increases. This is caused by subsequent chunks continuously augmenting $cwnd$ as well as the delivery rate, eventually up to the available bandwidth before the Buffering-State ends.

2) Bitrate level: Fig. 5a (only Steady-State under strong network with 50Mbps downstream bandwidth) shows that the correlation becomes lower as the bitrate increases. The reason is similar to the above: chunk size of high bitrates is large enough for $cwnd$ to ramp up to a large size, where the delivery rate of the server reaches the available bandwidth. As a consequence, the incoming congestion events prevent application throughput from increasing with chunk size.

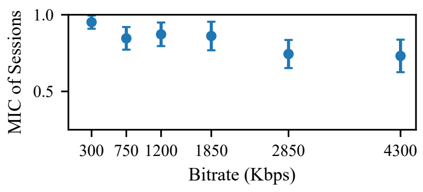
3) Network condition: We conduct tests in different network conditions to investigate how the correlation is affected by traffic policing and connection status.

- *Downstream bandwidth of the server*: We limit the downstream bandwidth of the server to 50Mbps and 5Mbps respectively and show the impact of traffic policing on the correlation in Fig. 5b (only Steady-State under strong signal). We find that the correlation under low downstream bandwidth is much lower than that under high downstream bandwidth, as delivering a chunk under low downstream bandwidth will more easily affected by traffic policing (e.g., token bucket).
- *Wireless signal strength of the client*: Fig. 5c presents the impact of signal strength on the correlation (only Steady-State with 50Mbps bandwidth). When signal strength gets weaker, the correlation becomes smaller correspondingly. To explain this observation, we analyze data collected from the transport layer by tcp_probe, and find that when signal strength is weak, smoothed RTT tends to be longer for the larger chunks (especially of the higher bitrates, not shown), which directly leads to an increase in delivery time, and further a decline in throughput.

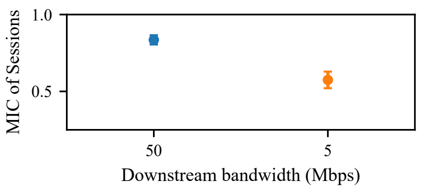
These results all indicate that application throughput is determined by both available bandwidth affected by network conditions and delivery rate affected by application behavior. Previous works conclude that throughput is positively correlated to chunk size, but we argue that this correlation is heavily affected by the player's state, video bitrate and network conditions in real world. Thus, when predicting throughput, taking these factors into consideration will improve the accuracy.

IV. Lumos: Design and Implementation

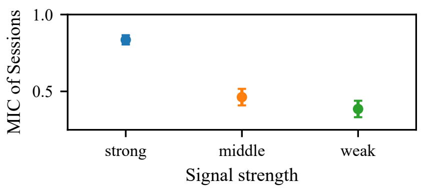
Our observations in §III indicate that by involving specific information, more accurate prediction of application through-



(a) Correlation under different bitrates



(b) Correlation under different bandwidth



(c) Correlation under different signal strength

Fig. 5: Correlation of throughput and chunk size varies under different factors (with 95% confidence)

put can be achieved. Based on that, we propose Lumos, a throughput predictor for video chunks in adaptive streaming. In this section, we introduce the design choice (§IV-A) and details (§IV-B) of Lumos, and show how Lumos is integrated into existing ABR algorithms as a plug-in (§IV-C).

A. Design Choice

As a predictor for ABR algorithms, Lumos aims to build a faithful map between input features and output predictions. To achieve this goal, two questions in designing Lumos arise:

1) Mapping function: which model is more suitable for prediction of throughput in video streaming?

State-of-the-art mapping functions are built by data-driven methods, including three off-the-shelf models: multiple linear regression (MLR), decision trees and Deep Neural Networks (DNNs). Although DNNs are qualified for accurately modeling sophisticated behavior, prior studies show that they are heavy-weight and short of interpretability [9], [38], [39], and thus hard to design and deploy. On the contrary, recent works [39], [40] show the rich expressiveness and high interpretability of decision trees for sophisticated policies, proving that decision trees are simple yet powerful enough to fit complex functions for prediction. For this reason, we develop predictors based on decision trees, and also MLR for comparison.

2) Output target: Which one of throughput and delivery time is a better target to predict?

Both throughput and delivery time can represent how soon the next chunk will be available in the playback buffer. Given the chunk size, each of the two targets can be calculated directly from the other. Most previous works (e.g., FESTIVE [3]) utilize predicted throughput to select bitrate, while recent ones (e.g., Fugu [11]) advocate that delivery time is a better choice.

To investigate these two questions, we develop different predictors based on decision trees and MLR for each prediction target separately, and evaluate their performance on prediction. These predictors are developed mainly following the methodology described in §IV-B[5], only without the two improvements of Lumos.

Prediction Metrics: For both of throughput and delivery time, the more accurately they can be predicted, the more precisely ABR algorithm could select bitrate for better QoE [5], [11], [41]. Thus, we use prediction accuracy to evaluate the performance of the two targets. Following [5], [10], we select Absolute Normalized Prediction Error Err (prediction error for short) as the metric, defined as Eq. 1.

$$Err(DTime) = \frac{1}{N}\sum_{k=1}^{N}\frac{|\hat{DT}_k - DT_k|}{DT_k}\ , \qquad (1)$$

where $\hat{DT}_k$ and DT_k denote the predicted value and the real value of delivery time of chunk k respectively, and N denotes the number of chunks in a session. *The predicted throughput is converted to delivery time for comparison with the directly predicted delivery time.*

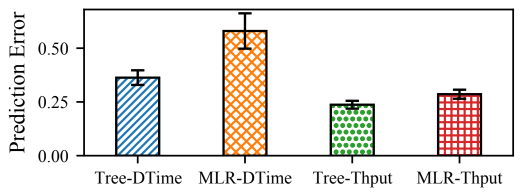


Fig. 6: Prediction error of the four considered predictors

Fig. 6 shows that decision trees (Tree for short) perform better than MLR does in prediction. More importantly, for both the two types of models, compared with delivery time prediction, throughput prediction has lower prediction error (34.7%∼50.7% reduction). Thus, we have the following observation.

Observation 3: ***Throughput prediction achieves better accuracy than delivery time prediction does.***

The first question is *why prediction error of throughput predictors is lower*. Prediction error is a metric to measure how much the target is overestimated or underestimated. For both delivery time and throughput, the prediction error of underestimation is no more than 100%, while that of overestimation could exceed 100%. Thus, overestimation is the key factor in increasing prediction error. Fig. 7 shows the distribution of prediction error of each video chunk (not session). Delivery time predictors perceive prediction error of over 100% for more chunks (6.6%∼12.4%) than throughput predictors (1.7%∼3.0%). This result indicates that delivery time predictors overestimate delivery time more seriously than throughput predictors do, leading to higher prediction error.

The second question is *why delivery time predictors tend to overestimate*. We infer that the root cause is the long-tailed distribution of delivery time. As shown in Fig. 8, delivery time is more long-tailed, with 0.8% of chunks lying in the 95% tail of time interval (from 10s to 208s). This phenomenon is known

[5]For delivery time predictors, the feature *throughput of the last chunk* is replaced by delivery *time of the last chunk*.

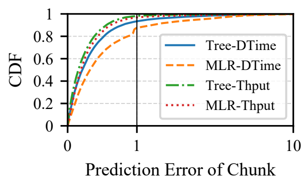


Fig. 7: CDF of prediction error of video chunks by various predictors

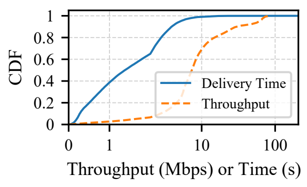


Fig. 8: CDF of delivery time and throughput of video chunks

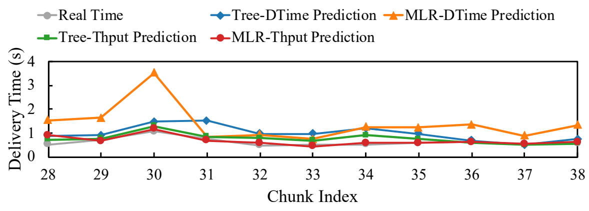


Fig. 9: A case of predicting delivery time by various predictors

as a character of the wild Internet, which is also mentioned in prior work [11]. For both regression decision trees and MLR, the models are determined by distribution of data. Delivery time predictors learn higher values due to the long tail of distribution of delivery time, and thus overestimate the real values, as shown in Fig. 9. Contrastingly, the distribution of throughput is more ”uniform”. As a result, throughput predictors are able to faithfully learn the characters of most data, avoiding being affected by outliers. Although we may improve delivery time predictors by filtering outliers offline, our goal is to design a faithful predictor performs well in real world, where the distribution of all data is unknown in advance. In this way, predicting throughput is a better choice.

B. *Lumos Mechanism*

In this subsection, we describe how to design, train and implement Lumos.

1) Design: Training decision trees is a supervised learning process. Since the values of throughput are continuous, we use regression trees. We select the input features for models following the observations in §III. Since information of some features can not be directly accessed by the client (e.g., downstream bandwidth of the server), we use other forms of features to approximate them. These features are divided into three categories as follows.

- Network condition: including (1) *maximum of throughput of past t chunks* to estimate limited bandwidth [14], (2) *maximum of delivery time of past t chunks*, (3) *connection type* (WiFi or 4G) of the client, and (4) *throughput of the last chunk*;
- Player's state: including (5) *relative index of the last chunk*, which indicates player's state by 0 for the Steady-State and others for the Buffering-State;
- Chunk information: including (6) *bitrate* and (7) *size of last chunk*, and (8) *bitrate* and (9) *size of the target chunk*.

We find that in real-world environment, it is hard for a single decision tree to fit throughput of all chunks due to its wide range of distribution (2.0Kbps∼84.7Mbps in our dataset). For this reason, we develop Lumos with two extra improvements as follows.

- Separate predictor for each network environment: As shown in Fig. 10, we divide network environment into three categories according to signal strength of the client (§III-B), and train a specific model for each type of network respectively. Note that we are not saying this classification is the best choice, but handling with distinct network conditions separately helps the model achieve better performance [8].
- Logarithm value of throughput as labels: Since the throughput values lie in a wide range across 5 orders of magnitude, we use the logarithm value of throughput as label instead of original value.

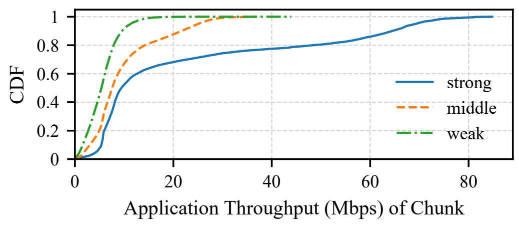


Fig. 10: Throughput under different signal strength

2) Training and Implementation: We train decision tress by Classification and Regression Tree (CART) [42]. CART uses mean squared error (MSE) as loss function for regression tress, and generates trees by minimizing MSE between predicted values and real values. We apply both prepruning and cost complexity pruning [42] to prune trees from overfitting. To identify optimal pruning parameters, we adopt exhaustive grid search and K-fold cross validation to select best model. We use sklearn [43] to implement decision trees of Lumos, and dump parameters of trained models in JavaScript so that Lumos models can be loaded in dash.js. Video dataset is same to that deployed on our measurement platform §III-A. We select data from 700+ video sessions (running ABR algorithms, not selecting constant bitrate) collected in real world, which contain 69,000+ chunks, to train and test Lumos. 70% of all sessions are used as training set. Since imbalanced dataset harms the performance of decision trees [39], we balance the data of various combinations of network conditions (downstream bandwidth and connection type); i.e., for each class of signal strength, the count of chunks under 50Mbps-4G, 5Mbps-4G, 50Mbps-WiFi and 5Mbps-WiFi is roughly equivalent.

C. *Lumos as A Plug-in of ABR algorithms*

As a throughput predictor of video chunks in adaptive streaming, Lumos can be integrated into any ABR algorithm which uses throughput prediction. We use three classic ABR algorithms as examples to illustrate how Lumos is applied to existing schemes.

(1) Rate-Based (RB) [3], which selects the highest bitrate below the predicted throughput by harmonic mean (HM) of

TABLE I: Variables and their meanings in ABR

Category	Variable	Meaning
Common	R_k	bitrate of the k-th chunk
	m	the number of bitrates each video slice has
	$R_{k\|r_j}$	select bitrate r_j for the k-th chunk
	T_k	throughput of the k-th chunk
	$\hat{T}_{k\|r_j}$	prediction value of T_k at bitrate r_j
	$d_k(r_j)$	size of the k-th chunk at bitrate r_j
	L	duration of a video chunk
	B_k	buffer occupancy when requesting the k-th chunk
MPC	μ	coefficient of rebuffering time in QoE of MPC
	λ	coefficient of smoothness in QoE of MPC
	n	number of chunks to be predicted
BBA	B_{lower}	the lower bound of playing buffer, also named reservoir

throughput samples for past chunks. We integrate Lumos with RB by replacing the HM predictor with Lumos. As the bitrate of video chunks (chunk size divided by chunk duration) encoded with H.264/MPEG-4 fluctuate widely around the average bitrate [8], RB+Lumos selects the chunk according to its real bitrate instead of the average bitrate to make more reasonable decisions, as shown in Eq. 2. Denotations of variables are listed in Tab. I.

$$R_k = \max_{1 \le j \le m} \{r_j, d_k(r_j)/\hat{T}_{k|r_j} \le L\} \tag{2}$$

(2) MPC [4], which uses both buffer level and predicted throughput (by HM predictor, the same as RB) to select the bitrate which maximizes estimated QoE of a series of subsequent chunks. We design MPC+Lumos by replacing the HM predictor in MPC with Lumos to predict the throughput for each of future chunks, shown in Eq. 3[6]. As Lumos requires past throughput as features, for the future chunks except the first one, Lumos takes the predicted throughput of the prior chunk as input.

$$R_k = \arg\max_{r_j, 1 \le j \le m} \sum_{l=k}^{k+n-1} Q\hat{o}E(R_{l|r_j}) \tag{3}$$

$$\begin{aligned} Q\hat{o}E(R_{l|r_j}) = R_{l|r_j} - \max(\mu(\frac{d_l(R_{l|r_j})}{\hat{T}_{l|r_j}} - B_l), 0) \\ - \lambda |R_{l|r_j} - R_{l-1}|, k \le l \le k+n-1 \end{aligned} \tag{4}$$

(3) Buffer-Based Approach (BBA) [16], which builds a linear mapping function between the level of playing buffer and target bitrate, and sets both lower and upper bounds of the buffer to select bitrate. When buffer level is below the lower bound or above the upper bound, BBA selects lowest bitrate or highest bitrate, respectively. When the buffer level is between the lower bound and the upper bound, it selects the bitrate simply according to that how much current buffer level exceeds the lower bound. Although BBA makes decisions without prediction, we argue that accurate prediction assists BBA to achieve better QoE. We integrate Lumos with BBA by replacing linear mapping function with the prediction result of Lumos. BBA+Lumos retains the lower bound and redesigns the bitrate selection function by predicted delivery time (chunk size divided by the predicted throughput). For each bitrate of next chunk, BBA+Lumos sets a threshold for it, which is the lower bound plus the difference between predicted delivery time of it and that of the lowest bitrate. BBA+Lumos selects the bitrate only when the buffer level exceeds the corresponding threshold, as illustrated in Eq. 5.

[6]In Eq. 4, $|R_k - R_{k-1}|$ is equal to 0 when $k = 1$.

$$R_k = \max_{1 \le j \le m} \{r_j, B_{lower} + \frac{d_k(r_j)}{\hat{T}_{k|r_j}} - \frac{d_k(r_1)}{\hat{T}_{k|r_1}} \le B_k\} \tag{5}$$

V. Evaluation

We implement the three ABR algorithms above (§IV-C) with their Lumos-assisted versions, as well as other two schemes (RobustMPC [4] and Pensieve [17]) in dash.js, and deploy them on our measurement platform (§III-A). We evaluate all the above algorithms in real-world Internet. Extensive sets of tests with about 300 sessions are carried out to cover various network environments, including 3 variables: downstream bandwidth of the server (50 Mbps or 5Mbps), connection types (WiFi or 4G) and signal strength (strong, middle and weak) of the mobile client. Sessions are balanced under different network conditions, as in training Lumos (§IV-B).

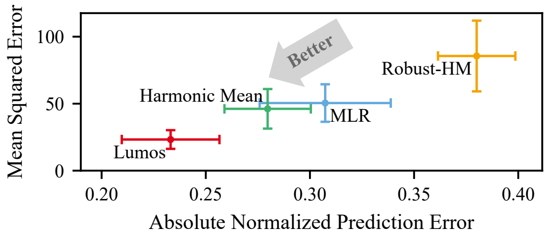


Fig. 11: Prediction performance of various methods

A. The Prediction Accuracy of Lumos

Firstly, we evaluate the accuracy that Lumos predicts the throughput of video chunks. We choose three widely used methods as baseline for comparison: MLR (multiple linear regression, trained in the same way as Lumos), HM (harmonic mean of past 5 samples) and Robust-HM (harmonic mean of past 5 samples with error rate normalization, used in RobustMPC [4]). We use both prediction error and mean squared error (MSE) to evaluate the accuracy for predicting throughput.

Fig. 11 shows the prediction accuracy of various methods. We can see that in terms of both metrics, Lumos achieves best performance for throughput prediction. Specifically, Lumos reduces two types of error by 16.8%∼38.7% and 49.6%∼72.8% compared to other methods. It is notable that under strong network connected with WiFi, Lumos achieves a remarkably low prediction error with around only 7.4% in average, and improves 57.7%$\sim$74.0% and 78.5%$\sim$90.5% in terms of two types of error than other methods. Even under weak network, Lumos still improves accuracy by 9.1%$\sim$28.9% and 17.8%$\sim$61.7% over the two metrics. These all demonstrate that Lumos learns well about the correlation between throughput and the considered factors.

Compared with HM which is widely used in existing ABRs, Lumos achieves much better prediction accuracy. The advantage of Lumos over HM are as follows.

- *Awareness of network conditions and player's state.* When there is no enough past data, e.g., in the start-up phase of a session, it is hard for predictors to make accurate prediction [5]. For strong network connected with WiFi (Fig. 12a as an example), due to lack of past data, HM obtains 95% prediction error in average for the first 5 chunks (5 is the number of chunks HM needs for prediction) of sessions. As for Lumos, by involving network conditions and player's state, it reduces the average prediction error to only 16% for the first 5 chunks in the same situation.
- *Consideration of the fluctuation of chunk sizes.* HM assumes that throughput fluctuation is determined by variation of available bandwidth. However, our analysis and observations indicate that when the available bandwidth is adequate, the perceived throughput is strongly affected by the chunk size. Fig. 12b gives a case in the Steady-State under strong network, and shows that Lumos clearly knows that how chunk size fluctuation affect throughput.
- *Quick reaction to bandwidth fluctuation.* Under weak network, we find that throughput is always unstable with frequent fluctuation, which makes it almost unpredictable. In such scenario, Lumos exceeds HM by quick reaction to fluctuation, as illustrated in Fig. 12c. From the 69th chunk, throughput suddenly decreased from 6.7Mbps to 3.9Mbps, and remained declining for the next two chunks. Since HM is insensitive to outliers [3], it is supposed to lower the predicted values, but increased them instead for continuous 3 chunks, which may introduce rebuffering. Contrastingly, Lumos instantly perceives the dynamics of throughput, and makes more accurate prediction accordingly.

B. The QoE of Lumos-assisted MPC

We now present how accurate throughput prediction contributes to the improvement of QoE for MPC. Based on throughput prediction, MPC models the evolution of QoE of future chunks as precisely as possible. In theory, the more accurately throughput is predicted, the better performance MPC will achieve [4]. We replace the HM in MPC by Lumos and have a new ABR algorithm, MPC+Lumos.

QoE metrics. To evaluate the QoE of the three ABR algorithms, we use the metrics in MPC [4], defined as:

$$\begin{aligned} QoE = \sum_{k=1}^{N} R_k - \mu \sum_{k=1}^{N} \max\left(\left(\frac{d_k(R_k)}{T_k} - B_k\right), 0\right) \\ - \lambda \sum_{k=1}^{N-1} |R_{k+1} - R_k| \ , \end{aligned} \tag{6}$$

where N is the number of chunks in the session, R_k is bitrate that client selects for chunk k, the second item represents the rebuffering time during delivering chunk k, and the last item represents smoothness on bitrate switch between chunks. In the equation, μ and λ are non-negative weighting parameters, which are respectively set to 4.3 and 1.0, following recent works [17], [24], [38].

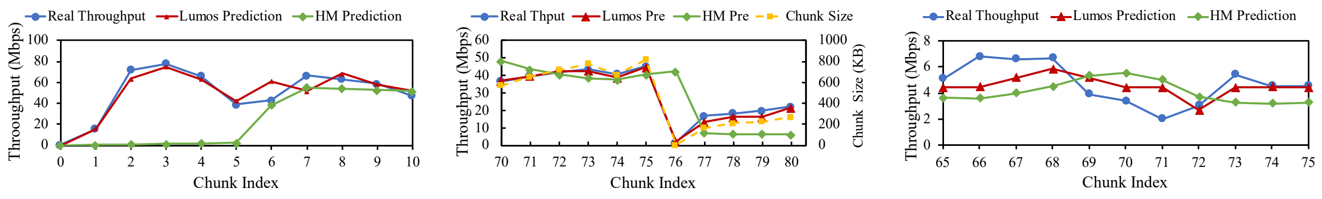
QoE performance. Fig. 13 shows the QoE of MPC, RobustMPC and MPC+Lumos. Combining the results in Fig. 13a and 13b, MPC+Lumos always selects higher bitrate with acceptable rebuffering time than the other two schemes. Overall, Lumos improves average QoE by 6.3% over MPC and 8.7% over RobustMPC. We find that although Lumos provides much more better accuracy than Robust-HM does (§V-A), rebuffering time of MPC+Lumos is more than that of RobustMPC. This is because RobustMPC is designed to avoid rebuffering by predicting lower throughput, which however sacrifices video bitrate (9% lower than MPC+Lumos) as well as total QoE. Moreover, compared with MPC, MPC+Lumos makes more aggressive decisions (3.4% increase in bitrate), while having 52% reduction in rebuffering time. These results confirm the benefits of the accurate prediction by Lumos.

C. MPC+Lumos vs. Pensieve

Pensieve [17] is one of state-of-the-art learning-based ABR algorithms, which utilizes deep reinforcement learning to learn the policy optimizing the overall QoE. To compare Lumos-assisted ABR algorithms with Pensieve, we select MPC+Lumos as it achieves better QoE than others. As the performance of Pensieve relies heavily on the similarity between the environment where it is trained and that where it is tested, we use the source code provided by the authors [44] and retrain Pensieve with our video dataset and network traces (which are also used to train Lumos) collected in the wild, following [11], [24]. We convert Pensieve's model into JavaScript, and deploy the model in dash.js by TensorFlow.js [45].

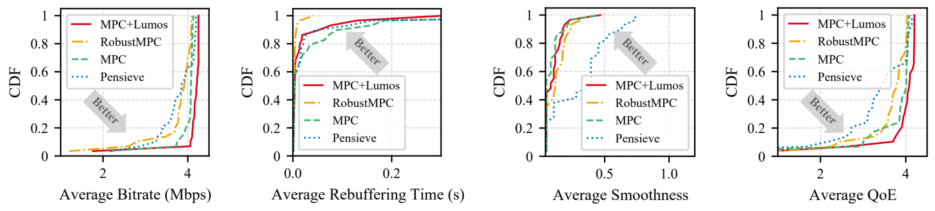
Fig. 13 shows the QoE metrics of MPC+Lumos and Pensieve. Compared with Pensieve, MPC+Lumos achieves remarkable improvement in QoE, which outperforms Pensieve on 93% of all the sessions, with an increase on QoE of 19.2% in average. As for underlying QoE metrics, MPC+Lumos improves the average bitrate by 8.9%, and reduces the average rebuffering time and smoothness by 44.2% and 66.1% respectively. The reasons that MPC+Lumos outperforms Pensieve in real-world network are summarized as follows.

- *Limited ability of generalization.* Throughput of chunks varies greatly in real world, with the maximum and minimum values differing by 5 orders of magnitude in our dataset (§IV-B). It poses a huge challenge for Pensieve model to predict accurately. Although we have already retrained Pensieve, Pensieve still appears to be not powerful enough to specialize to all scenarios as reported in [24].
- *Difference between simulator and real world.* The simulator used to train Pensieve model is designed with the precondition of disabling slow-start restart [17]. While in the real world, most Linux servers enable slow-start restart



(a) Lumos performs well in start-up phase (b) Lumos takes chunk size into consideration (c) Lumos reacts quickly to fluctuation

Fig. 12: Cases of Lumos outperforming HM in throughput prediction



(a) (b) (c) (d)

Fig. 13: QoE Performance of MPC+Lumos vs. MPC, RobustMPC and Pensieve

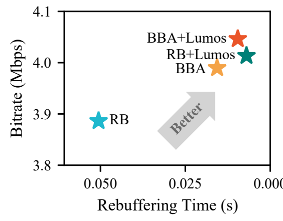


Fig. 14: QoE Improvement of RB+Lumos and BBA+Lumos

by default, which Pensieve is blind to. Moreover, Pensieve's simulator works with constant network traces, in which the bitrate selection has no impact on the throughput of next chunk, which is far from the real-world cases according to our observations in §III.

D. *Lumos with Other ABR algorithms*

Finally, we evaluate the QoE performance of RB+Lumos and BBA+Lumos, both of which are run in real-world network. As RB and BBA are not designed to consider smoothness, we only focus on the average bitrate and rebuffering time of RB+Lumos and BBA+Lumos. The results of the four considered schemes are given in Fig. 14. It turns out that with Lumos assisted, both RB and BBA can obtain better QoE.

With accurate prediction by Lumos, RB+Lumos improves bitrate by 3.3% and reduces rebuffering time by 86.3%, compared with RB. Besides, although BBA is quite effective in the real world [11], BBA+Lumos still outperforms BBA on both bitrate (1.4% increase) and rebuffering time (37.5% reduction), demonstrating the advantages brought by integrating accurate prediction of throughput in ABR algorithms.

VI. Related Work

There is a large amount of existing works on ABR algorithms for video steaming [3], [4], [6]–[9], [11], [16]–[18], [24], [38], [46]. Most of them rely on predicting throughput or delivery time explicitly or implicitly, while only a few studies focus on this issue [5], [10]–[13].

Among them, [12] and [13] indicate that it is difficult to predict throughput as background traffic exists, but they do not investigate how to achieve better prediction. Both CS2P [5] and MPC-CDN [10] propose predictors based on observations in real world. However, they only focus on factors about connection status, and believe that throughput fluctuation is all brought by change of network conditions. Fugu [11] is the first work to consider chunk size in predicting delivery time, which also utilizes statistic information of the transport layer as the indicator of network conditions. Nevertheless, chunk size only represents the amount of delivered data of the application, how the application behaves during delivery also matters.

Different from the above, our work considers both application behavior and network conditions in prediction, brings insight that predicting throughput is better than predicting delivery time in terms of prediction error, and proposes a throughput predictor which could be integrated into ABR algorithms and performs well in the wild Internet.

VII. Conclusion

In this work, we built a real-world measurement platform for video streaming and collected dataset containing 2500+ sessions in the wild Internet. We conceptually distinguished application throughput and available bandwidth, identified the important factors that affect perceived throughput, and found that throughput is a better prediction target than delivery time for ABR algorithms. Further, we developed a throughput predictor for ABR algorithms named Lumos, which assists existing ABR algorithms to achieve better QoE, by offering accurate predictions.

Acknowledgment

This work was partially supported by National Key R&D Program of China (No. 2019YFB1802800), National Natural Science Foundation of China (No.61725206, 62072437) and Beijing Natural Science Foundation (No. JQ20024). Corresponding Author: Gaogang Xie.

References

[1] DASH, "Dash industry forum — catalyzing the adoption of mpeg-dash," https://dashif.org/, 2021.

[2] V. Cisco, "Cisco visual networking index: Forecast and trends, 2017–2022," *White Paper*, vol. 1, p. 1, 2018.

[3] J. Jiang, V. Sekar, and H. Zhang, "Improving fairness, efficiency, and stability in http-based adaptive video streaming with festive," in *Proceedings of the 8th international conference on Emerging networking experiments and technologies*, 2012, pp. 97–108.

[4] X. Yin, A. Jindal, V. Sekar, and B. Sinopoli, "A control-theoretic approach for dynamic adaptive video streaming over http," in *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, 2015, pp. 325–338.

[5] Y. Sun, X. Yin, J. Jiang, V. Sekar, F. Lin, N. Wang, T. Liu, and B. Sinopoli, "Cs2p: Improving video bitrate selection and adaptation with data-driven throughput prediction," in *Proceedings of the 2016 ACM SIGCOMM Conference*, 2016, pp. 272–285.

[6] Y. Qin, R. Jin, S. Hao, K. R. Pattipati, F. Qian, S. Sen, B. Wang, and C. Yue, "A control theoretic approach to abr video streaming: A fresh look at pid-based rate adaptation," in *IEEE INFOCOM 2017-IEEE Conference on Computer Communications*. IEEE, 2017, pp. 1–9.

[7] K. Spiteri, R. Sitaraman, and D. Sparacio, "From theory to practice: improving bitrate adaptation in the dash reference player," in *Proceedings of the 9th ACM Multimedia Systems Conference*, 2018, pp. 123–137.

[8] Y. Qin, S. Hao, K. R. Pattipati, F. Qian, S. Sen, B. Wang, and C. Yue, "Abr streaming of vbr-encoded videos: characterization, challenges, and solutions," in *Proceedings of the 14th International Conference on emerging Networking EXperiments and Technologies*, 2018, pp. 366–378.

[9] H. Mao, S. Chen, D. Dimmery, S. Singh, D. Blaisdell, Y. Tian, M. Alizadeh, and E. Bakshy, "Real-world video adaptation with reinforcement learning," in *ICML 2019 Workshop RL4RealLife*, 2019.

[10] E. Ghabashneh and S. Rao, "Exploring the interplay between cdn caching and video streaming performance," in *IEEE INFOCOM 2020-IEEE Conference on Computer Communications*. IEEE, 2020, pp. 516–525.

[11] F. Y. Yan, H. Ayers, C. Zhu, S. Fouladi, J. Hong, K. Zhang, P. Levis, and K. Winstein, "Learning in situ: a randomized experiment in video streaming," in *17th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 20)*, 2020, pp. 495–511.

[12] S. Akhshabi, L. Anantakrishnan, A. C. Begen, and C. Dovrolis, "What happens when http adaptive streaming players compete for bandwidth?" in *Proceedings of the 22nd international workshop on Network and Operating System Support for Digital Audio and Video*, 2012, pp. 9–14.

[13] T.-Y. Huang, N. Handigol, B. Heller, N. McKeown, and R. Johari, "Confused, timid, and unstable: picking a video streaming rate is hard," in *Proceedings of the 2012 internet measurement conference*, 2012, pp. 225–238.

[14] N. Cardwell, Y. Cheng, C. S. Gunn, S. H. Yeganeh, and V. Jacobson, "Bbr: Congestion-based congestion control," *Queue*, vol. 14, no. 5, pp. 20–53, 2016.

[15] B. Wang and F. Ren, "Towards forward-looking online bitrate adaptation for dash," in *Proceedings of the 25th ACM international conference on Multimedia*, 2017, pp. 1122–1129.

[16] T.-Y. Huang, R. Johari, N. McKeown, M. Trunnell, and M. Watson, "A buffer-based approach to rate adaptation: Evidence from a large video streaming service," in *Proceedings of the 2014 ACM conference on SIGCOMM*, 2014, pp. 187–198.

[17] H. Mao, R. Netravali, and M. Alizadeh, "Neural adaptive video streaming with pensieve," in *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, 2017, pp. 197–210.

[18] K. Spiteri, R. Urgaonkar, and R. K. Sitaraman, "Bola: Near-optimal bitrate adaptation for online videos," in *IEEE INFOCOM 2016-The 35th Annual IEEE International Conference on Computer Communications*. IEEE, 2016, pp. 1–9.

[19] T. Flach, P. Papageorge, A. Terzis, L. Pedrosa, Y. Cheng, T. Karim, E. Katz-Bassett, and R. Govindan, "An internet-wide analysis of traffic policing," in *Proceedings of the 2016 ACM SIGCOMM Conference*, 2016, pp. 468–482.

[20] A. Narayanan, E. Ramadan, R. Mehta, X. Hu, Q. Liu, R. A. Fezeu, U. K. Dayalan, S. Verma, P. Ji, T. Li *et al.*, "Lumos5g: Mapping and predicting commercial mmwave 5g throughput," in *Proceedings of the ACM Internet Measurement Conference*, 2020, pp. 176–193.

[21] M. Allman, V. Paxson, and W. Stevens, "Tcp congestion control," *RFC*, vol. 2581, pp. 1–14, 1999.

[22] X. K. Zou, J. Erman, V. Gopalakrishnan, E. Halepovic, R. Jana, X. Jin, J. Rexford, and R. K. Sinha, "Can accurate predictions improve video streaming in cellular networks?" in *Proceedings of the 16th International Workshop on Mobile Computing Systems and Applications*, 2015, pp. 57–62.

[23] M. Licciardello, M. Grüner, and A. Singla, "Understanding video streaming algorithms in the wild," in *International Conference on Passive and Active Network Measurement*. Springer, 2020, pp. 298–313.

[24] Z. Akhtar, Y. S. Nam, R. Govindan, S. Rao, J. Chen, E. Katz-Bassett, B. Ribeiro, J. Zhan, and H. Zhang, "Oboe: auto-tuning video abr algorithms to network conditions," in *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, 2018, pp. 44–58.

[25] "Xiph.org: Derf's test media collection," https://media.xiph.org/video/derf, 2021.

[26] FFmpeg, https://www.ffmpeg.org/, 2021.

[27] J. Le Feuvre, C. Concolato, and J.-C. Moissinac, "Gpac: open source multimedia framework," in *Proceedings of the 15th ACM international conference on Multimedia*, 2007, pp. 1009–1012.

[28] Selenium, https://www.selenium.dev/, 2021.

[29] Nginx, http://nginx.org/, 2021.

[30] MDN, "Network information api - web apis," https://developer.mozilla.org/en-US/docs/Web/API/Network_Information_API, 2021.

[31] J. Sheng-Jhih, "pywifi," https://pypi.org/project/pywifi/, 2021.

[32] JerseyHo, "Cellular-z," https://play.google.com/store/apps/details?id=make.more.r2d2.cellular_z, 2021.

[33] Android, https://developer.android.com/reference/android/net/wifi/WifiInfo#getRssi(), 2021.

[34] ——, https://developer.android.com/reference/android/telephony/CellSignalStrengthLte#getRsrp(), 2021.

[35] A. Kraskov, H. Stögbauer, and P. Grassberger, "Estimating mutual information." *Physical review. E, Statistical, nonlinear, and soft matter physics*, vol. 69 6 Pt 2, p. 066138, 2004.

[36] B. C. Ross, "Mutual information between discrete and continuous data sets," *PLoS ONE*, vol. 9, 2014.

[37] D. N. Reshef, Y. A. Reshef, H. K. Finucane, S. R. Grossman, G. McVean, P. J. Turnbaugh, E. S. Lander, M. Mitzenmacher, and P. C. Sabeti, "Detecting novel associations in large data sets," *science*, vol. 334, no. 6062, pp. 1518–1524, 2011.

[38] T. Huang, C. Zhou, R.-X. Zhang, C. Wu, X. Yao, and L. Sun, "Stick: A harmonious fusion of buffer-based and learning-based approach for adaptive streaming," in *IEEE INFOCOM 2020-IEEE Conference on Computer Communications*. IEEE, 2020, pp. 1967–1976.

[39] Z. Meng, M. Wang, J. Bai, M. Xu, H. Mao, and H. Hu, "Interpreting deep learning-based networking systems," *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication*, 2020.

[40] Z. Meng, J. Chen, Y. Guo, C. Sun, H. Hu, and M. Xu, "Pitree: Practical implementation of abr algorithms using decision trees," *Proceedings of the 27th ACM International Conference on Multimedia*, 2019.

[41] A. Narayanan, X. Zhang, R. Zhu, A. Hassan, S. Jin, X. Zhu, X. Zhang, D. Rybkin, Z. Yang, Z. M. Mao *et al.*, "A variegated look at 5g in the wild: performance, power, and qoe implications," in *Proceedings of the 2021 ACM SIGCOMM 2021 Conference*, 2021, pp. 610–625.

[42] L. Breiman, J. Friedman, C. J. Stone, and R. A. Olshen, *Classification and regression trees*. CRC press, 1984.

[43] 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, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[44] Pensieve, https://github.com/hongzimao/pensieve, 2021.

[45] TensorFlow.js, https://tensorflow.google.org/js/, 2021.

[46] T. Zhang, F. Ren, W. Cheng, X. Luo, R. Shu, and X. Liu, "Modeling and analyzing the influence of chunk size variation on bitrate adaptation in dash," in *IEEE INFOCOM 2017-IEEE Conference on Computer Communications*. IEEE, 2017, pp. 1–9.