Accurate Throughput Prediction for Improving QoE in Mobile Adaptive Streaming

Gerui Lv, Qinghua Wu, Qingyue Tan, Weiran Wang, Zhenyu Li, *Member, IEEE*, and Gaogang Xie

***Abstract*—Video streaming is the most important mobile application today. To improve users' quality of experience (QoE), the client player runs adaptive bitrate (ABR) algorithms that dynamically select the bitrate for video chunks based on throughput or delivery time predictions. This paper aims to design an accurate predictor for mobile adaptive streaming by investigating all its components, including input features, output target, and mapping function. We construct the first theoretical framework that reveals potential factors affecting chunk throughput and delivery time. To verify this framework, we provide formulation analysis and measurement observations based on 2500+ video sessions collected in real-world mobile networks. We find that previous works have failed to achieve accurate prediction due to overlooking the impact of the transport mechanism and application behavior on throughput. Furthermore, we show that throughput is a better target for data-driven predictors than delivery time, due to the long-tailed distribution of delivery time. Based on the above, we propose Lumos, a decision-tree-based throughput predictor that can be integrated into various ABR algorithms. Extensive experiments in real-world mobile Internet show that Lumos achieves high prediction accuracy and improves the QoE of MPC by 6.3%, and MPC+Lumos outperforms Pensieve by 19.2%.**

***Index Terms*—Adaptive streaming, mobile internet, QoE improvement, throughput prediction.**

I. Introduction

ADVANCED wireless technology, such as 5G and WiFi7, enables high-speed delivery for mobile devices. As a result, mobile video streaming currently accounts for the majority of the Internet traffic and has become the primary application of over-the-top (OTT) services [2]. HTTP-based video streaming (standardized as DASH [3]) is wildly deployed in commercial video services such as Netflix [4], YouTube [5], and Hulu [6]. In DASH systems, each video is encoded into multiple versions with the same content but different average bitrates (i.e., quality). Each version is further segmented into chunks with equal duration, usually 2–10 seconds. The client video player runs adaptive bitrate (ABR) algorithms to select the bitrate of each chunk based on network capacity, in order to maximize the quality of experience (QoE), including maximizing video quality and minimizing rebuffering time and quality switches.

Most ABR algorithms use throughput prediction to estimate network capacity [7], [8], [9], [10], [11], [12], [13], [14], [15]. As an alternative, recent works [16], [17] advocate to predict the delivery time of chunks for better QoE. Despite all these efforts, predicting network capacity remains challenging in the mobile Internet due to the high dynamic of bandwidth caused by wireless channel interference, weak signal strength, etc.

In this work, we aim to design an accurate predictor for adaptive streaming to optimize QoE in real-world mobile Internet. Naturally, two key problems related to prediction in ABR algorithms emerge:

i) Input features: What factors assist in achieving better prediction? Taking throughput as an example, the application's perceived throughput is affected by both network condition and application behavior [18], [19], [20]. Traditional throughput predictors for video streaming, whether history-based [7], [15], [21] or learning-based [10], consider throughput fluctuation mainly as a change of network condition. Recent works [13], [16], [17] start considering chunk size in prediction. However, chunk size cannot represent all application behaviors, such as ON-OFF period [18], [19]. Furthermore, most studies explore the design space from single disconnected points, and few attempt to take a holistic view of this issue.

ii) 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 the chunk size given, they are considered interchangeable in representing network capacity. However, while throughput corresponds to bitrate selection, delivery time is directly used to calculate QoE ((22)). Therefore, it is intuitive to regard delivery time as a more effective indicator for ABR algorithms. Nevertheless, there is still a lack of quantitative comparisons between these two indicators.

This paper tackles the above problems and achieves the following contributions:

- We construct a theoretical framework that incorporates potential factors impacting throughput and delivery time prediction for mobile video streaming (Section III-A).

Manuscript received 18 April 2023; revised 20 July 2023; accepted 31 August 2023. Date of publication 11 September 2023; date of current version 4 April 2024. This work was supported in part by the National Key R&D Program of China under Grant 2022YFB2901800, in part by Natural Science Foundation of China under Grants U20A20180 and 62072437, and in part by Beijing Natural Science Foundation under Grant JQ20024. Recommended for acceptance by G. Xylomenos. *(Gerui Lv and Qinghua Wu are co-first authors.) (Corresponding author: Gaogang Xie.)*

Gerui Lv, Qingyue Tan, and Weiran Wang are with the Institute of Computing Technology, Chinese Academy of Sciences and University of Chinese Academy of Sciences, Beijing 101408, China (e-mail: lvgerui@ict.ac.cn; tanqingyue22s@ict.ac.cn; wangweiran@ict.ac.cn).

Qinghua Wu and Zhenyu Li are with the Institute of Computing Technology, Chinese Academy of Sciences and University of Chinese Academy of Sciences, Beijing 101408, China, and also with the Purple Mountain Laboratories, Nanjing 211100, China (e-mail: wuqinghua@ict.ac.cn; zyli@ict.ac.cn).

Gaogang Xie is with the Computer Network Information Center, Chinese Academy of Sciences and University of Chinese Academy of Sciences, Beijing 101408, China (e-mail: xie@cnic.cn).

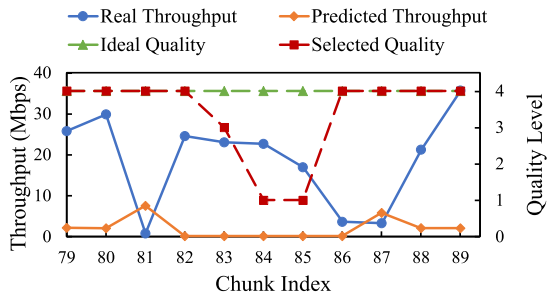
Digital Object Identifier 10.1109/TMC.2023.3313592

Additionally, we perform a formulation analysis to validate this framework (Section III-B). Our framework differentiates between application throughput and available bandwidth and shows how the transport mechanism and application behavior affect perceived throughput and delivery time. To the best of our knowledge, this is the first theoretical framework for network prediction in mobile adaptive streaming.

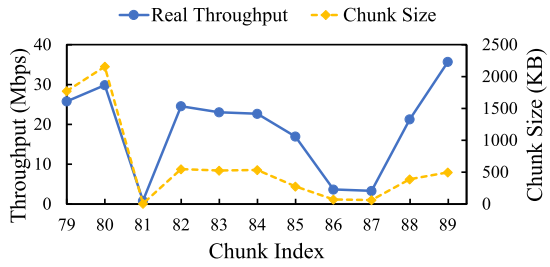
- We build a mobile video streaming measurement platform and collect an extensive dataset[1] containing 2500+ sessions in real-world mobile networks. Through data analysis, we quantitatively verify our proposed theoretical framework and make the following observations (Section III-C). *(i)* There is a strong correlation between chunk size and throughput, which has been overlooked by most previous works, leading to inaccurate throughput prediction. *(ii)* This correlation is deeply affected by the state of the client player, the relative chunk index, and the signal strength of the mobile devices.
- We find that throughput is a better prediction target than the delivery time for data-driven methods (Section IV-A). Results from controlled experiments indicate that *(iii)* predictors for the delivery time have more significant prediction errors due to the long tail distribution of delivery time in the mobile Internet, which can be fitted by a power law.
- Based on the above observations, we propose Lumos, a decision-tree-based throughput predictor for mobile adaptive streaming (Section IV-B). Lumos can be integrated into existing ABR algorithms as a plug-in to improve the prediction accuracy and assist in improving QoE (Section IV-C).
- We evaluate three Lumos-assisted ABR algorithms (RB, MPC [8], and BBA [22]) in real-world mobile networks (Section V). Experimental results indicate that Lumos achieves much better prediction accuracy, and Lumos-assisted ABR algorithms outperform the original algorithms in terms of QoE. In particular, MPC+Lumos improves average QoE by 6.3% over original MPC and even 19.2% over Pensieve [23], which is a state-of-the-art learning-based ABR algorithm.

II. Background and Motivation

Background: ABR algorithms determine the bitrate of each video chunk based on network and player information obtained when retrieving previous chunks. Existing ABR algorithms can be classified into four categories: rate-based (e.g., [7], [9], [10]), buffer-based (e.g., [22], [24]), mixed (e.g., [8]) and learning-based (e.g., [23]). *Rate-based* approaches and *buffer-based* approaches select the bitrate based on the predicted throughput and the buffer occupancy of the video player, respectively. *Mixed* approaches select the bitrate based on both throughput and buffer level, which are also taken as input in *learning-based* approaches. Note that rate-based and mixed approaches originally require explicit throughput prediction. Besides, even buffer-based and learning-based approaches tend to rely on throughput prediction when deployed in real-world environments, e.g., the change from BOLA [24] to DYNAMIC [12], and from Pensieve [23] to ABRL [14]. Under this circumstance, accurate throughput (or delivery time) prediction is vital to improving the QoE of video streaming.



(a) Inaccurate throughput prediction decreases video quality



(b) Chunk size and real throughput

Fig. 1. A real-world case of RobustMPC.

Motivation: Achieving accurate throughput prediction is still a challenging problem in mobile networks. For both academic research and wildly deployed video services, a large gap exists between the bitrate selected by ABR algorithms and the available bandwidth [25], [26]. A real case under the Wi-Fi connection of a classical ABR algorithm (RobustMPC [8]) is present to illustrate how inaccurate throughput prediction harms QoE. RobustMPC is reported to perform well in the wild Internet [16], predicting chunk throughput based on the harmonic average value of past samples and further decreasing the prediction according to past prediction errors. Fig. 1(a) shows that if the real throughput of one chunk suddenly falls (at the 81st chunk), the predicted throughput of the next chunk by RobustMPC will also decrease. However, when the real throughput of chunks rises in the future (from the 82nd chunk), RobustMPC's predictions cannot react to the changes in time. As a result, although the real throughput is high enough for the highest bitrate (i.e., chunk size divided by chunk duration), RobustMPC keeps selecting lower bitrate levels for three consecutive chunks, unnecessarily reducing video quality and causing fluctuation and QoE degradation.

One will usually attribute the throughput plummeting in Fig. 1(a) only to network bandwidth fluctuations and believes that throughput is difficult to predict [22]. However, this is partly caused by confusing application throughput with available bandwidth, as many prior works have done [7], [9], [10], [23], [27].

[1]To facilitate future work, part of our collected dataset has been made publicly available at https://github.com/GreenLv/Lumos.

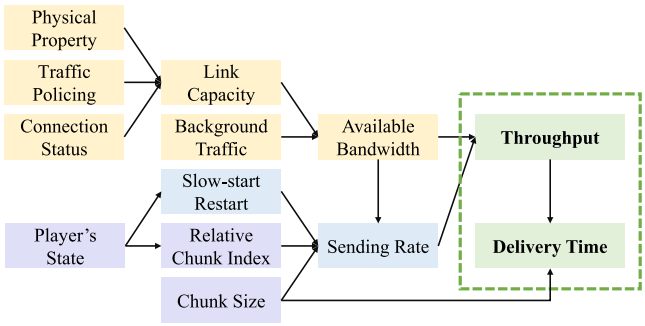


Fig. 2. Theoretical framework about factors impacting chunk throughput and the delivery time.

As illustrated in Fig. 2, many factors influencing throughput are overlooked in most existing prediction schemes. For instance, as shown in Fig. 1(b), it seems that throughput changes in the same trend as chunk size, which was also mentioned in prior works [16], [19]. This indicates that variation in application perceived throughput may not be dominated by network conditions as previously considered. Although throughput prediction of video streaming has already been widely investigated [9], [10], [15], [16], [17], [18], [19], [21], [25], there still lacks a fundamental understanding of this issue, which motivates our work in this paper.

III. Study on Prediction Factors

In this section, we aim to identify the input features of the predictor by answering the following question: *What factors assist in achieving better prediction?* We begin by describing our proposed theoretical framework about factors that impact application throughput and delivery time (Section III-A). We then formulate the chunk delivery procedure to investigate further how these factors affect throughput and delivery time (Section III-B). Finally, we verify our theoretical analysis based on the extensive dataset collected from the real-world mobile Internet and report our observations (Section III-C).

A. Theoretical Framework for Prediction

Throughput prediction in adaptive streaming is not a new topic. Previous studies have analyzed throughput evolution and pursued more accurate predictions. For instance, [19] first studied the influence of the ON-OFF behavior and the background traffic on the application throughput in adaptive streaming. CS2P [10] found that connections with similar status (e.g., ISP, region) exhibit similar throughput patterns, [28] investigated how traffic policing affects throughput, and [15] showed that different CDN layers where chunks are served from lead to different throughput. Inspired by [18], [19], Fugu [16] started considering chunk size in predicting delivery time and using the statistics of the transport layer (e.g., RTT, congestion window, and sending rate) as connection status. As a follower, Xatu [17] combined static features from CS2P with temporal features, including chunk size and Time to First Byte (TTFB).

Despite all these efforts, there is still a lack of fundamental understanding in predicting throughput and delivery time in mobile networks. For this reason, by summarizing existing works, we construct a theoretical framework containing potential factors impacting throughput and the delivery time of chunks, as shown in Fig. 2. The application throughput of a chunk is calculated as the chunk size divided by the delivery time. The delivery time is the duration from when the first byte of the request is made to when the last byte of the response is received [15]. Since delivery time can be calculated from application throughput with chunk size given, factors that impact throughput also impact delivery time. Therefore, we focus on factors directly impacting application throughput.

The application throughput is close to but less than the receiving rate of chunks due to two reasons. First, application throughput only considers bytes in chunks, excluding bytes in headers of protocols at various layers (e.g., TCP and HTTP headers). Second, application throughput involves the processing delay of the application. However, if we disregard these differences, application throughput is equivalent to the receiving rate.

Two factors determine the receiving rate: the bottleneck link's available bandwidth and the sender's sending rate [20]. If the sending rate is below the available bandwidth, it will determine the receiving rate. Otherwise, the receiving rate depends on the available bandwidth. Factors that impact available bandwidth and sending rate are listed below.

- *Available bandwidth* equals link capacity minus background traffic. Link capacity is further determined by the physical property of the link, traffic policing [28], and connection status between the server and the client player. Previous works have explored various methods to characterize connection status, leveraging information of network-side (e.g., LTE base station [9], ISP, region [10], CDN layer [15]) or endpoint-side (e.g., RTT, congestion window [16], TTFB [17], connection type and signal strength [29]).
- *Sending rate* is controlled by the congestion window (CWND) of the server and essentially depends on the available bandwidth, the transport mechanism (e.g., congestion control), and the application behavior. The ON-OFF period is the unique behavior of video streaming [18], [19], indicating that the player requests 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 client player and the server. If the inactivity time exceeds a timeout (200 ms in Linux), the server resets the CWND to the initial size (e.g., 10MSS) and returns to the slow-start phase [19], which is named slow-start restart [23], [30], [31]. Due to the application behavior and transport mechanism, the request pattern of the video player can be divided into two states [18]: 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.

Our theoretical framework reveals the relationship between *network conditions* (the yellow area in Fig. 2), *transport mechanism* (the blue area in Fig. 2), and *application behavior* (the purple area in Fig. 2), as well as their impact on application throughput and delivery time. To the best of our knowledge, this

framework presents a clear roadmap for selecting input features of the network predictor for the first time.

B. Formulation Analysis

To gain a deep understanding of our proposed framework, we formulate the chunk delivery procedure for theoretical analysis to investigate how the transport mechanism and application behavior impact throughput and delivery time.

1) Application Throughput and Delivery Time: The delivery time of a chunk begins with sending the first byte of the request and ends with receiving the last byte of the response, corresponding to several (or more) complete RTT rounds. For the kth chunk, we denote its size, delivery time, and application throughput as S_k, D_k, and T_k, respectively. Let $\overline{RTT}_k$ be the average RTT and m_k be the number of RTT rounds in this chunk. Then we have:

$$D_k = \overline{RTT}_k * m_k. \tag{1}$$

Given S_k, application throughput can be calculated as:

$$T_k = \frac{S_k}{D_k}. \tag{2}$$

(1) and (2) indicate that application throughput depends on the number of RTT rounds in delivering the chunk, namely m_k. We now focus on how to calculate it. Since m_k is determined by how many bytes can be delivered in each RTT and need to be delivered in total (i.e., S_k), we have the following equation:

$$\sum_{i=1}^{m_k-1} cwnd_i^{(k)} < S_k \leq \sum_{i=1}^{m_k} cwnd_i^{(k)}, \tag{3}$$

where $cwnd_i^{(k)}$ indicates the congestion window (CWND) size (bytes that can be sent) in the ith RTT during delivering chunk k. We only discuss the case when $m_k \geq 2$ because video chunks are always larger than the CWND size.

2) State of Congestion Control: CWND size is determined by the congestion control algorithm (CCA). CCAs typically have three states: slow start (SS), congestion avoidance (CA), and fast recovery (FR) [32]. The typical CCA starts with the SS state, doubling its CWND every RTT until the sending rate reaches the available bandwidth. After that, it switches to the CA and FR states, controlling the CWND size in order to avoid congestion in the link. This part illustrates how to calculate the sum of the CWND size for CCAs in all states.

SS State: If the CCA stays in the SS state throughout the delivery of the entire chunk, the sum of its CWND in m_k rounds is calculated as:

$$\sum_{i=1}^{m_k} cwnd_i^{(k)} = cwnd_1^{(k)} * (2^{m_k} - 1), \tag{4}$$

where $cwnd_1^{(k)}$ is the initial CWND size of chunk k.

CA&FR States: In these states, different CCAs adjust CWND in different ways. We will discuss two categories of widely deployed CCAs: loss-based CCA and BBR [20].

i) Loss-based CCA: This type of CCA, such as Reno and Cubic [33], serves as the default CCA in Linux Kernel. In a stable network environment, the loss-based CCA periodically switches between CA and FR states according to the ACKs. We denote this period as p_k RTTs for chunk k ($p_k \geq 2$). Specifically, in the CA state, the CCA's CWND size grows at a certain rate (depending on the algorithm) within $p_k - 1$ RTTs until the packet loss event occurs, where the CWND size is denoted as $W_{\max}^{(k)}$. After that, the CCA enters the FR state and reduces the CWND size to $W_{\min}^{(k)}$ in one RTT before re-entering the CA state. Taking Reno as an instance, this process is known as Additive Increase/Multiplicative Decrease (AIMD), and $W_{\min}^{(k)} = 1/2 * W_{\max}^{(k)}$.

In the p_k RTTs period of any loss-based CCA, the sum of CWND takes on the following range of values (depending on the CWND adjustment function in the CA&FR states):

$$W_{\min}^{(k)} * (p_k - 1) + W_{\max}^{(k)} \leq \sum_{i=1}^{p_k} cwnd_i^{(k)} \leq W_{\min}^{(k)} + W_{\max}^{(k)} * (p_k - 1). \tag{5}$$

Let $W_{\min}^{(k)} = \theta * W_{\max}^{(k)}$, where $\theta \in (0, 1)$. Further, the following equation holds extensively for loss-based CCAs:

$$\sum_{i=1}^{p_k} cwnd_i^{(k)} = \beta * W_{\max}^{(k)} * p_k, \tag{6}$$

where $\theta + (1-\theta)/p_k \leq \beta \leq 1 - (1-\theta)/p_k$ (derived from (5)) and $\beta \in (0, 1)$. For Reno, $\beta = 0.75$ [32]. When the CCA converges to a stable state, p_k, $W_{\max}^{(k)}$, and $W_{\min}^{(k)}$ remain almost unchanged, and the following holds [32]:

$$\beta * W_{\max}^{(k)} = \overline{BW}_k * \overline{RTT}_k, \tag{7}$$

where $\overline{BW}_k$ denotes the average available bandwidth during the delivery of chunk k. More generally, for the entire video chunk delivered in m_k RTTs, we have:

$$\sum_{i=1}^{m_k} cwnd_i^{(k)} = \gamma * W_{\max}^{(k)} * m_k, \tag{8}$$

where $\gamma \approxeq \beta$ ($\approxeq$ means almost equal or equal to) and $\gamma \in (0, 1)$. When m_k is an integer multiple of p_k, γ equals β.

ii) BBR: BBR's ProbeBW phase corresponds to the CA&FR states. Note that we exclude BBR's ProbeRTT phase in the formulation because it occurs infrequently and does not change the main conclusion. In the ProbeBW phase, the CWND size is adjusted per RTT by periodically changing the pacing_gain. Each period lasts for 8 RTTs in BBR, and the pacing_gain is adjusted in the order of [5/4, 3/4, 1, 1, 1, 1, 1, 1], with a mean value of 1 per RTT in the period. Therefore, in every 8-RTT period, the sum of CWND size is $\sum_{i=1}^{8} cwnd_i^{(k)} = 8 * W_{base}^{(k)}$, where $W_{base}^{(k)}$ indicates the base CWND size of BBR. Moreover, for the entire chunk, the sum of CWND size is calculated as:

$$\sum_{i=1}^{m_k} cwnd_i^{(k)} \approxeq m_k * W_{base}^{(k)}. \tag{9}$$

The two sides are not equal when and only when $m_k \bmod 8 = 1$, where the difference is $1/4 * W_{base}^{(k)}$. This difference can always

be negligible due to (3). Additionally, when BBR converges to a stable state, the following holds [20]:

$$W_{base}^{(k)} = \overline{BW}_k * \overline{RTT}_k. \tag{10}$$

Summary: Based on (6) to (10), if the CCA stays in the CA&FR states during chunk delivery, the following equation holds for various CCAs in a stable network:

$$\sum_{i=1}^{m_k} cwnd_i^{(k)} = \overline{BW}_k * \overline{RTT}_k * m_k. \tag{11}$$

Furthermore, the CCA may start from the SS state and ends with the CA and FR states. Let the jth ($j \in [1, \ldots, m_k - 1]$ hereafter) RTT be when the state changes. Given (4) and (11), the CWND sum can be calculated by the following.

$$\sum_{i=1}^{m_k} cwnd_i^{(k)} = cwnd_1^{(k)} * (2^{j-1} - 1) + \overline{BW}_k * \overline{RTT}_k * (m_k - j + 1). \tag{12}$$

Equations (3), (4), (11), (12) clearly indicate that m_k depends heavily on the chunk size and the state of CCA, which is further affected by the state of client player (i.e., buffering state or steady state), as discussed in Section III-A.

3) State of Client Player: In the startup phase of a session, the client player continuously requests new chunks to accumulate in the playback buffer. Once the buffer level exceeds a certain threshold (e.g., 12 seconds in dash.js [34]), the player only requests the next chunk when the buffer level falls below the threshold due to playback. This creates two different request patterns for the player: the buffering and steady states. In the steady state, the CCA starts from the SS state (due to slow-start restart)[2] for each chunk, increasing the CWND from the initial size (e.g., 10MSS in Linux Kernel Cubic [33]). On the other hand, in the buffering state, the initial CWND size of a chunk equals the CWND size in the last RTT of the previous chunk. Therefore, chunk delivery is independent for each chunk in the steady state, but not in the buffering state.

Steady State: We begin with calculating m_k in the steady state. If the CCA stays in the SS state for the entire chunk transmission, considering (3) and (4), we have:

$$m_k = \left\lceil \log_2 \left(\frac{S_k}{cwnd_1} + 1 \right) \right\rceil, \tag{13}$$

where $\lceil x \rceil$ rounds up x to the nearest integer, and $cwnd_1$ is the constant initial CWND size for all chunks, such as 10MSS. Correspondingly, if the CCA changes from the SS to the CA&FR states in the jth RTT, given (3) and (12), m_k is calculated as:

$$m_k = \left\lceil \frac{S_k - cwnd_1 * (2^{j-1} - 1)}{\overline{BW}_k * \overline{RTT}_k} \right\rceil + j - 1. \tag{14}$$

Buffering State: In the buffering state, the CWND size keeps increasing with consecutive chunks until the CCA transitions from the SS to the CA&FR states, which is assumed to occur in the jth RTT while delivering the chunk k_l. We ignore the case where the CCA only stays in the SS state during the player's buffering state because it is less likely. Then we can compute m_k as follows:

$$m_k = \begin{cases} \lceil \log_2(\frac{S_k}{cwnd_1^{(k)}} + 1) \rceil, & k < k_l, \\ \lceil \frac{S_k - cwnd_1^{(k)} * (2^{j-1} - 1)}{\overline{BW}_k * \overline{RTT}_k} \rceil + j - 1, & k = k_l, \\ \lceil \frac{S_k}{\overline{BW}_k * \overline{RTT}_k} \rceil, & k > k_l. \end{cases} \tag{15}$$

Note that $cwnd_1^{(k)} = cwnd_{m_{k-1}}^{(k-1)}$ when $k \geq 2$.

4) Summary: Based on (1), (2), (13), (14), (15), we draw the following conclusions.[3] *(i)* Application throughput (as well as delivery time) of chunks is influenced by network conditions ($\overline{BW}_k$ and $\overline{RTT}_k$), transport mechanism (CCA's state), and application behavior (chunk size S_k and player's state), which is consistent with the theoretical framework proposed in Section III-A. Specifically, *(ii)* when the player is in the steady state, chunk throughput heavily depends on chunk size, as shown by the nonlinear (piece-wise linear or logarithmic) relationship in (13), (14). On the other hand, *(iii)* in the buffering state, the relative index of successive chunks affects throughput when sending rate is below available bandwidth, corresponding to $cwnd_1^{(k)}$ in (15).

To verify our theoretical framework and formulation analysis, we conduct an in-depth measurement-based study of real-world mobile adaptive streaming, as described in the following subsection.

C. Measurement in Real-World Mobile Internet

In this part, we first describe our automated video streaming measurement platform and the extensive dataset of video streaming collected through this platform in the real-world mobile Internet (Section III-C1). By analyzing the collected dataset, we identify the factors that affect the throughput of video chunks and quantitatively characterize how they assist in throughput prediction (Section III-C2). Next, we examine the correlation between throughput and chunk size, as well as the impact of various factors on this correlation (Section III-C3).

1) Mobile Video Streaming Measurement Platform: To collect the dataset for analysis, we built an automated video streaming measurement platform following the DASH specification, as shown in Fig. 3. The platform consists of a client that runs the video player and a server that stores the video content.

Video Content: We used Elephant Dream and Big Buck Bunny, two raw videos from [35]. Each video is approximately 10 minutes long. Using FFmpeg [36], we encoded these videos 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] (following [13], [23]). We created two video versions for each bitrate with chunk durations of 2 and 4 seconds, respectively. Each version of the video is indexed

[2]For other CCAs (such as BBR) that do not incorporate the slow-start restart, their CWND is *theoretically* not affected by the player's state. Consequently, the value of m_k for these CCAs is determined solely by (15).

[3]Note that this formulation provides the upper bound of application throughput (also the lower bound of delivery time). In real-world mobile networks, the actual throughput can be lower due to changes in bandwidth ($\overline{BW}_k$) and RTT ($\overline{RTT}_k$). Therefore, using data-driven methods is more practical to capture real-world dynamics (see Section IV).

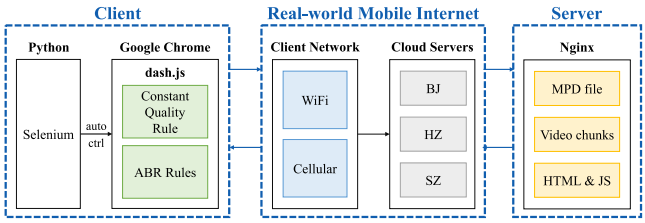


Fig. 3. Architecture of the real-world mobile video streaming measurement platform.

by a Media Presentation Description (MPD) file generated by MP4Box [37].

Video Client: We used the Google Chrome browser with dash.js [34] as the video player. Python scripts based on Selenium [38] were developed to automatically control the browser, imitating users' requesting and playing behavior. The dash.js settings were kept as default. The scripts ran on Windows laptops, which were connected to APs by WiFi (2.4 GHz or 5 GHz) or cellular (4G)[4] hotspots through USB.

Video Server: We used the Nginx server [40] to host video content and dash.js codes. The server was deployed across three distinct cloud servers located in three different cities, all running Ubuntu 16.04, with different downstream bandwidths (5 Mbps or 50 Mbps). We also deployed scripts on the server to interact with the client. Note that our measurement platform is deployed in the real world, so we conducted tests in the wild mobile Internet instead of an emulator with throughput traces.

Methodology: A test contains several sessions and is conducted as the following.

- Before each test starts, we record the client device's connection type and wireless signal strength. Since our aim is to conduct tests rather than implement a complete production system, the connection type is manually input into the scripts.[5] As for signal strength, RSSI of WiFi is automatically obtained by pywifi [42] in the scripts, while RSRP and SINR of cellular are manually fetched from cellular-Z [43]. In practice, the signal strength of WiFi and cellular can be accessed through specific Android APIs, such as [44], [45]. To eliminate the impact of mobility, we conducted tests with the client remaining still.
- To obtain data of chunks at all bitrates in one test, we implemented a custom rule in dash.js that selects a constant bitrate for the entire video session.[6] 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 the codes of the dash.js player from the server and initiates the video session. In this way, all six bitrate levels can be traversed in a test with six sessions.
- During each session, we collect information from both the client player and the server's TCP/IP stack. 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 on TCP connection via tcp_probe in the Linux kernel, including CWND size, slow-start threshold, smoothed RTT, and inflight size.

We conducted extensive tests for each combination at different time periods of the day (morning, noon, afternoon, or evening), in different locations (four cities for the client), and with different access network types (cellular or WiFi), to cover as many usage scenarios as possible. In total, we collected data from 2500+ video sessions (800+ with a constant bitrate and 1700+ with ABR algorithms) containing 300,000+ video chunks from December 2019 to May 2021.

2) Factors That Impact Throughput Prediction: To investigate how factors in our theoretical framework (Fig. 2) contribute to predicting the throughput of video chunks, we evaluate the maximal information coefficient (MIC) [46] of throughput and these factors. MIC measures the strength of the linear or nonlinear association between two variables based on the mutual information [47], ranging from 0 (no correlation) to 1 (perfect correlation).

Since it is difficult to accurately obtain the link's physical property and background traffic, we use past throughput samples (e.g., the throughput of the last chunk) to roughly estimate their effects, which are widely used to predict throughput in ABR algorithms [7], [8], [21]. We use the downstream bandwidth of the server to indicate traffic policing (Section III-C1). Regarding connection status, several prior works focus on features of the whole network, such as information of ISP, AS [10] and CDN [15]. However, this information can not directly describe the network environment of the mobile 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 [29], to characterize wireless connection status (Section III-C1). We classify signal strength into three categories (strong, middle, and weak) according to the RSSI of WiFi and RSRP of cellular for comparison. Moreover, we set the inactivity time to 200 ms to distinguish the two states of the player, which are indicated by the relative index. The relative index of each chunk is increased by consecutive chunks in the buffering state; otherwise, it stays 0. Fig. 4 shows the MIC score of throughput and each considered factor, computed using Python minepy library [48].

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

Previous works that only consider past throughput samples, such as [7], [8], fail to accurately predict throughput due to their disregard for target chunk size. This distinction is important because application throughput varies with different chunk sizes, whereas available bandwidth does not. Although other factors have a less direct impact on predicting throughput, we argue that incorporating them leads to better prediction accuracy, an investigation detailed in the following subsection.

[4]We also conducted tests under 5G connections. Since 5G bandwidth is always above 100Mbps [29], [39], the results of these tests are similar to those conducted under strong signal strength with 4 G.

[5]The connection type can be accessed in dash.js through Network Information API [41]. However, desktop browsers did not support this experimental API when we built this platform.

[6]We further implemented several existing ABR algorithms for training and testing; see Sections IV-B and V.

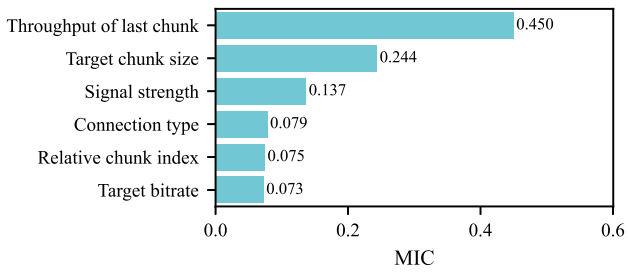


Fig. 4. MIC score of throughput and its impacting factors.

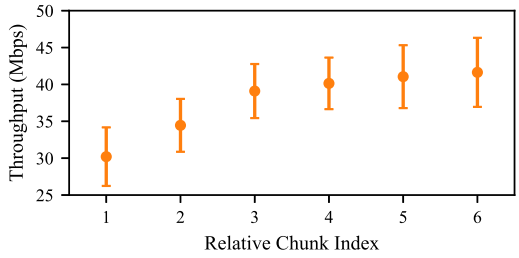


Fig. 6. Chunk throughput increases with relative chunk index in the buffering state (with 95% confidence).

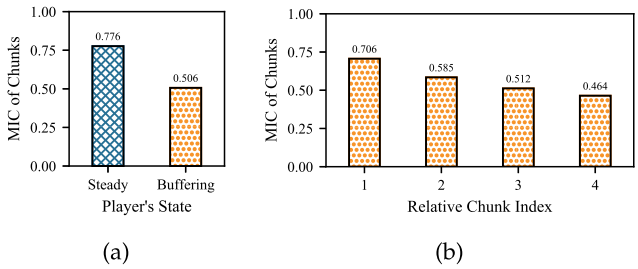


Fig. 5. MIC correlation of throughput and chunk size varies in different player's states.

3) Correlation Between Throughput and Chunk Size: Both prior works [18], [19] and our formulation analysis in Section III-B have indicated that application throughput is related to chunk size. To further investigate this relationship, we utilize MIC to assess the correlation between throughput and chunk size and make the following observation.

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

Player's State: To investigate the impact of the player's state on the correlation, we focus on data collected under the best network condition, i.e., strong signal strength with 50 Mbps downstream bandwidth, where link capacity is sufficient for delivery with the highest bitrate. Under this condition, the CWND evolution closely follows the formulation analysis in Section III-B. Fig. 5(a) 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 independently from the initial value for each chunk before congestion is triggered (i.e., the CCA is in the SS state). In this case, a larger chunk size leads to a larger CWND and thus higher throughput, as illustrated in (13).

We further analyze how the correlation is affected by the relative chunk index in the buffering state. Fig. 5(b) shows that correlation decreases as the relative chunk index increases. This is caused by successive chunks continuously ramping up CWND and the sending rate ($k < k_l$ in (15)), eventually up to the available bandwidth before the buffering state ends ($k \geq k_l$ in (15)). Additionally, we also find that in the buffering state, throughput is positively correlated to the relative chunk index, as shown in Fig. 6.

Bitrate Level: Fig. 8(a), which pertains to the steady state under a strong network with 50 Mbps downstream bandwidth, illustrates that the correlation decreases as the bitrate increases. The reason is similar to the above. At higher bitrate levels, the large chunk size allows the CWND to ramp up to a larger size, where the server's sending rate reaches the available bandwidth. As a result, application throughput is mainly determined by the network condition rather than the chunk size, as shown in (14).

Fig. 7 further depicts scatter plots of chunk throughput versus chunk size at all bitrate levels. We can clearly see that scatters of data points form two non-linear curves with different upper bounds, corresponding to the buffering and steady states, respectively. These results are consistent with the conclusions of our theoretical analysis in Section III-B4.

Network Condition: We conducted tests in different network conditions to investigate how traffic policing and connection status affect the correlation. The following results show that changes in $\overline{BW}_k$ and $\overline{RTT}_k$ in real-world mobile networks make it difficult to directly calculate application throughput based on our formulation equations in Section III-B.

- *Downstream bandwidth of the server:* We limited the downstream bandwidth of the server to 50 Mbps and 5 Mbps, respectively, and showed the impact of traffic policing on the correlation in Fig. 8(b) (only steady state under strong signal). We found that the correlation under low downstream bandwidth is much lower than that under high downstream bandwidth because delivering a chunk under low downstream bandwidth is more easily affected by traffic policing, such as the token bucket [28].
- *Wireless signal strength of the client:* Fig. 8(c) presents the impact of signal strength on the correlation (only steady state with 50 Mbps bandwidth). As signal strength weakens, the correlation becomes smaller correspondingly. To explain this observation, we analyzed the data collected from the transport layer by tcp_probe. We found that when the signal strength is weak, smoothed RTT tends to be longer for the larger chunks (especially at the higher bitrates), which directly leads to an increase in delivery time and a decrease in throughput. This phenomenon has also been reported by prior works, such as [49], which mentioned that RTT is correlated with inflight size.

Summary of Observations: These results all indicate that application throughput depends on both the available bandwidth affected by network conditions and the sending rate affected by

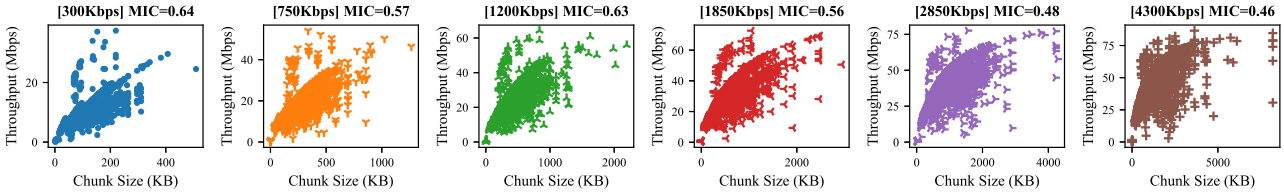
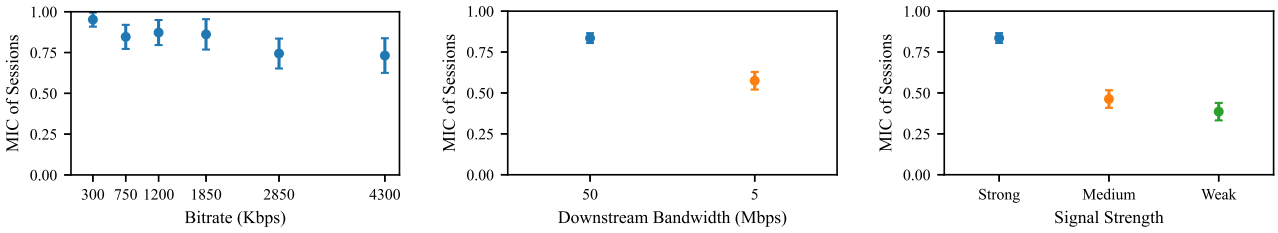


Fig. 7. Scatter plots of chunk throughput versus chunk size at all bitrate levels in both buffering and steady states.



(a) Under different bitrate levels (b) Under different bandwidths (c) Under different signal strengths

Fig. 8. MIC correlation of throughput and chunk size varies under different factors (with 95% confidence).

transport mechanism and application behavior. This finding confirms the validity of our theoretical framework and formulation analysis. While previous studies have concluded that throughput positively correlates with chunk size, we find that this correlation is heavily influenced by the player's state, video bitrate, and network conditions in the real world. Therefore, to improve accuracy when predicting throughput, these factors should be considered.

IV. Lumos: Design and Implementation

Our theoretical analysis and real-world observations in Section III indicate that involving specific information can lead to more accurate application throughput prediction. Based on that, we propose Lumos, a throughput predictor for mobile adaptive streaming. This section introduces the design choice (Section IV-A) and details (Section IV-B) of Lumos, and shows how Lumos is integrated into existing ABR algorithms as a plug-in (Section IV-C).

A. Design Choice

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

i) Output target: Which is a better target to predict, throughput or delivery time? 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 [7] and MPC [8]) utilize predicted throughput to select the bitrate, while recent ones (e.g., Fugu [16] and Xatu [17]) argue that delivery time is a better choice. We wonder if there are differences in predictions between these two targets.

ii) Mapping function: Which model is more suitable for prediction in mobile video streaming? As discussed in Sections III-B and III-C, dynamics in mobile networks (e.g., due to traffic policing or weak signal strength) are hard to model by formulation equations. Therefore, state-of-the-art mapping functions are built by data-driven methods, including multiple linear regression (MLR), decision trees, and Deep Neural Networks (DNNs). Although DNNs can accurately model complex behavior, prior studies have shown that they are heavyweight and lack interpretability [14], [50], [51], making them difficult to design and deploy. In contrast, recent works [51], [52] demonstrate that decision trees are simple yet powerful enough to fit complex functions for prediction, and provide high interpretability. Random forest, a decision-tree-based method, is not considered here, whose computation and storage overhead grow linearly with the number of decision trees (100 as default in sklearn [53]). This overhead is unaffordable when combined with ABR algorithms that require many predictions per chunk, such as MPC [8], as shown in Section V-E3. For these reasons, we develop predictors based on decision trees, and also MLR for comparison.

To investigate the questions above, we develop different predictors based on decision trees and MLR for each prediction target separately, and evaluate their performance on prediction through controlled experiments. These predictors are developed mainly following the methodology described in Section IV-B,[7] but without the two improvements of Lumos.

Prediction Metrics: Accurate prediction of throughput and delivery time enables the ABR algorithm to select bitrates more precisely, resulting in better QoE [10], [16], [39]. Therefore, we evaluate the performance of the two targets based on prediction

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

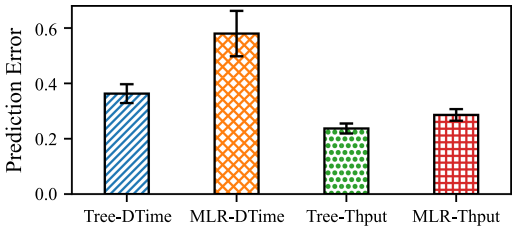


Fig. 9. Prediction error of the four considered predictors (with 95% confidence).

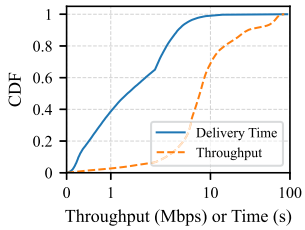


Fig. 11. CDF of delivery time and throughput of video chunks.

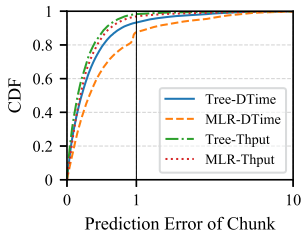


Fig. 10. CDF of the prediction error of video chunks by various predictors.

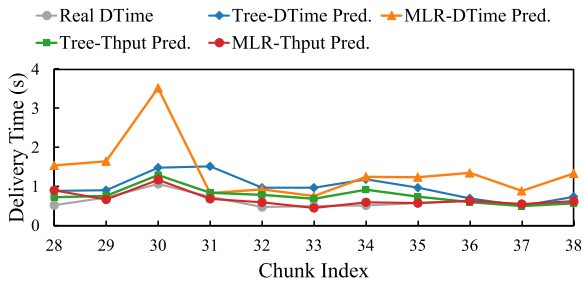


Fig. 12. A case of the predicted delivery time of various predictors.

accuracy. Following [10], [15], we use Absolute Normalized Prediction Error Err (prediction error for short) as the metric, defined as (16).

$$Err(DTime) = \frac{1}{N}\sum_{k=1}^{N}\frac{|\hat{D}_k - D_k|}{D_k}, \tag{16}$$

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

Controlled Experiments: Fig. 9 shows that decision trees (Tree for short) outperform MLR in prediction. Moreover, for both the two types of models, throughput prediction has a lower prediction error (34.7%∼50.7% reduction) compared with delivery time prediction. Thus, we have the following observation.

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

This observation is striking because it suggests that to predict the delivery time, it is better to construct a throughput predictor and convert its predictions to delivery time, rather than directly using a delivery time predictor.

To determine the underlying reasons, we first ask *why the prediction error of throughput predictors is lower*. Prediction error measures the amount by which 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. 10 shows the prediction error's Cumulative Distribution Function (CDF) of each video chunk (not session).[8] 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, leading to higher prediction errors.

The second question is *why delivery time predictors tend to overestimate real values*. We find that the root cause is the long-tailed distribution of delivery time in mobile networks. We consider values with a throughput greater than 100 Mbps or a delivery time greater than 100 s as outliers and filter them out, leaving us with 69163 data points for analysis. As shown in Fig. 11, delivery time is much more long-tailed, with 0.8% of chunks lying in the 89.2% tail of time interval (from 10 s to 93 s). This phenomenon is known as a character of the wild mobile Internet, as noted in prior works [16], [54]. The data distribution determines the models for both regression decision trees and MLR. Delivery time predictors learn higher values due to the long tail of the distribution of delivery time and thus overestimate the actual values, as shown in Fig. 12. In contrast, the distribution of throughput is more "uniform". As a result, throughput predictors can faithfully learn the characters of most data and avoid being affected by outliers at the long tail.

Understanding the Tail of Delivery Time: Since power-law behavior is a classic sign of the long-tailed phenomenon and is widely reported to exist on the Internet [9], [55], [56], [57], [58], we wonder whether it also applies to the delivery time of chunks in mobile adaptive streaming. [59] gives the standard form of the probability density function of the power law distribution (also

[8]Fig. 10 and Fig. 11 are plotted with a logarithmic horizontal axis for display convenience.

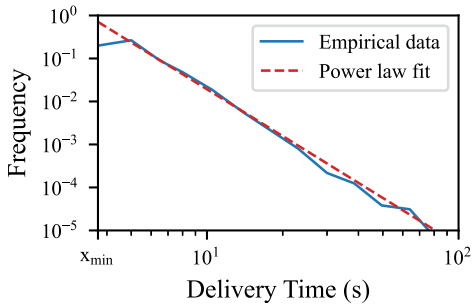


Fig. 13. Log-log plot of frequency versus chunk delivery time.

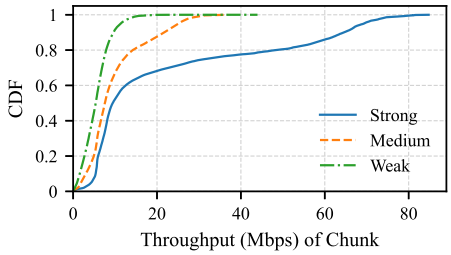


Fig. 14. Throughput under different signal strengths.

known as Pareto distribution or Zipf's law [60]), defined as:

$$p(x) = \frac{\alpha - 1}{x_{\min}} \left(\frac{x}{x_{\min}} \right)^{-\alpha}, \quad \alpha, x_{\min} \geq 0, \quad x \geq x_{\min}, \quad (17)$$

where α is the exponent of the power law. A power-law distribution appears as a straight line with slope α when plotted with logarithmic horizontal and vertical axes [61].

Based on prior works in quantifying power law [59], [62], we find that the tail of delivery time in mobile networks can be well-fitted by a power-law distribution, as shown in Fig. 13. Specifically, when $x_{\min} = 3.69$, the 11.6% tail (accounting for 96% of the entire time interval) of the delivery time can be modeled by a power-law distribution with $\alpha = 3.62$. This result is highly consistent with the observation in [59] that only the tail of a distribution follows a power law in most cases. The power law indicates that events with larger values appear less frequently. Thus, the tail of the delivery time, which may trigger significant stalling [39], [54], is hard to predict. Furthermore, given that power-law distribution is scale-free [61] (or scale-invariant [56]), no matter how to filter the outliers, the distribution is constantly unchanged. Therefore, we cannot improve the performance of delivery time predictors by pre-processing data, making predicting throughput a better choice for data-driven methods.

B. Lumos Mechanism

After identifying the answers to the three problems about input features, output target, and mapping function, we can establish a clear design principle for our decision-tree-based throughput predictor: Lumos. Lumos involves the player's state, chunk information, and networking conditions as features. In this subsection, we describe how to design, train, and implement Lumos.

Design: Training decision trees is a supervised learning process. Since the values of throughput are continuous, we use regression trees. Besides, we tried to train classification trees that discretize throughput into bins, but they perform worse than regression trees. We select the input features for models following the observations in Section III. Given that the 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:

- Network conditions: including (1) the *maximum of throughput of past t chunks* to estimate available bandwidth [20], (2) the *maximum of the delivery time of past t chunks*, (3) the *connection type* (WiFi or cellular) of the client, and (4) the *throughput of the last chunk*;
- Player's state: including (5) the *relative index of the last chunk*, which indicates the player's state by 0 for the steady state and others for the buffering state;
- Chunk information: including (6) the *bitrate* and (7) the *size of last chunk*, and (8) the *bitrate* and (9) the *size of the target chunk*.

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

- *Separate predictor for each network environment:* As shown in Fig. 14, we divide the network environment into three categories according to the signal strength of the mobile client (Section III-C2) 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 [27], [54], [63].
- *Logarithm value of throughput as labels:* Since the throughput values lie in a wide range across 5 orders of magnitude, we use logarithm values of throughput as labels instead of original values. As for obtaining predicted throughput, we calculate 10 raised to the power of which Lumos outputs.

Training and Implementation: In order to train decision trees, we use the Classification and Regression Tree (CART) algorithm [64]. For regression trees, CART utilizes mean squared error (MSE) as the loss function and generates trees by minimizing the MSE between predicted and real values. We implement decision trees in Lumos based on sklearn [65], and train and test Lumos using a video dataset that is the same as the one deployed on our measurement platform Section III-C1. We select data from 700+ video sessions running ABR algorithms (not selecting constant bitrate) collected in real-world mobile Internet containing 69,000+ chunks. 70% of all sessions are used as the training set. Since an imbalanced dataset harms the performance of decision trees [51], we balance the data of various combinations of network conditions (downstream bandwidth and connection type). Specifically, for each class

TABLE I
PARAMETER SEARCH SPACE OF REGRESSION TREES

Parameter	Meaning & Search space
max_depth	the maximum depth of the tree [5, 6, ..., 19, unlimited]
min_samples_leaf	the minimum samples of each leaf [1, 5, 10, 20, 25, 30, 40, 50]

TABLE II
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

of signal strength, we ensure that the count of chunks under 50 Mbps-cellular, 5 Mbps-cellular, 50 Mbps-WiFi, and 5 Mbps-WiFi is roughly equivalent.

During model training, both prepruning and Cost Complexity Pruning (CCP) [64] are applied to prune trees to prevent overfitting. In order to identify the optimal pruning parameters, we conduct an exhaustive grid search and K-fold cross-validation (K = 5) to select the best model. The parameter search space is presented in Table I. After training, all of the models are converted to JavaScript and loaded into dash.js for deployment.

C. Lumos as a Plug-In of ABR Algorithms

Lumos can be integrated into any ABR algorithm that uses throughput prediction as a throughput predictor in mobile adaptive streaming. We use three classic ABR algorithms as examples to show how Lumos is applied to existing schemes. We further compare the performance of Lumos-assisted ABR algorithms with that of original ABR algorithms in Section V. The three ABR algorithms we use are:

i) Rate-Based (RB), which selects the highest bitrate below the predicted throughput by harmonic mean (HM) of throughput samples for past chunks. We integrate Lumos with RB by replacing the HM predictor with Lumos. As bitrates of video chunks (chunk size divided by chunk duration) encoded with H.264/MPEG-4 fluctuate widely around the average bitrate [13], RB+Lumos selects the chunk according to its real bitrate instead of the average bitrate to make reasonable decisions, as shown in (18). Table II lists variable denotations.

$$R_k = \max_{1\le j\le m} \{r_j, d_k(r_j)/\hat{T}_{k|r_j} \le L\} \tag{18}$$

ii) MPC [8], which uses both buffer level and predicted throughput (by HM predictor, the same as RB) to select the bitrate that maximizes the estimated QoE of a series of consecutive 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 (19).[9] As Lumos requires past throughput as features, for the future chunks except the first one, Lumos takes its predicted throughput of the prior chunk as input.

$$R_k = \arg\max_{r_j, 1\le j\le m} \sum_{l=k}^{k+n-1} \hat{QoE}(R_{l|r_j}) \tag{19}$$

$$\hat{QoE}(R_{l|r_j}) = R_{l|r_j} - \max\left(\mu\left(\frac{d_l(R_{l|r_j})}{\hat{T}_{l|r_j}} - B_l\right), 0\right) - \lambda|R_{l|r_j} - R_{l-1}|, k \le l \le k+n-1 \tag{20}$$

iii) Buffer-Based Approach (BBA) [22], which builds a linear mapping function between the level of playback buffer and target bitrate, and sets both lower and upper bounds of the buffer to select bitrate. When the buffer level is below the lower bound or above the upper bound, BBA selects the lowest bitrate or highest bitrate, respectively. When the buffer level is between the lower and upper bound, it chooses the bitrate simply according to how much the current buffer level exceeds the lower bound. Although BBA makes decisions without prediction, we argue that accurate prediction assists BBA in achieving better QoE. We integrate Lumos with BBA by replacing the linear mapping function with Lumos's predicted values. 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 the next chunk, BBA+Lumos sets a threshold: the lower bound plus the difference between its predicted delivery time and that of the lowest bitrate. BBA+Lumos selects the bitrate only when the buffer level exceeds the corresponding threshold, as illustrated in (21).

$$R_k = \max_{1\le j\le m} \left\{ r_j, B_{lower} + \frac{d_k(r_j)}{\hat{T}_{k|r_j}} - \frac{d_k(r_1)}{\hat{T}_{k|r_1}} \le B_k \right\} \tag{21}$$

V. EVALUATION IN THE WILD MOBILE INTERNET

This section presents evaluation results of Lumos's prediction performance (Section V-A), the QoE performance of Lumos-assisted ABR algorithms (Sections V-B, V-C, and V-D) in real-world mobile networks, and a deep dive into how Lumos uses input features in prediction and Lumos's overhead (Section V-E).

Methodology: We implemented the three ABR algorithms above (Section IV-C) with their Lumos-assisted versions, as well

[9] In (20), $|R_k - R_{k-1}|$ is equal to 0 when $k = 1$

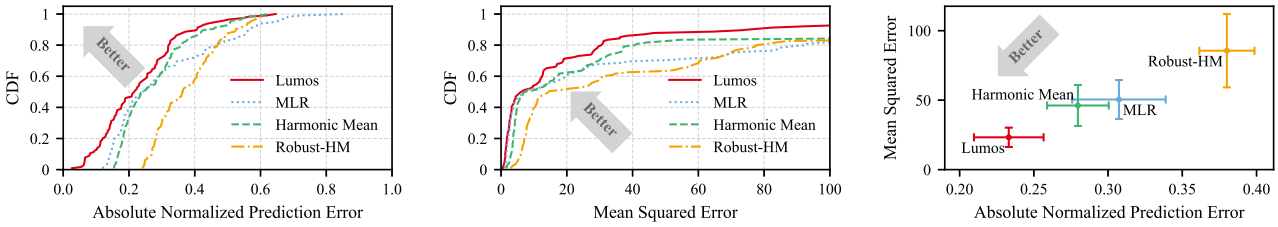


Fig. 15. Prediction performance of various throughput prediction methods.

as two additional schemes: RobustMPC [8] and Pensieve [23]. These were deployed in dash.js and tested on our measurement platform, as described in Section III-C1. To evaluate all the algorithms, we conducted extensive sets of tests with about 300 sessions to cover various network environments, including three variables: downstream bandwidth of the server (50 Mbps or 5 Mbps), connection types (WiFi or cellular), and signal strength (strong, medium and weak) of the mobile client. We balanced the sessions under different network conditions, as during the training of Lumos (Section IV-B). In addition, the default target buffer threshold in dash.js is 30 or 60 seconds (depending on the total length of the video) for the highest bitrate and 12 seconds for the lower bitrates. However, tests with a large target buffer yielded similar QoE after the startup phase for all ABR algorithms, as the bandwidth is higher than the highest bitrate in most cases under strong or medium signal strength. Therefore, we set the target buffer threshold to constant 12 seconds for all bitrates to better differentiate the performance of various ABR algorithms in the steady state.

A. Prediction Accuracy of Lumos

First, we evaluate the accuracy of Lumos in predicting the throughput of video chunks.

Baselines and Metrics: We choose three widely used methods as baselines: 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 [8]). We use both prediction error and mean squared error (MSE) to evaluate the accuracy of predicting throughput.

Overall Prediction Accuracy: Fig. 15 shows the prediction accuracy of various methods. Lumos achieves the best throughput prediction performance in terms of prediction error and MSE. Specifically, Lumos reduces two types of error for 87.8% and 82.6% of sessions compared to MLR, and for 91.3% and 93.9% of sessions compared to HM. Regarding the average performance, Lumos reduces two types of error by 16.8%∼38.7% and 49.6%∼72.8% compared to all baselines.

Performance Under Various Environments: It is notable that under strong networks connected with WiFi, Lumos achieves a remarkably low prediction error with an average of only 7.4% (Fig. 16 gives an example), and improves 57.7%∼74.0% and 78.5%∼90.5% in terms of two types of error compared to other methods. Even under weak networks, Lumos still improves accuracy by 9.1%∼28.9% and 17.8%∼61.7% over the two

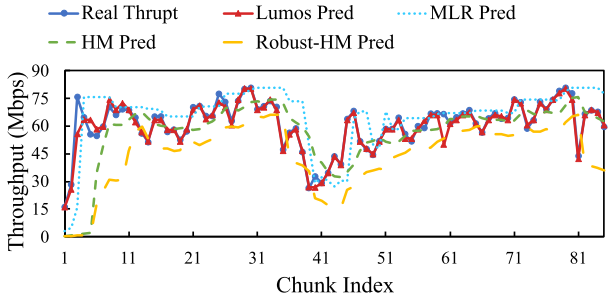
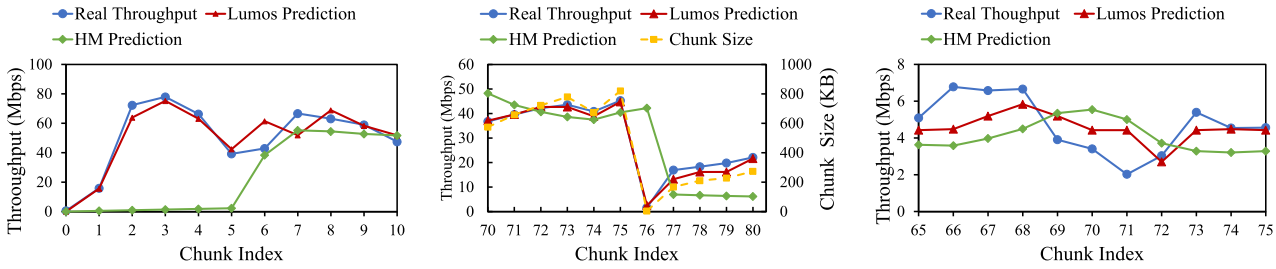


Fig. 16. A case of Lumos's accurate throughput prediction under a strong network environment.

metrics. These results demonstrate that Lumos learns well about the correlation between throughput and the considered factors.

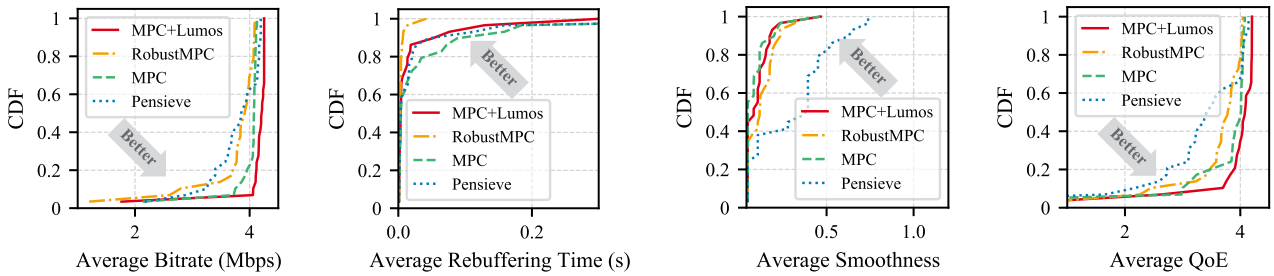
Comparison With HM: Lumos achieves significantly better prediction accuracy than HM, which is widely used in existing ABRs. Results indicate that the advantages of Lumos over HM are as follows:

- *Awareness of network conditions and player's state:* When there is not enough past data, i.e., in the start-up phase of a session, it is difficult for predictors to make accurate predictions [10]. Under strong networks connected with WiFi (Fig. 17(a) as an example), HM obtains an average prediction error of 95% for the first 5 chunks of sessions due to a lack of past samples. In the same situation, Lumos reduces the average prediction error to only 16% by involving network conditions and the player's state.
- *Consideration of the fluctuation of chunk sizes:* HM assumes that the variation of available bandwidth determines throughput fluctuation. However, our analysis and observations indicate that the chunk size strongly affects the perceived throughput when the available bandwidth is adequate. Fig. 17(b) shows a case in the steady state under the strong network and demonstrates that Lumos clearly knows how the fluctuation of chunk size affects throughput.
- *Quick reaction to bandwidth fluctuation:* Under weak network conditions, throughput is always unstable with frequent fluctuation, making it almost unpredictable. In such scenarios, Lumos exceeds HM by quickly reacting to fluctuations, as illustrated in Fig. 17(c). From the 69th chunk, throughput suddenly decreases from 6.7 Mbps to 3.9 Mbps, lasting for the subsequent two chunks. Since HM is insensitive to outliers [7], it should lower the predicted values but increases them instead for three consecutive



(a) Performing well in start-up phase (b) Taking chunk size into consideration (c) Reacting quickly to fluctuation

Fig. 17. Cases of Lumos outperforming HM in throughput prediction.



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

Fig. 18. QoE Performance of MPC+Lumos versus MPC, RobustMPC and Pensieve in real-world mobile networks.

chunks, which may trigger rebuffering events. Contrastingly, Lumos instantly perceives the dynamics of throughput and makes more accurate predictions accordingly.

B. QoE of Lumos-Assisted MPC

We now evaluate the QoE performance of MPC+Lumos in real-world mobile networks, comparing it to MPC and RobustMPC. MPC models the evolution of perceived QoE in future chunks based on predicted throughput. The accuracy of the throughput prediction directly corresponds to the QoE performance of MPC [8].

QoE Metrics: To evaluate the QoE performance of the three ABR algorithms, we use the metrics in MPC [8], defined as:

$$QoE = \sum_{k=1}^{N} R_k - \mu \sum_{k=1}^{N} \max\left(\left(\frac{d_k(R_k)}{T_k} - B_k\right), 0\right) - \lambda \sum_{k=1}^{N-1} |R_{k+1} - R_k|, \quad (22)$$

where N is the number of chunks in the session, R_k is the bitrate that the client selects for chunk k, the second item represents the rebuffering time during delivering chunk k, and the last item represents the smoothness of the 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 previous works [23], [27], [50].

Overall QoE Performance: Fig. 18 displays the QoE of MPC, RobustMPC, and MPC+Lumos. Combining the results in Fig. 18(a) and (b), MPC+Lumos always selects higher bitrates 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 better accuracy than Robust-HM (Section V-A), the 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) and total QoE. Moreover, compared with MPC, MPC+Lumos makes more aggressive decisions (3.4% increase in bitrate) while having a 52% reduction in rebuffering time. These results confirm the benefits of Lumos's accurate prediction.

Performance Breakdown: We further observe that under various network environments in real-world mobile Internet, MPC+Lumos consistently outperforms MPC and RobustMPC as shown in Fig. 19. Specifically, MPC+Lumos improves the average QoE of sessions by 3.81%, 2.73%, and 10.82% with respect to MPC under strong, medium, and weak networks, respectively. Compared to RobustMPC, these improvements are 5.49%, 9.75%, and 10.32%, respectively. Note that RobustMPC performs slightly better than MPC under weak networks.

Under strong networks, MPC+Lumos improves QoE mainly during the startup phase by selecting much higher bitrates for chunks. Fig. 20 shows that MPC+Lumos can select the highest bitrate level from the 2nd chunk, while neither MPC nor

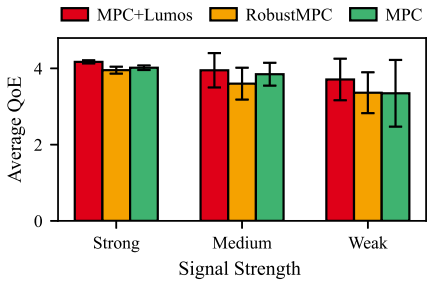


Fig. 19. MPC+Lumos outperforms MPC and RobustMPC under all conditions.

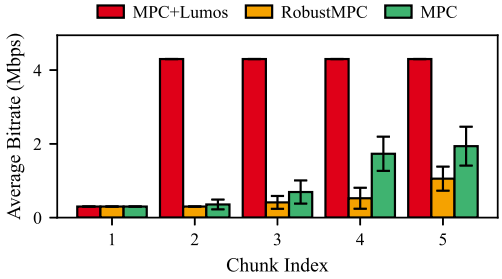


Fig. 20. MPC+Lumos selects the highest bitrate during the start-up phase in strong networks.

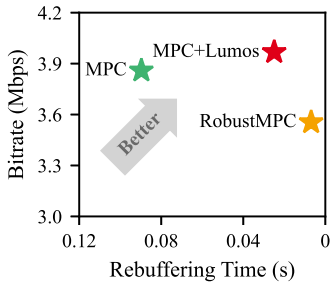


Fig. 21. MPC+Lumos performs best in weak networks.

RobustMPC can achieve this for the first 5 chunks. Additionally, based on Lumos's accurate prediction, MPC+Lumos achieves the highest bitrates under weak networks while avoiding severe rebuffering events, as illustrated in Fig. 21. These results are consistent with the advantages of Lumos over HM as discussed in Section V-A.

C. MPC+Lumos versus Pensieve

Pensieve [23] is a state-of-the-art learning-based ABR algorithm that utilizes deep reinforcement learning to optimize the overall QoE. To compare Lumos-assisted ABR algorithms with Pensieve, we selected MPC+Lumos because it achieves better QoE than others.

Pensieve Deployment: As the performance of Pensieve heavily relies on the similarity between the training and testing environments, we used the source code provided by the authors [66] and retrained Pensieve with our video dataset and network traces (as used to train Lumos) collected in the wild mobile networks, following [16], [27]. We converted Pensieve's model into JavaScript and deployed it in dash.js by TensorFlow.js [67].

Comparison With Pensieve: Fig. 18 shows the QoE metrics of MPC+Lumos and Pensieve. Compared to Pensieve, MPC+Lumos achieves remarkable improvement in QoE, outperforming Pensieve on 93% of all the sessions, with an increase in QoE of 19.2% on 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 why MPC+Lumos outperforms Pensieve in real-world mobile networks are summarized below.

- *Bias between the simulator and the real world:* The simulator used to train Pensieve's model operates with constant network traces, where bitrate selection does not change chunk throughput. As noted in Section III and recent studies [16], [54], [68], this simulation environment differs significantly from the wild mobile Internet. Furthermore, Pensieve's simulator is designed with the precondition of disabling slow-start restart [23]. In the real world, most Linux servers enable slow-start restart by default, which Pensieve is blind to. Consequently, Pensieve may have difficulty accurately predicting throughput dynamics in the real world.
- *Limited ability of generalization:* The throughput of chunks varies greatly in the real world, with the maximum and minimum values differing by 5 orders of magnitude in our dataset (Section IV-B). This poses a huge challenge for Pensieve's single model to predict accurately. Although we have already retrained Pensieve, it still appears not powerful enough to specialize in all scenarios as reported in [27].

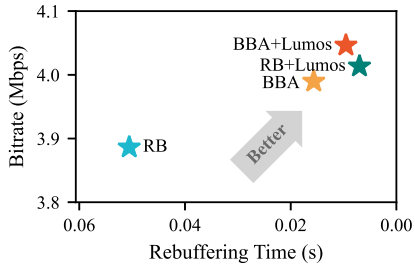


Fig. 22. QoE Improvement of RB+Lumos and BBA+Lumos.

D. Lumos With Other ABR Algorithms

Finally, we evaluate the QoE performance of RB+Lumos and BBA+Lumos. Since RB and BBA do not 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 presented in Fig. 22 and confirm that with Lumos's assistance, both RB and BBA can obtain better QoE.

RB: RB+Lumos improves average bitrate by 3.3% and reduces average rebuffering time by 86.3% compared to RB. Lumos learns from data in different environments to acquire knowledge about what factors inherently change throughput.

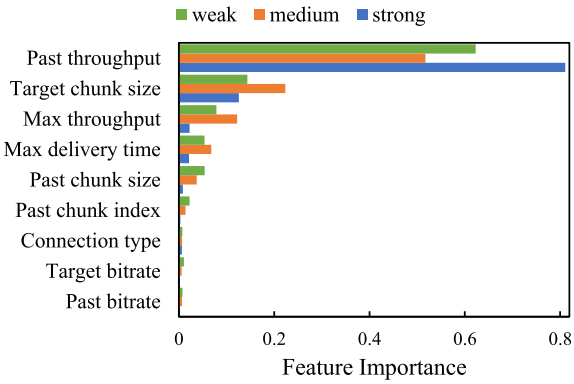


Fig. 23. Feature importance of Lumos's models.

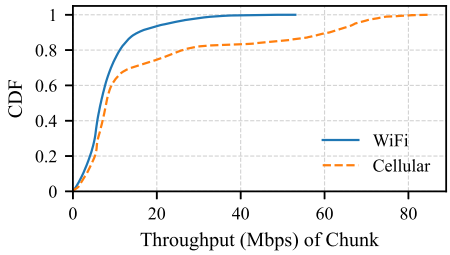
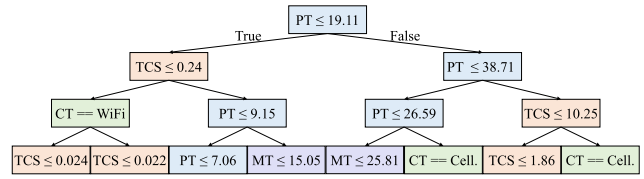
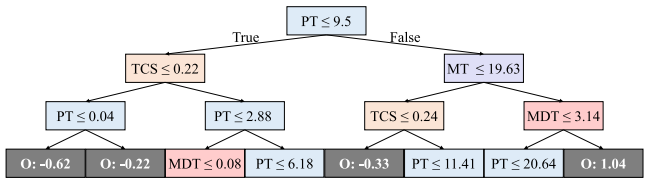


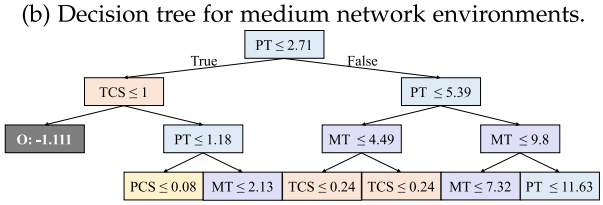
Fig. 24. Throughput under different connection types.



(a) Decision tree for strong network environments.



(b) Decision tree for medium network environments.



(c) Decision tree for weak network environments.

Fig. 25. Top four layers of Lumos decision trees for three network environments. PT: Past throughput (Mbps); MT: Maximum throughput (Mbps); TCS: Target chunk size (MB); PCS: Past chunk size (MB); MDT: Maximum delivery time (s); CT: Connection type; O: Leaf node output (the power of 10, Mbps). Different colors correspond to different features, except for dark gray which indicates the leaf node.

However, HM cannot provide RB with information on why throughput varies. Besides, RB overlooks the chunk size, resulting in unnecessary rebuffering events or bitrate reduction. By integrating Lumos, RB+Lumos makes more flexible decisions than RB to appropriately deal with the throughput dynamic.

BBA: Compared with BBA, BBA+Lumos improves the two metrics by 1.4% and 37.5%, respectively. Note that since the target buffer level in our platform is only 12 s, we set the lower and upper bound of BBA to 5 s and 10, respectively. Our tests confirm that BBA itself is a simple yet effective ABR algorithm in the real world, as reported in [16]. Specifically, compared to MPC+HM, BBA delivers chunks with slightly higher bitrate and lower rebuffering time on average.[10] Even though, BBA+Lumos still outperforms BBA on both bitrate and rebuffering time. While BBA only considers the current buffer level, Lumos obtains information about the environment from the past and further predicts how it evolves in the future, enabling BBA+Lumos to better balance the trade-off between higher bitrate and less rebuffering time.

E. Lumos Deep Dive

1) Feature Importance: We start with investigating how Lumos uses features to make predictions. To do so, we measure the importance of features using the mean decrease impurity metric, as described in [64] and implemented in sklearn [65]. This metric quantifies the total decrease in node impurity (MSE for regression trees) brought by the feature. To analyze the importance of features, we retrain Lumos models on the same dataset without pruning decision trees. We hope that these results will inspire designers of predictors.

Fig. 23 shows the importance of all features in Lumos models trained with different traces, in descending order of the average value. We observe that Lumos makes its decisions mainly based on features about the network condition (i.e., the throughput of the last chunk, maximum of past throughput, and delivery time) and chunk information (i.e., size of the last and the target chunk). Additionally, the player's state and relative chunk index are more important in weaker networks than in stronger ones. This is because in medium and strong networks, the throughput is always sufficient to deliver chunks within their duration, and the player rarely needs to request chunks continuously.

On the other hand, the remaining features are used less frequently, including the client's connection type and the bitrate of the last and target chunk. These features are correlated with other features that provide more information of the same type. For example, although throughput varies under different connection types of the client (as shown in Fig. 24), decision trees find that other features (such as the throughput of the last chunk) are more useful indicators. The same is true for the bitrate with respect to chunk size. However, since decision trees are built greedily, they cannot fully utilize the information in all features simultaneously. Therefore, these features may be more useful for other data-driven methods such as DNNs (e.g., [16] and [17]).

[10]However, BBA perceives more bitrate switches and lower overall QoE performance than MPC. These results are not presented because they are irrelevant to Lumos.

2) Structure of Decision Trees: Fig. 25 depicts the structure of Lumos's decision trees deployed in dash.js. It can be seen that although Lumos utilizes different features with different thresholds to predict throughput, it mainly relies on past throughput and target chunk size. This result is consistent with our *observation 1* in Section III-C2. Moreover, some features that are less important may still be placed in a high layer of the decision tree, as exemplified by the connection type in Fig. 25(a).

The pruned decision tree models for strong, medium, and weak networks have depths of 11, 9, and 10, respectively, with the number of nodes being 1099, 73, and 69, respectively. The decision tree for strong networks is the most complex probably because strong networks have the largest range of throughput variations, as shown in Fig. 14.

3) Overhead: Lumos is lightweight and performs well in real-world mobile networks, making it practical to run on mobile devices.

Computation Overhead: Lumos accesses nodes in decision trees based on feature values layer by layer to predict the throughput, as shown in Fig. 25. The time spent on each prediction is determined by the number of layers Lumos must access. Even in the worst case, Lumos only searches 11 nodes (i.e., the depth of the strong network's decision tree).

When integrated with ABR algorithms, Lumos generates multiple predictions for each chunk. For instance, RB+Lumos predicts the throughput of the next chunk at all 6 bitrate levels, while MPC+Lumos considers 5 future chunks in its solution space and requires a total of $6^5 = 7776$ predictions to select the bitrate for each chunk. Based on evaluation results on a Windows 10 laptop with an Intel Core i5-8300H 2.30 GHz CPU, RB+Lumos takes less than 1 ms (the finest time granularity that can be measured), while MPC+Lumos takes 13.3 ms on average to obtain prediction values for each chunk, indicating that *Lumos requires only 1.7us to perform each prediction*. Given that the median delivery time of each chunk in our dataset is 1.4 s (Fig. 11), the average prediction time of MPC+Lumos is less than 1% of that. Additionally, if the application is latency-sensitive, MPC+Lumos can use only 6 predictions of the next chunk for the future 5 chunks.

Storage Overhead: Although Lumos's models require additional storage space in the client player, we find that Lumos consumes only a small amount of memory. Specifically, the size of Lumos's three JavaScript models is 48.2 KB, 3.23 KB, and 3.07 KB, respectively, totaling 54.5 KB. This size is smaller than the 2-second chunk at the lowest 300 Kbps bitrate level (75 KB on average). Moreover, compared to the TensorFlow.js-converted model of Pensieve, which has a total size of 443 KB, Lumos's models are 8.1 times smaller.

VI. Discussion and Future Work

A. Improving the Theoretical Framework

The theoretical framework proposed in Section III-A provides an in-depth understanding of the factors that impact the throughput of mobile adaptive streaming, which can guide future studies on network prediction and faithful simulation [68]. Nevertheless, there are several promising directions for future research.

Modeling Mobile Connection Status: Plenty of works have been devoted to characterizing the connection status of the link. Some focus on the server perspective (e.g., utilizing RTT and delivery rate [16]), while others consider the client perspective (e.g., considering TTFB [17]). Some investigate the global network (e.g., ISP [10], or CDN information [15]), while others work on the single endpoint (e.g., wireless status [29], or application throughput evolving [27]). Lumos uses connection type and wireless signal strength as indicators for mobile networks. However, a uniform model for understanding this issue is still lacking. Since connection status heavily influences mobile network conditions, we look forward to an in-depth study on modeling it.

Investigating the Interaction Between Transport Mechanism and Application Behavior: Congestion control algorithms (CCAs) are the essential component of the transport layer. In this work, we analyze both loss-based CCAs (e.g., Reno and Cubic [33]) and path model-based BBR [20],[11] which are widely deployed in production systems. In recent years, more CCAs have emerged as alternatives in large-scale deployments, such as Copa [69] in Facebook [70]. Under this circumstance, although recent work has started to focus on the interaction between CCAs and adaptive streaming [71], it remains unclear how different transport mechanisms affect the application throughput.

B. Broader Application of Lumos

Integrated Into Other ABR Algorithms: We demonstrate the benefits of Lumos's accurate prediction in improving QoE for ABR algorithms including MPC, RB, and BBA. In fact, Lumos can be integrated into any ABR algorithm that relies on network prediction, whether explicitly or implicitly. For example, DYNAMIC [12] involves a rate-based algorithm that requires explicit throughput prediction, where the existing predictor can be replaced by Lumos, as in RB+Lumos. In addition, Pensieve [23] utilizes implicit throughput predictions via 1D convolution layers based on past throughput, delivery time, and target chunk size, and passes the output to deeper neural network layers. To integrate Lumos into Pensieve, we can substitute Pensieve's 1D convolution layers (and corresponding features) with Lumos. Note that Lumos and Pensieve are trained by supervised and reinforcement learning, respectively. Hence, we need to train Lumos's models first, then incorporate them into Pensieve's neural network, and finally retrain Pensieve.

Applied in Other Video Applications: The core idea of Lumos is to consider network conditions, transport mechanism, and application behavior in throughput prediction using lightweight data-driven techniques. Although Lumos is designed based on observations in on-demand streaming, its core idea is applicable to other mobile applications, such as live streaming, 360-degree streaming, and volumetric streaming. However, different applications may rely on different transport mechanisms and exhibit different behaviors. For example, live streaming may have idle

[11] We conducted several tests using BBR, but the chunk throughput was significantly lower than that with Cubic. We opted not to use these results since we are not sure whether this problem originates from the kernel implementation of the cloud server or the BBR mechanism itself.

periods during the download when the video content is unavailable on the server (e.g., not uploaded yet) [72], which differs from the on-demand chunk delivery procedure in Section III-B. In other words, further investigations are required to determine the input features for individual applications.

C. Future Work: Faithful Simulation

Motivated by the observation that chunk size affects video streaming throughput, recent studies start to focus on faithful simulation. For example, CausalSim [68] aims to remove the bias between Peniseve's trace-driven simulator and the real world (as described in Section V-C) by explicitly modeling the effect of the ABR decisions (i.e., different chunk sizes) on perceived throughput. In light of our proposed theoretical framework, we find that other factors that affect throughput, such as CCAs and the ON-OFF period, can also introduce biases in simulation. As removing these biases is promising to further improve simulation accuracy, we plan to investigate this research area in the future.

VII. Related Work

There is a large amount of existing work on ABR video steaming [7], [8], [11], [12], [13], [14], [16], [22], [23], [24], [27], [50], [63], [73], [74], [75], [76]. Most of them rely on predicting throughput or delivery time explicitly or implicitly. However, only a few studies focus on this issue [9], [10], [15], [16], [17], [18], [19].

Among them, [18] and [19] indicate that it is difficult to predict throughput as background traffic exists, but they do not investigate how to achieve better prediction. PiStream [9], CS2P [10], and MPC-CDN [15] propose predictors based on observations in the real world. However, they only focus on factors about connection status and believe that throughput fluctuation is solely brought about by changes in network conditions. Fugu [16] and Xatu [17] consider chunk size in predicting delivery time. Nevertheless, chunk size only represents the amount of the application's delivered data. How the application behaves during delivery and how it interacts with the transport layer also matter.

Unlike the above, our work provides a fundamental understanding of network prediction for mobile adaptive streaming based on theoretical analysis and real-world measurement. We consider the impact of the transport mechanism and application behavior in prediction, bring insight that predicting the throughput is better than the delivery time, and propose a throughput predictor that can be integrated into ABR algorithms and performs well in the real-world mobile Internet.

VIII. Conclusion

Accurate network prediction is crucial for ABR algorithms to improve QoE in mobile video streaming. This paper examines all components in designing a predictor for ABR algorithms, including input features, output target, and mapping function. We construct a theoretical framework to identify features for the predictor and verify this framework through formulation analysis and real-world measurement involving 2500+ sessions collected in wild mobile Internet. Observations indicate that network conditions, transport mechanism, and application behavior determine application throughput. In addition, we find that delivery time follows a long-tailed distribution, which a power law can fit. Therefore, throughput is a better prediction target for data-driven methods than delivery time. Based on the above, we develop Lumos, an accurate throughput predictor for mobile adaptive streaming. Evaluations in real-world mobile networks demonstrate that Lumos assists existing ABR algorithms in achieving better QoE. We further discuss directions worth exploring in our theoretical framework, which we believe will be useful for subsequent studies on network prediction and faithful simulation.

References

[1] G. Lv, Q. Wu, W. Wang, Z. Li, and G. Xie, "Lumos: Towards better video streaming QoE through accurate throughput prediction," in *Proc. IEEE Conf. Comput. Commun.*, 2022, pp. 650–659.

[2] SANDVINE, "2023 global internet phenomena report," 2023. [Online]. Available: https://www.sandvine.com/global-internet-phenomena-report-2023

[3] DASH, "Dash industry forum| catalyzing the adoption of MPEG-DASH," 2023. [Online]. Available: https://dashif.org/

[4] Bitmovin, "Why Youtube and Netflix use MPEG-DASH in HTML5," 2015. [Online]. Available: https://bitmovin.com/mpeg-dash-youtube-netflix-html5/

[5] YouTube, "Delivering live Youtube content via dash," 2023. [Online]. Available: https://developers.google.com/youtube/v3/live/guides/encoding-with-dash

[6] S. Media, "Hulu: DASH is definitely the future for us," 2014. [Online]. Available: https://www.streamingmedia.com/Articles/Editorial/Featured-Articles/Hulu-DASH-Is-Definitely-the-Future-for-Us-97468.aspx

[7] J. Jiang, V. Sekar, and H. Zhang, "Improving fairness, efficiency, and stability in HTTP-based adaptive video streaming with festive," in *Proc. 8th Int. Conf. Emerg. Netw. Experiments Technol.*, 2012, pp. 97–108.

[8] X. Yin, A. Jindal, V. Sekar, and B. Sinopoli, "A control-theoretic approach for dynamic adaptive video streaming over HTTP," in *Proc. ACM Conf. Special Int. Group Data Commun.*, 2015, pp. 325–338.

[9] X. Xie, X. Zhang, S. Kumar, and L. E. Li, "piStream: Physical layer informed adaptive video streaming over LTE," in *Proc. 21st Annu. Int. Conf. Mobile Comput. Netw.*, 2015, pp. 413–425.

[10] Y. Sun et al., "Cs2P: Improving video bitrate selection and adaptation with data-driven throughput prediction," in *Proc. ACM SIGCOMM Conf.*, 2016, pp. 272–285.

[11] Y. Qin et al., "A control theoretic approach to ABR video streaming: A fresh look at PID-based rate adaptation," in *Proc. IEEE INFOCOM Conf. Comput. Commun.*, 2017, pp. 1–9.

[12] K. Spiteri, R. Sitaraman, and D. Sparacio, "From theory to practice: Improving bitrate adaptation in the dash reference player," in *Proc. 9th ACM Multimedia Syst. Conf.*, 2018, pp. 123–137.

[13] Y. Qin et al., "ABR streaming of VBR-encoded videos: Characterization, challenges, and solutions," in *Proc. 14th Int. Conf. Emerg. Netw. EXperiments Technol.*, 2018, pp. 366–378.

[14] H. Mao et al., "Real-world video adaptation with reinforcement learning," 2020, *arXiv:2008.12858*.

[15] E. Ghabashneh and S. Rao, "Exploring the interplay between CDN caching and video streaming performance," in *Proc. IEEE Conf. Comput. Commun.*, 2020, pp. 516–525.

[16] F. Y. Yan et al., "Learning in situ: A randomized experiment in video streaming," in *Proc. 17th USENIX Symp. Networked Syst. Des. Implementation*, 2020, pp. 495–511.

[17] Y. S. Nam et al., "Xatu: Richer neural network based prediction for video streaming," *ACM SIGMETRICS Perform. Eval. Rev.*, vol. 50, no. 1, pp. 9–10, 2022.

[18] S. Akhshabi, L. Anantakrishnan, A. C. Begen, and C. Dovrolis, "What happens when HTTP adaptive streaming players compete for bandwidth?," in *Proc. 22nd Int. Workshop Netw. Operating Syst. Support Digit. Audio Video*, 2012, pp. 9–14.

[19] T.-Y. Huang, N. Handigol, B. Heller, N. McKeown, and R. Johari, "Confused, timid, and unstable: Picking a video streaming rate is hard," in *Proc. Internet Meas. Conf.*, 2012, pp. 225–238.

[20] 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.

[21] B. Wang and F. Ren, "Towards forward-looking online bitrate adaptation for dash," in *Proc. 25th ACM Int. Conf. Multimedia*, 2017, pp. 1122–1129.

[22] 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 *Proc. ACM Conf. SIGCOMM*, 2014, pp. 187–198.

[23] H. Mao, R. Netravali, and M. Alizadeh, "Neural adaptive video streaming with pensieve," in *Proc. Conf. ACM Special Int. Group Data Commun.*, 2017, pp. 197–210.

[24] K. Spiteri, R. Urgaonkar, and R. K. Sitaraman, "Bola: Near-optimal bitrate adaptation for online videos," in *Proc. IEEE 35th Annu. Int. Conf. Comput. Commun.*, 2016, pp. 1–9.

[25] X. K. Zou et al., "Can accurate predictions improve video streaming in cellular networks?," in *Proc. 16th Int. Workshop Mobile Comput. Syst. Appl.*, 2015, pp. 57–62.

[26] M. Licciardello, M. Grüner, and A. Singla, "Understanding video streaming algorithms in the wild," in *Proc. Int. Conf. Passive Act. Netw. Meas.*, 2020, pp. 298–313.

[27] Z. Akhtar et al., "OBOE: Auto-tuning video ABR algorithms to network conditions," in *Proc. Conf. ACM Special Int. Group Data Commun.*, 2018, pp. 44–58.

[28] T. Flach et al., "An internet-wide analysis of traffic policing," in *Proc. ACM SIGCOMM Conf.*, 2016, pp. 468–482.

[29] A. Narayanan et al., "Lumos5G: Mapping and predicting commercial mmwave 5G throughput," in *Proc. ACM Internet Meas. Conf.*, 2020, pp. 176–193.

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

[31] Y.-S. Lim, E. M. Nahum, D. Towsley, and R. J. Gibbens, "ECF: An MPTCP path scheduler to manage heterogeneous paths," in *Proc. 13th Int. Conf. Emerg. Netw. Experiments Technol.*, 2017, pp. 147–159.

[32] J. F. Kurose and K. W. Ross, *Computer Networking: A. Top-Down Approach. 6th*. Harlow, U.K.: Pearson Education Ltd, 2012.

[33] S. Ha, I. Rhee, and L. Xu, "Cubic: A new TCP-friendly high-speed TCP variant," *ACM SIGOPS Operating Syst. Rev.*, vol. 42, no. 5, pp. 64–74, 2008.

[34] Dindustry forum: Dashjs, "A reference client implementation for the playback of MPEG dash via JavaScript and compliant browsers," 2023. [Online]. Available: https://github.com/Dash-Industry-Forum/dash.js/

[35] xiph.org, "Xiph.org: Derf's test media collection," 2023. [Online]. Available: https://media.xiph.org/video/derf

[36] FFmpeg, 2023. [Online]. Available: https://www.ffmpeg.org/

[37] J. Le Feuvre, C. Concolato, and J.-C. Moissinac, "GPAC: Open source multimedia framework," in *Proc. 15th ACM Int. Conf. Multimedia*, 2007, pp. 1009–1012.

[38] Selenium, "Seleniumhq browser automation," 2023. [Online]. Available: https://www.selenium.dev/

[39] A. Narayanan et al., "A variegated look at 5G in the wild: Performance, power, and QoE implications," in *Proc. ACM SIGCOMM Conf.*, 2021, pp. 610–625.

[40] Nginx, "nginx news," 2023. [Online]. Available: http://nginx.org/

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

[42] J. Sheng-Jhih, "pywifi · pypi," 2023. [Online]. Available: https://pypi.org/project/pywifi/

[43] JerseyHo, "Cellular-z," 2023. [Online]. Available: https://play.google.com/store/apps/details?id=make.more.r2d2.cellular_z

[44] Android, "Wifiinfo– Android developers," 2023. [Online]. Available: [Online]. Available: https://developer.android.com/reference/android/net/wifi/WifiInfo#getRssi()

[45] Android, "Cellsignalstrengthlte– Android developers," 2023. [Online]. Available: [Online]. Available: https://developer.android.com/reference/android/telephony/CellSignalStrengthLte#getRsrp()

[46] D. N. Reshef et al., "Detecting novel associations in large data sets," *Science*, vol. 334, no. 6062, pp. 1518–1524, 2011.

[47] A. Kraskov, H. Stögbauer, and P. Grassberger, "Estimating mutual information," *Phys. Rev. E., Stat. Nonlinear, Soft Matter Phys.*, vol. 69, 2004, Art. no. 066138.

[48] minepy, "Python api — minepy 1.2.6 documentation," 2023. [Online]. Available: https://minepy.readthedocs.io/en/latest/python.html?highlight=mic#minepy.MINE.mic

[49] Y. E. Guo, A. Nikravesh, Z. M. Mao, F. Qian, and S. Sen, "Accelerating multipath transport through balanced subflow completion," in *Proc. 23rd Annu. Int. Conf. Mobile Comput. Netw.*, 2017, pp. 141–153.

[50] 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 *Proc. IEEE Conf. Comput. Commun.*, 2020, pp. 1967–1976.

[51] Z. Meng, M. Wang, J. Bai, M. Xu, H. Mao, and H. Hu, "Interpreting deep learning-based networking systems," in *Proc. Annu. Conf. ACM Special Int. Group Data Commun. Appl. technol. Architectures Protoc. Comput. Commun.*, 2020, pp. 154–17.

[52] Z. Meng, J. Chen, Y. Guo, C. Sun, H. Hu, and M. Xu, "PiTree: Practical implementation of ABR algorithms using decision trees," in *Proc. 27th ACM Int. Conf. Multimedia*, 2019, pp. 2431–2439.

[53] scikit learn, "sklearn.ensemble.randomforestregressor — scikit-learn 1.3.0 documentation," 2023. [Online]. Available: [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html#sklearn.ensemble.RandomForestRegressor

[54] H. Zhang et al., "Loki: Improving long tail performance of learning-based real-time video adaptation by fusing rule-based models," in *Proc. 27th Annu. Int. Conf. Mobile Comput. Netw.*, 2021, pp. 775–788.

[55] W. E. Leland, M. S. Taqqu, W. Willinger, and D. V. Wilson, "On the self-similar nature of ethernet traffic," in *Proc Conf. Appl. Technol. Architectures Protoc. Comput. Commun.*, 1993, pp. 183–193.

[56] V. Paxson and S. Floyd, "Wide-area traffic: The failure of poisson modeling," in *Proc. Conf. Commun. Architectures Protoc. Appl.*, 1994, pp. 257–268.

[57] M. Faloutsos, P. Faloutsos, and C. Faloutsos, "On power-law relationships of the internet topology," in *Proc. Conf. Appl. Technol. Architectures Protoc. Comput. Commun.*, 1999, pp. 251–262.

[58] R. Albert, H. Jeong, and A. L. Barabasi, "InterNet: Diameter of the World-Wide Web," *Nature*, vol. 401, pp. 130–131, 1999.

[59] A. Clauset, C. R. Shalizi, and M. E. J. Newman, "Power-law distributions in empirical data," *SIAM Rev.*, vol. 51, pp. 661–703, 2007.

[60] L. A. Adamic, "Zipf, power-laws, and pareto-a ranking tutorial," 2000. [Online]. Available: http://ginger.hpl.hp.com/shl/papers/ranking/ranking.html

[61] M. E. J. Newman, "Power laws, pareto distributions and zipf's law," *Contemporary Phys.*, vol. 46, pp. 323–351, 2004.

[62] B. C. Ross, "Mutual information between discrete and continuous data sets," *PLoS One*, vol. 9, no. 2, 2014, Art. no. e87357.

[63] T. Huang, C. Zhou, R.-X. Zhang, C. Wu, and L. Sun, "Learning tailored adaptive bitrate algorithms to heterogeneous network conditions: A domain-specific priors and meta-reinforcement learning approach," *IEEE J. Sel. Areas Commun.*, vol. 40, no. 8, pp. 2485–2503, Aug. 2022.

[64] L. Breiman, J. Friedman, C. J. Stone, and R. A. Olshen, *Classification and Regression Trees*. Boca Raton, FL, USA: CRC Press, 1984.

[65] F. Pedregosa et al., "Scikit-learn: Machine learning in Python," *J. Mach. Learn. Res.*, vol. 12, pp. 2825–2830, 2011.

[66] Pensieve, "Neural adaptive video streaming with pensieve (sigcomm '17)," 2023. [Online]. Available: https://github.com/hongzimao/pensieve

[67] TensorFlow.js, "Machine learning for javascript developers," 2023. [Online]. Available: https://www.tensorflow.org/js

[68] A. Alomar, P. Hamadanian, A. Nasr-Esfahany, A. Agarwal, M. Alizadeh, and D. Shah, "{CausalSim }: A causal framework for unbiased { Trace-Driven} simulation," in *Proc. 20th USENIX Symp. Networked Syst. Des. Implementation*, 2023, pp. 1115–1147.

[69] V. Arun and H. Balakrishnan, "COPA: Practical Delay-Based congestion control for the internet," in *Proc. 15th USENIX Symp. Networked Syst. Des. Implementation*, 2018, pp. 329–342.

[70] Facebook, "Evaluating COPA congestion control for improved video performance," 2019. [Online]. Available: https://engineering.fb.com/2019/11/17/video-engineering/copa/

[71] S. Vargas, R. Drucker, A. Renganathan, A. Balasubramanian, and A. Gandhi, "BBR bufferbloat in dash video," in *Proc. Web Conf.*2021, pp. 329–341.

[72] A. Bentaleb, C. Timmerer, A. C. Begen, and R. Zimmermann, "Bandwidth prediction in low-latency chunked streaming," in *Proc. 29th ACM Workshop Netw. Operating Syst. Support Digit. Audio Video*, 2019, pp. 7–13.

[73] 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 *Proc. IEEE Conf. Comput. Commun.*, 2017, pp. 1–9.

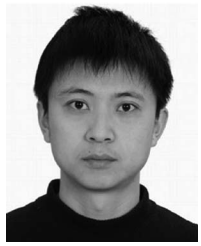
[74] M. Palmer, M. Appel, K. Spiteri, B. Chandrasekaran, A. Feldmann, and R. K. Sitaraman, "VOXEL: Cross-layer optimization for video streaming with imperfect transmission," in *Proc. 17th Int. Conf. Emerg. Netw. EXperiments Technol.*, 2021, pp. 359–374.
[75] X. Zuo, J. Yang, M. Wang, and Y. Cui, "Adaptive bitrate with user-level QoE preference for video streaming," in *Proc. IEEE Conf. Comput. Commun.*, 2022, pp. 1279–1288.
[76] H. Tianchi, Z. Chao, Z. Rui-Xiao, W. Chenglei, and S. Lifeng, "Buffer awareness neural adaptive video streaming for avoiding extra buffer consumption," in *Proc. IEEE Conf. Comput. Commun.*, 2023, pp. 1–9.

Gerui Lv received the BS degree in computer science from Hunan University, Changsha, China, in 2016. He is currently working toward the PhD degree with the Institute of Computing Technology (ICT), Chinese Academy of Sciences (CAS). His research interests lie in adaptive streaming, network transport protocol, and Internet measurements.

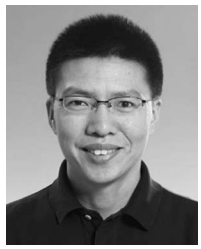
Qinghua Wu received the PhD degree from ICT/CAS, in 2015. He is currently an associate researcher with ICT/CAS. His research interests lie in network transport protocol and Internet measurements.

Qingyue Tan is currently working toward the MS degree with the Institute of Computing Technology (ICT), Chinese Academy of Sciences (CAS). Her research interests lie in adaptive streaming and Internet measurements.

Weiran Wang received the MS degree from ICT/CAS, in 2021. Her research interests include adaptive streaming and Internet measurement.



Zhenyu Li (Member, IEEE) received the BS degree from Nankai University, in 2003 and the PhD degree in Graduate School of Chinese Academy of Sciences (CAS), in 2009. He is a professor with the Institute of Computing Technology, CAS. His research interests include Internet measurement and Networked Systems.



Gaogang Xie received the PhD degree in computer science from Hunan University, in 2002. He is a professor in the Computer Network Information Center, Chinese Academy of Sciences. His research interests include Internet architecture, SDN/NFV, and Internet measurement.