# Enhanced Bandwidth Measurement and Robust Rate Adaptation for Low-latency Live Streaming

***Abstract*—Low latency live streaming (LLLS) like LL-DASH has significantly reduced the end-to-end latency via chunked transfer encoding (CTE). However, LLLS also comes with more challenges for adaptive bitrate (ABR) algorithms: (1) bandwidth measurement is non-trivial and inaccurate due to the possible idle time between chunks in CTE; (2) the various uncertainty in LLLS such as fluctuating segment size further lead to inaccurate buffer estimation, severely degrading ABR's performance. In this paper we propose AAR consisting of two modules: (1) accurate bandwidth measurement with Flag from the server that represents the burst chunks within a segment, which allows for more consecutive valid HTTP chunks; (2) an LLLS tailored ABR with a novel robust objective that maximizes the minimum quality of experience (QoE) brought by the uncertainty. To obtain the minimum QoE, we propose a theorem based on the upper bound of download time estimation, which is backed up by theoretical guarantees. To derive the maximum QoE, we propose a new LLLS state evolution mechanism and apply Model Predictive Controller (MPC) to search for optimal bitrate. Extensive real-world experiments demonstrate that AAR outperforms existing baselines with 10%-80% measurement error reduction, and QoE improves by 39%-104% throughout all considered network conditions.**

***Index Terms*—Live video streaming, Bandwidth measurement, Adaptive streaming**

## I. Introduction

Video streaming has seen significant growth over the years, it is reported by cisco [1] that video traffic contributes more than 70% of the global Internet traffic, where live video makes up 17%. Therefore it motivates us to develop better adaptive bitrate (ABR) algorithms to deliver users better quality of experience (QoE). Different from traditional video-on-demand (VOD) streaming, low latency live streaming (LLLS) [2] presents more challenges because the video segment can be delivered only after it's been captured and rendered from the ingest side. Therefore the total latency is at least the duration of a video segment (about 4 seconds in VOD scenario), and it can increase from rebuffering events which is more likely to happen due to smaller buffer constraints to ensure low latency.

To address this issue, researchers propose to use MPEG Common Media Application Format (CMAF) [3] coupled with HTTP/1.1 chunked transfer encoding (CTE) [4], for example, in LL-DASH [5]. The ingest side consistently produces captured frames and uploads to the HTTP server in the unit of CMAF chunk. When the client requests for the latest segment, the server can directly deliver the cached CMAF chunks to the client via basic HTTP chunks, instead of waiting for the full segment's arrival. In this way, the end-to-end latency can be greatly reduced from about 10-30 seconds to 1-5 seconds.

However, such LLLS implementation grapples with two critical issues: (1) inaccurate bandwidth measurement. On the one hand, there may exist idle time between CMAF chunks and by extension the corresponding HTTP chunks, therefore the client's perceived download time is longer than the actual network transmission time. On the other hand, we cannot obtain the sending time of each HTTP chunk due to the physical time gap. Therefore we can only analyze the HTTP chunks' arrival time to deduce the bandwidth. Although there have been some solutions trying to identify actual download time like [6] and [7], they either neglect such idle time or apply inappropriate filtering algorithms, rendering them far from being accurate, and in turn greatly degrading ABR's performance; (2) various uncertainty in LLLS. Apart from inaccurate bandwidth, the client also cannot fetch the exact next chunk size which is available only in VOD streaming, yet the estimated chunk size from prefixed bitrates can deviate with up to 50% relative error. Moreover, the idle time from CMAF chunks and player's fetch logic further add up the difficulty to accurately predict the download time and buffer evolution, leading to inferior bitrate choices.

In this paper, we propose the AAR framework designed for LLLS. Specifically, to address issue (1), we first identify the root cause for poor measurement as lacking CMAF chunk-sending pattern within a segment. Therefore, we propose to attach a Flag parameter within the HTTP header from the server, indicating the number of burst CMAF chunks and that the rest of the chunks may contain idle time. In this way, we can accumulate the consecutive valid HTTP chunks, and obtain more accurate bandwidth via size weighted average to smooth the smaller CMAF chunks' bandwidth deviation. Moreover, we also propose an improved CMAF boundary identification patch in the latest official dash.js to further guarantee accuracy. To cope with issue (2), we propose a novel max-min objective for ABR to guarantee the lower bound of QoE brought by the various LLLS uncertainty. To derive the min solution, we propose a theorem that by applying the upper bound of download time, we can model the comprehensive uncertainty including idle time, segment size, and future bandwidth, thus estimating the worst LLLS-QoE. We also provide theoretical guarantees that our objective falls back to regular maximum QoE when such uncertainty is eliminated, thanks to our careful upper bound estimation. To derive the max solution, we further propose a new LLLS model such as the buffer and latency evolution, with which we apply Model Predictive Controller (MPC) to combine our novel objective in search of the optimal bitrates.

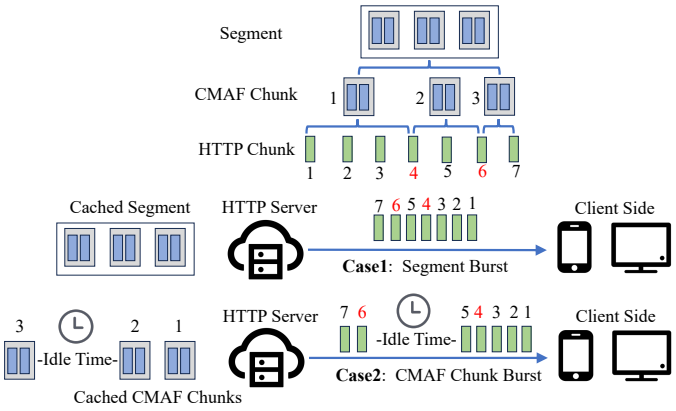



Fig. 1: Idle time between CMAF chunks and HTTP chunks.

Finally, we conduct extensive experiments in real world, comparing 7 bandwidth measurement baselines and 6 representative ABRs with the latest dash.js throughout four network trace datasets. The results demonstrate that AAR outperforms existing methods with 10%-80% bandwidth error reduction and 39%-104% QoE improvement. The ablation study also validates the efficiency of each module of AAR.

**Contributions**. We summarize our contributions as follows:

• We carry out a comprehensive analysis from off-the-shelf baselines to reveal the two critical issues in LLLS, i.e. the inaccurate bandwidth measurement due to chunk idle time, and inferior ABR performance due to various uncertainty.

• To tackle the two issues, we propose the AAR framework consisting of two modules. On the one hand, we propose to leverage the burst CMAF chunk pattern to accumulate more valid samples for accurate measurement via robust size weighted averaging, and we also propose an improved CMAF boundary identification patch to gain better accuracy. On the other hand, we propose a novel objective for ABR to maximize the minimum QoE caused by the uncertainty. To address the min optimization, we propose a practical theorem based on the upper bound of download time, followed by theoretical guarantees for our adaption when uncertainty diminishes. To derive the max solution, we first propose an improved LLLS evolution model and apply MPC to obtain the optimal bitrates.

• Extensive real world experiments throughout 4 network trace datasets demonstrate that AAR outperforms existing 7 measurement baselines and 6 representative ABRs.

## II. Background and Motivation

### A. Bandwidth Measurement in LLLS

To meet the strict low latency requirement in LLLS, researchers propose to combine CMAF and CTE, where the segment duration is shorter and the transmission unit is the smaller CMAF chunk instead of the whole segment, greatly reducing the total latency. However, it also brings challenges for accurate bandwidth measurement. The perceived download time is no longer accurate because there may exist idle time between two CMAF chunks and by extension, the corresponding HTTP chunks.

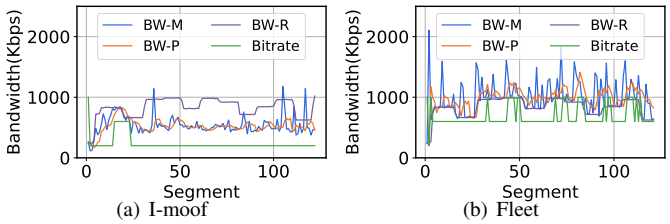
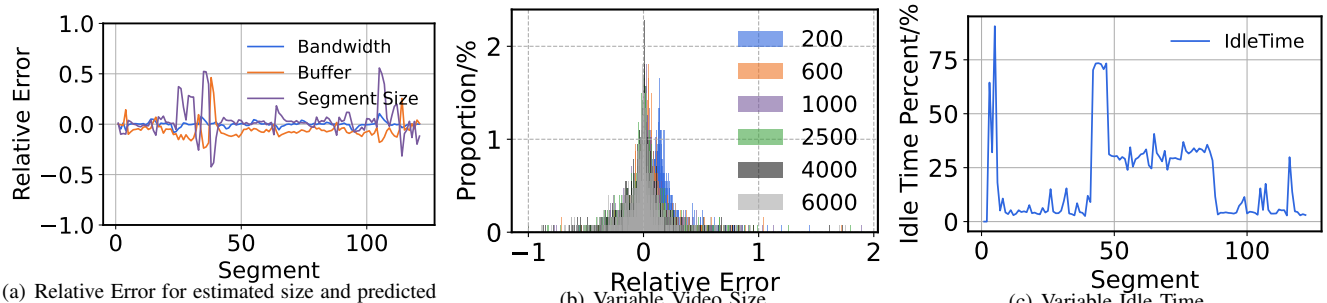



Fig. 2: Examples of existing bandwidth measurements' impact on fixed ABR's performance.

We illustrate the reason in Fig. 1, assuming the segment is composed of three CMAF chunks, each divided into several small HTTP chunks, where No.4 and No.6 contain parts of adjacent CAMF chunks. Case 1 represents the catch-up phase where the requested segment is behind the ingest side, therefore the whole segment is delivered without pause. While in case 2, the third CMAF chunk arrives at the server after some idle time, and the first two are sent out in 5 consecutive HTTP chunks, followed by the later 6-7 HTTP chunks, where No.6 carries idle time and No.4 does not. Therefore from the client side there's no way to distinguish whether certain HTTP chunk contains idle time.

Accordingly, researchers have proposed countermeasures to eliminate such idle time to derive the actual bandwidth [6]–[9]. LoL+ [6] proposes to store the start and end time of a CMAF chunk, i.e. the arrival time of the HTTP chunk that contains $moof$ and $mdat$ boxes respectively. And then it computes the average bandwidth of all CMAF chunks. However, as indicated in Fig. 1 case 2, the last HTTP chunk No.6 still carries idle time, which leads to overestimation of the end time of CMAF chunk 2. Moreover, the first CMAF chunk of a segment usually contains I frame and can make up 50% size of the segment, thus averaging on all CMAF chunks can introduce noises instead. To tackle the issue, Fleet [7] further proposes to neglect the last HTTP chunk to prevent such idle time, and it stores all the valid consecutive HTTP chunks to average bandwidth. However, some valid HTTP chunks can be filtered out like No.4 in Fig. 1, and the HTTP chunks are usually about 1500 Bytes, whose download time can be easily disturbed by random noises.

To verify our analysis, we conduct real world experiments with rate-based ABR that picks the highest bitrate under the available bandwidth, refer to Section IV-A for detailed setup. The results are in Fig. 2, where BW-M, P and R represent the measured, the predicted and the real bandwidth respectively. We can find that I-moof (from LoL+ [6]) in (a) does present relatively lower measurement due to the idle time in the last HTTP chunk, and the sudden deviation comes from the small chunks disturbance. As a result, ABR constantly selects the lowest bitrate (200Kbps) and wastes the bandwidth. In contrast, Fleet in (b) almost always overestimates because a single HTTP chunk's download time can deviate several milliseconds, causing totally different bandwidth result. Therefore, ABR sometimes picks bitrate (1000Kbps) higher than the




(a) Relative Error for estimated size and predicted buffer evolution

(b) Variable Video Size

(c) Variable Idle Time

Fig. 3: RMPC's buffer prediction error due to varying size and idle time. We directly use the bitrates and duration to estimate the segment size.

available bandwidth (e.g. around segment 75 and 120), which in turn causes the player to stall and degrades the QoE.

• **Insight 1:** We need to identify cached CMAF chunks from the server to accumulate the first consecutive HTTP chunks without idle time, and also mitigate the noise deviation from the rest of the small chunks.

### *B. ABR in LLLS*

Typical ABRs are designed for VOD scenario, like the heuristic method RobustMPC (RMPC) [10] that estimates the bandwidth lower bound and iterates all bitrates combination based on the evolution model. LLLS ABRs are more sophisticated in terms of low latency. Representative method LoL+ [6] leverages self-organizing maps (SOM) to learn the bitrate adaption with a new heuristic method to optimize the playback speed and coordinate latency. However, despite different designs, existing ABRs still operate in VOD pattern that only considers the uncertainty from bandwidth, while LLLS brings several new. In addition to the inaccurate measured and predicted bandwidth, LLLS also includes the unavailable segment size and comprehensive idle time which consists of many aspects, such as round trip time (RTT), server's pending for CMAF chunks, player's pause before fetching (e.g. scheduling in dash.js), etc. Without proper estimation for the varying size and idle time, ABR cannot establish accurate modeling and search for optimal bitrates, even with accurate bandwidth alone.

To prove our concern, we conduct experiments with RMPC [10] to investigate the uncertainty impact from segment size and idle time in Fig. 3. Note that we fix the bandwidth and ABR to control variables. Fig. 3 (a) illustrates the relative error for estimated segment size and final buffer evolution with fixed bandwidth. When the estimated size appears lower, RMPC tends to underestimate the download time and overestimate the next buffer, which is very likely followed by a stalling event due to smaller buffer in LLLS. In detail, Fig. 3 (b) presents the size deviation PDF of a 10 minutes video encoded at different bitrates (Kbps), which validates that video size can even be twice the estimated value. In addition, the possible idle time from both the server and client also contributes to the inaccurate buffer. Fig. 3 (c) reveals the total idle time percentage of the total segment download time, which almost appears random due to comprehensive factors and severely falsifies the final buffer prediction. More importantly, a small mismatch in buffer prediction leads to obvious QoE degradation (refer to Table V in ablation study), e.g. either wasting bandwidth or causing stalling.

• **Insight 2:** We need to model the comprehensive uncertainty in LLLS and derive a robust objective for ABR to guarantee the lower bound of QoE.

## III. Proposed Method: AAR

### *A. Overview of AAR*

We first summarize the frequently used notations in Table I. To address the two issues in LLLS, i.e. inaccurate bandwidth measurement and various LLLS uncertainty, we propose the AAR (Accurate And Robust) framework and present the overview in Fig. 4. The workflow is as follows: The player first requests for $seg_n$, and the HTTP server then bursts out the cached CMAF chunks ($cc_{n,j}$) via HTTP chunks ($hc_{n,z}$). AAR performs bandwidth measurement (Section III-B) with Flag parameter upon receiving all the HTTP chunks, and then delivers the predicted future bandwidth via typical smooth average to the player. Meanwhile, AAR's ABR (Section III-C) receives current states from the player such as buffer and latency. Based on the novel max-min objective, AAR derives the min solution backed up by theoretical guarantees, and then combines our novel LLLS modeling with MPC to obtain optimal bitrates. Finally, the player requests for a new segment and this procedure continues.

### *B. Flag-based Bandwidth Measurement*

**Burst Chunk Measurement.** Based on insight 1, we need to identify the CMAF chunk sending pattern to accumulate consecutive HTTP chunks. However this information can be only acquired from the server side, which knows exactly how many CMAF chunks are cached for burst. Therefore, we propose to attach a Flag=$k_i$ parameter for $seg_i$ in CTE's header, along with the following HTTP chunks, representing the number of burst CMAF chunks. In this way, we can ensure that the first $k_i$ CMAF chunks are sent without pause, and by extension, the HTTP chunks. In contrast, the later CMAF

TABLE I: Summary of notations.

| Category | Notation | Meaning | Category | Notation | Meaning |
|---|---|---|---|---|---|
| Measurement | $N$ | number of segments | ABR | $R_i$ | bitrate for $seg_i$ |
| | $K$ | number of CMAF chunks in a segment | | $\alpha_k$ | weights for QoE, $k \in [1,5]$ |
| | $Z_i$ | number of HTTP chunks in segment $i$ | | $s_i/s_{i,j}$ | size of $seg_i/cc_{i,j}$ |
| | $seg_i$ | segment $i, i \in [1,N]$ | | $u_{i,j}$ | idle time before downloading $cc_{i,j}$ |
| | $cc_{i,j}$ | CMAF chunk $j$ of $seg_i, j \in [1,K]$ | | $c_i$ | availabel bandwidth for $seg_i$ |
| | $hc_{i,z}$ | HTTP chunk $z$ of $seg_i, z \in [1,Z_i]$ | | $pc_i$ | predicted bandwidth for $seg_i$ |
| | $t_{i,z}$ | arrival time of $hc_{i,z}$ of $seg_i$ | | $d_i/d_{i,j}$ | estimated download time of $seg_i/cc_{i,j}$ |
| | $h_{i,z}$ | size of $hc_{i,z}$ of $seg_i$ | | $d_i^*/d_{i,j}^*$ | actual download time of $seg_i/cc_{i,j}$ |
| | $k_i$ | Flag for $seg_i$ | | $d_i^u/d_{i,j}^u$ | upper bound of download time deviation of $seg_i/cc_{i,j}$ |
| | | | | $b_{i,j}$ | buffer after downloading $cc_{i,j}$ |
| | | | | $T$ | CMAF chunk duration $cc_{i,j}$ |
| | | | | $l_i/l_{i,j}$ | latency after downloading $seg_i/cc_{i,j}$ |
| | | | | $r_i/r_{i,j}$ | rebuffer time after downloading $seg_i/cc_{i,j}$ |
| | | | | $p_i/p_{i,j}$ | playback rate after downloading $seg_i/cc_{i,j}$ |

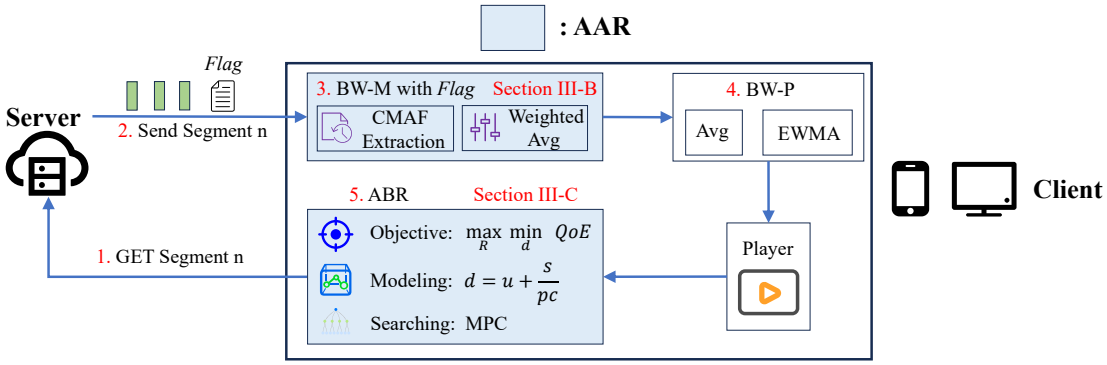



Fig. 4: Overview of the AAR framework. BW-M and BW-P represent bandwidth measurement and prediction, respectively.

chunks may contain idle time and bandwidth computation must operate separately. To reduce the noise deviation, we propose to leverage size weighted average to guarantee the major bandwidth from the $k_i$ accumulated CMAF chunks, which contain I frame that makes up 50% of the segment size.

**CMAF identification.** Regarding the extraction of CMAF chunk from HTTP chunks, we propose an improved algorithm different from LoL+ introduced in Section II-A, where a CMAF chunk is checked only once in an HTTP chunk, we need to keep parsing the same $hc_{i,z}$ in loop in case a single HTTP chunk contains multiple CMAF chunks (refer to Fig. 1). In addition, we identify and fix some patches in $offset$ setting during the $moof + mdat$ boxes search in the latest dash.js.

**Implementation.** The detailed procedure of AAR's bandwidth measurement is in Algorithm 1. First of all, we directly compute the segment bandwidth as done VOD if $k_i$ is exactly $K$ (line 1). Then we initialize $bw$ to store bandwidth samples. The next key step is to identify the CMAF boundary from $hc_{i,z}$ via our improved searching algorithm (line 3), the resulting $T^s$ and $T^e$ store the start and end number of HTTP chunk for each CMAF chunk, followed by our first and foremost bandwidth sample that considers all the $hc_{i,z}$ for the burst $k_i$ $cc_{i,j}$ (line 5). Note that we need to remove the last $hc$ if it contains the $moof$ box of the next $cc$ ($T^s_{k_i+1} == T^e_{k_i}$ in line 4). Moreover, we also compute the bandwidth of smaller CMAF chunks (lines 6-8) to derive the final accurate bandwidth weighted by chunk size (line 10) to ensure the priority of our burst chunks' bandwidth.

**Algorithm 1:** AAR's bandwidth measurement

**Input:** Flag=$k_i$, received all HTTP chunks $hc_{i,z}$, fetch request time $t_{req}$
**Output:** $c_i$

1 **if** $k_i == K$ **then return** $\frac{s_i - h_{i,1}}{t_{i,Z_i} - t_{i,1}}$;
2 $bw \leftarrow []$;
3 $T^s, T^e \leftarrow extract(hc_{i,z})$; ▷ CMAF extraction
4 $end \leftarrow T^s_{k_i+1} == T^e_{k_i} ?\ T^e_{k_i} - 1 :\ T^e_{k_i}$; ▷ Burst chunks' last HTTP chunk
5 $bw$.append($[\frac{\sum_{z=2}^{end} h_{i,z}}{t_{i,end} - t_{i,1}}, \sum_{z=2}^{end} h_{i,z}]$); ▷ [bandwidth, size]
6 **for** $j \leftarrow k_i + 1$ *to* $K$ **do**
7 $\quad end \leftarrow T^s_{j+1} == T^e_j ?\ T^e_j - 1 :\ T^e_j$;
8 $\quad bw$.append($[\frac{\sum_{z=T^s_j+1}^{end} h_{i,z}}{t_{i,end} - t_{i,T^s_j}}, \sum_{z=T^s_j+1}^{end} h_{i,z}]$);
9 **end**
10 $c_i \leftarrow \frac{\sum bw[:,0] \times bw[:,1]}{\sum bw[:,1]}$; ▷ Size weighted averaging
11 **return** $c_i$

### C. LLLS tailored ABR

**Max-min Objective.** Based on insight 2, we need to design a robust objective to include all possible uncertainty in LLLS. Following LoL+ [6] and QoE standard in [11]. The typical objective is to decide the bitrates of $N$ segments to ensure high

video quality $R_i$, low video rebuffering time $r_i$, low latency $l_i$, normal playback rate $p_i$ and low bitrate switches $|R_i - R_{i-1}|$, formulated as follows in Equ. 1:

$$QoE = \sum_{i=1}^{N} (\alpha_1 R_i - \alpha_2 r_i - \alpha_3 l_i - \alpha_4 |p_i - 1|) - \sum_{i=2}^{N} \alpha_5 |R_i - R_{i-1}| \tag{1}$$

where $\alpha_k > 0, k \in [1, 5]$ is the corresponding weight for each metric. According to Section II-B and Fig. 3, the uncertainty comes from idle time $u_{i,j}$, varying segment size $s_i$ and inaccurate future bandwidth $pc_i$. The intuitive solution is to maximize the worst QoE brought by the three uncertain variables, formulated as follows:

$$\max_{R_i} \min_{\{u_{i,j}, s_i, pc_i\}} QoE \tag{2}$$

**Evolution Model.** However, it remains unclear how they impact the specific metrics in QoE, let alone the inner objective solution. To derive detailed QoE impact, we first propose a new state evolution model tailored to LLLS to replace the existing segment-based ones:

$$d_{i,j} = u_{i,j} + \frac{s_{i,j}}{pc_i} \tag{3}$$

$$b_{i,j} = max(b^-_{i,j} - p^-_{i,j} \times d_{i,j}, 0) + T \tag{4}$$

$$r_{i,j} = max(d_{i,j} - \frac{b^-_{i,j}}{p^-_{i,j}}, 0) \tag{5}$$

$$l_{i,j} = l^-_{i,j} - (p^-_{i,j} - 1) \times min(d_{i,j}, \frac{b^-_{i,j}}{p^-_{i,j}}) + r_{i,j} \tag{6}$$

$$p_{i,j} = \begin{cases} f(l_{i,j}) > 1, & \text{if } l_{i,j} - l_{target} > \delta \\ 1, & \text{if } |l_{i,j} - l_{target}| < \delta \\ f(l_{i,j}) < 1, & \text{if } l_{i,j} - l_{target} < -\delta \end{cases} \tag{7}$$

where $\delta$ is the prefixed threshold and $x_{i,j}, x \in \{b, p, l\}$ is the attribute for the $j^{th}$ CMAF chunk of $seg_i$, $x^-_{i,j} = x_{i,j-1}$ if $j > 1$ and $x_{i-1,K}$ if not. In LLLS, $r_i = \sum_{j=1}^{K} r_{i,j}$, $s_i = \sum_{j=1}^{K} s_{i,j}$ and $l_i/p_i$ is the final latency/speed after downloading all the CMAF chunks. Specifically, the key variable is the download time $d_{i,j}$ which includes idle time $u_{i,j}$ and the actual transmission time $\frac{s_{i,j}}{pc_i}$, where $s_{i,j}$ is the estimated CMAF chunk size using bitrates. With $d_{i,j}$, we can predict the buffer $b_{i,j}$ change by draining $p^-_{i,j} \times d_{i,j}$ seconds of cached video and appending a new CMAF chunk duration $T$, along with possible stalling $r_{i,j}$.

As for the latency evolution Equ. 6, it first catches up or loses behind at speed $p^-_{i,j} > 1$ or $p^-_{i,j} < 1$. The magnitude is up to the download time $d_{i,j}$ if there is no rebuffering, otherwise up to $\frac{b^-_{i,j}}{p^-_{i,j}}$, because there's no more content to playback after that. The rest of the increased latency solely depends on the stalling time. In comparison, current work estimates in the **WRONG** way that only considers the $d_{i,j}$ such as Tightrope [12] and Fleet's ABR. [7]. Regarding the playback speed $p^-_{i,j}$, it depends on current latency deviation around the threshold $\delta$ of target latency $l_{target}$, $f$ function maps latency to speed.

**Min Solution.** With our novel LLLS model, we can transform the objective 2 into the following:

$$\max_{R_i} \min_{d_{i,j}} QoE \tag{8}$$

The reasons are twofold. On the one hand, we can observe that the three variables $u_{i,j}$, $s_i$ and $pc_i$ are only related to the download time $d_{i,j}$, optimizing one variable decreases the overall complexity. On the other hand, the min objective is only related to $d_{i,j}$ because rebuffering $r_i$ and latency $l_i$ are the only ones that include the uncertain download time, and by extension the three variables in objective 2. Specifically, we can divide the QoE into two parts according to the related variables as follows:

$$\begin{aligned} &\max_{R_i} \sum_{i=1}^{N} \min_{d_{i,j} \in [0, d^u_{i,j}]} [g_1(R_i, p_i) + g_2(d_{i,j})] = \\ &\max_{R_i} \sum_{i=1}^{N} g_1(R_i, p_i) + \min_{d_{i,j} \in [0, d^u_{i,j}]} g_2(d_{i,j}) \end{aligned} \tag{9}$$

where $g_1(R_i, p_i) = \alpha_1 R_i - \alpha_4 |p_i - 1| - \alpha_5 |R_i - R_{i-1}|$ and $g_2(d_{i,j}) = -\alpha_2 r_i - \alpha_3 l_i$. The key to the min solution is to identify the $d_{i,j}$ that maximizes the rebuffering and latency. Note that we **Do Not** tend to accurately predict the actual download time like Fugu [13] in VOD because it requires the exact next chunk size in the first place. The solution is not intuitive since latency $l_{i,j}$ is not always $\propto d_{i,j}$ while rebuffering $r_{i,j}$ is. To this end, we propose Theorem 1.

***Assumption** 1:* Assume that $d^u_{i,j} \geq T$, meaning that each CMAF chunk's download time theoretically has at least $T$ idle time from the ingest side waiting for the captured frames.

***Theorem** 1:* Let Assumption 1 hold, the min solution in Equ. 9 is when and only when for each $d_{i,j} = d^u_{i,j}, i \in [1, N], j \in [1, K]$.

***Proof 1 (Theorem 1):*** First of all,

$$\begin{aligned} g_2(d_{i,j}) &= -\alpha_2 \sum_{j=1}^{K} r_{i,j} - \alpha_3 l_i \\ &= -\alpha_2 \sum_{j=1}^{K} max(d_{i,j} - \frac{b^-_{i,j}}{p^-_{i,j}}, 0) - \alpha_3 l_i \end{aligned} \tag{10}$$

Therefore $r_{i,j} \propto d_{i,j}$. As for latency:

$$l_{i,j} = \begin{cases} l^-_{i,j} - (p^-_{i,j} - 1) \times \frac{b^-_{i,j}}{p^-_{i,j}}, & \text{if } d_{i,j} < \frac{b^-_{i,j}}{p^-_{i,j}} \\ l^-_{i,j} + d_{i,j} - b^-_{i,j}, & \text{if } d_{i,j} \geq \frac{b^-_{i,j}}{p^-_{i,j}} \end{cases} \tag{11}$$

Therefore $l_{i,j} \propto d_{i,j}$ except when $p^-_{i,j} > 1$ and $d_{i,j} < \frac{b^-_{i,j}}{p^-_{i,j}}$ (denoted as case P). However, this is when $l^-_{i,j} - l_{target} > \delta$ and in order for this to happen, $l_{i,j}$ would first go through $|l^-_{i,j} - l_{target}| \leq \delta$ and $p^-_{i,j} = 1$, therefore $l_{i,j} = r_{i,j}$. This means there must have been rebuffering events for the first time $p^-_{i,j} > 1$.

**Algorithm 2:** AAR's LLLS tailored ABR

**Input:** estimated player idle time $u_{client}$, past CMAF chunk size $s^*_{i,j}$ and idle time $u^*_{i,j}, j \in [1, K]$, future bandwidth $pc$, future horizon $N_h$, past $d^*_j, j \in [i-4, i]$

**Output:** $R_{i+1}$

1 $\Delta d = \frac{\sum_{j=i-4}^{i} |d^*_j - d_j|}{5}$;
2 **for** *each bitrate combination* **do**
3  **for** $k \leftarrow i+1$ *to* $i+N_h$ **do**
4  **for** $j \leftarrow 1$ *to* $K$ **do**
5  $u_{k,j} \leftarrow u^*_{i,j}$;
6  **if** $j == 1$ **then** $u_{k,j} \leftarrow u_{k,j} + u_{client}$;
7  $s_{k,j} \leftarrow \frac{R_k \times K \times T \times s^*_{i,j}}{s^*_i}$;
8  $d_{k,j} \leftarrow u_{k,j} + \frac{s_{k,j}}{pc}$;
9  $d^u_{k,j} \leftarrow d_{k,j} + \frac{|\Delta d| \times d_{k,j}}{d_k}$;
10  Other variable evolution in Equ. 4-7;
11  **end**
12  **end**
13  Compute QoE as in Equ. 9 with Theorem 1;
14 **end**
15 $R_{i+1} \leftarrow argmax(QoE)$;
16 **return** $R_{i+1}$

Therefore for the first time in case P, $b^-_{i,j} = T$ and $p^-_{i,j} > 1$, then we have $\frac{b^-_{i,j}}{p^-_{i,j}} < b^-_{i,j} = T$, and that $l_{i,j}$ first decreases and then increases as $d_{i,j}$ increases, $d_{i,j} \in [0, d^u_{i,j}]$. Based on Assumption 1, $l_{i,j}(d^u_{i,j}) \geq l_{i,j}(T) = l^-_{i,j} = l_{i,j}(0)$. Therefore we can still set $d_{i,j} = d^u_{i,j}$ to get the maximum latency $l_{i,j}$. Moreover, after this operation, $r_{i,j} = d^u_{i,j} - \frac{b^-_{i,j}}{p^-_{i,j}} > 0$, which means next time $b_{i,j} = T$, and this process continues.

In summary, we can always set $d_{i,j} = d^u_{i,j}$ to derive the maximum $l_i$, $r_i$ and the minimum $g_2(d_{i,j})$. ■

**$d^u_{i,j}$ Estimation.** To put the theorem in practice, we collect the past actual segment download time $d^*_i$ from the player feedback to compute the maximum estimated download time deviation. Specifically, we propose to store historical segment level d-error as $\Delta d_i = |d^*_i - d_i|$, then we distribute such $\Delta d_i$ by the ratio of the estimated $d_{i,j}$ over $d_i$ to obtain the upper bound $d^u_{i,j} = d_{i,j} + \frac{|\Delta d_i| \times d_{i,j}}{d_i}$. Note that we do not store $\Delta d_{i,j}$ because the actual $d^*_{i,j}$ is not available since the sending time can be falsified by idle time in HTTP chunks.

To guarantee Theorem 1 does not impact the maximum QoE when the LLLS uncertainty is eliminated, we propose Theorem 2 to justify our $d^u_{i,j}$ computation:

***Theorem 2:*** With accurate $u_{i,j}$, $s_{i,j}$ and $pc_i$, objective 9 falls back to regular objective $\max_{R_i} QoE$.

***Proof 2 (Theorem 2):*** The proof is intuitive, accurate variables imply $\Delta d_i = |d^*_i - d_i| = 0$, thus $d^u_{i,j} = d_{i,j} + \frac{|\Delta d_i| \times d_{i,j}}{d_i} = d_{i,j}$. Therefore the min solution is plain. In contrast, if we adopt $\Delta d_i = |d^*_i - d^u_i| = |d_i - d^u_i| \neq 0$, we will overestimate $d^u_{i,j}$ which is supposed to be $d_{i,j}$, therefore the min solution is not maximized. ■

TABLE II: Bandwidth (Kbps) distribution of datasets.

| Datasets \ Metrics | Mean | Std | 25th | 50th | 75th |
|---|---|---|---|---|---|
| FCC | 1526 | 1288 | 596 | 901 | 3426 |
| Oboe | 2966 | 1552 | 1765 | 2940 | 3957 |
| 3G/HSDPA | 1933 | 1053 | 1130 | 1836 | 2472 |
| Online | 8594 | 5674 | 2789 | 8115 | 15359 |

**Implementation.** Finally, to derive the max solution, we apply MPC algorithm to combine our novel LLLS model to iterate all possible bitrates combination. The specific procedure of AAR's ABR is in Algorithm 2. We first derive the average d-error from the past 5 segments (line 1). Then we perform regular MPC search, where each $u_{k,j}$ is estimated by server's chunk idle time (line 5), and the first CMAF chunk also experiences additional client's fetching idle time and RTT (line 6). $s_{k,j}$ is estimated by the last segment's chunk size ratio, and the total segment size $s_k$ is fixed as $R_k \times K \times T$ (line 7). Based on Theorem 1, we leverage the upper bound $d^u_{k,j}$ to estimate the minimum QoE. Finally, we can derive the maximum QoE (line 13) for each iteration and search for the optimal bitrate (line 15).

## IV. Evaluation

### A. Evaluation Setup

We deploy an HTTP server and dash.js (v.4.7.2) client in different Ubuntu server IPs. The video codec setting is from the mmsys-grand-challenge [14]: The target latency $l_{target}$ and buffer are 1.5 seconds with threshold $\delta = 0.3$, and the duration for each segment is 0.5 seconds, which contains $K = 15$ CMAF chunks and exactly 15 frames, meaning FPS=30 and $T = 33ms$. We use the DASH reference video from [15] encoded at 6 bitrates: $R_i \in$ {200, 600, 1000, 2500, 4000, 6000}Kbps. The Chrome-dev tool is used to simulate the network bandwidth change as suggested in [14], while network trace datasets include FCC [16], 3G/HSDPA [17], Oboe [18] and a real-world dataset collected in live streaming of an online e-commercial APP. The specific bandwidth distribution is in Table II.

**Bandwidth Measurement Baselines.** We compare AAR with 7 baselines, i.e. Fleet [7], DeeProphet [19], Moof parsing [6] in LoL+, I-Moof that replaces the wrong $moof + mdat$ parsing in [6] with AAR's improved patch (line 3 in Algorithm 1), AAST [20] that decides which chunks are downloaded at network speed or at producer rate, Seg that directly uses the total download time to evaluate bandwidth, Default in dash.js that filters out HTTP chunks' download time whose inter-arrival time is longer than the average.

**ABR Baselines.** We select 6 representative baselines: LoL+ [6], L2ALL [21], STALLION [22], rate-based (RB) that picks the highest bitrate under the available bandwidth, the default ABR Dynamic in dash.js, and Pensieve [23] which is a reinforcement learning-based ABR originally designed for VOD scenario, we retrain this model with our new live streaming

TABLE III: Bandwidth measurement/prediction error (%) ↓ for different methods.

| Category | Dataset | AAR | Fleet | I-moof | Moof | AAST | Seg | Default | DeeProphet |
|---|---|---|---|---|---|---|---|---|---|
| Measurement | FCC | 2.55±0.97 | 21.49±4.07 | 18.66±2.99 | 68.15±5.53 | 24.89±4.50 | 45.10±6.54 | 5679 | 11.8 |
| | Oboe | 3.59±0.48 | 13.78±2.06 | 21.00±5.17 | 69.55±5.97 | 27.31±4.66 | 61.08±7.84 | 1762 | |
| | HSDPA | 3.97±0.26 | 17.42±1.91 | 27.56±3.55 | 76.11±4.00 | 27.34±3.57 | 59.79±5.15 | 2336 | |
| | Online | 3.02±0.20 | 18.07±1.97 | 18.79±4.36 | 45.51±7.46 | 30.31±4.93 | 85.69±3.78 | 562 | |
| Prediction | FCC | 5.42±1.36 | 21.52±3.29 | 18.31±3.10 | 66.74±6.19 | 27.04±3.97 | 45.09±6.36 | 5421 | 16.50 |
| | Oboe | 6.50±1.03 | 15.55±2.35 | 20.95±4.98 | 69.07±6.06 | 28.07±4.66 | 60.31±7.68 | 1608 | |
| | HSDPA | 15.06±1.46 | 24.15±2.31 | 30.84±3.87 | 79.60±4.07 | 31.19±4.02 | 60.47±5.06 | 2219 | |
| | Online | 14.58±2.97 | 26.38±4.09 | 26.09±6.15 | 51.82±9.19 | 37.20±7.38 | 84.61±3.41 | 541 | |

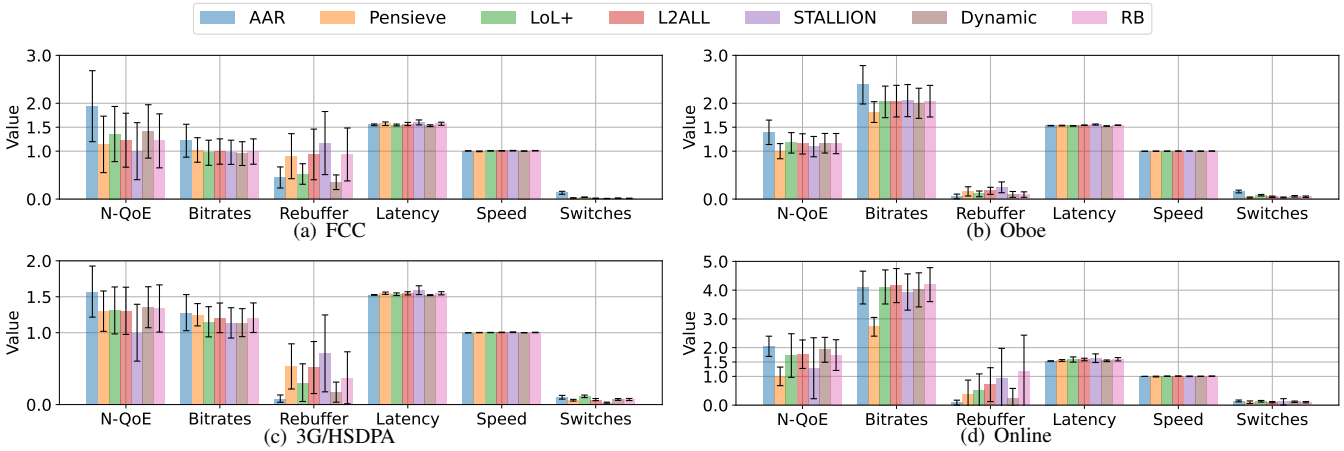



Fig. 5: ABR's QoE comparison with fixed bandwidth measurement. We present the normalized QoE ↑, bitrates in Mbps ↑, rebuffer in stalling ratio % ↓, latency in seconds → 1.5, relative playback speed → 1, and bitrate switches in Mbps ↓.

simulator using random samples from FCC and 3G/HSDPA datasets and the QoE model in Equ. 1. Note that the RMPC baseline is presented in **ablation study**, refer to Section IV-D for setup.

**Algorithm Parameters.** All the baselines including AAR are parameter-free, the input parameters from Algorithm 1 and 2 are not prefixed and they depend on the specific streaming content.

**Evaluation Metrics.** (1) Absolute relative error $\frac{|x-x^*|}{x^*}$, where $x$ and $x^*$ is the estimated and ground truth value, $x$ can represent bandwidth measurement and prediction, and also buffer prediction in Section IV-D. (2) QoE model. We follow Equ. 1 and set the weights $\alpha_i$ according to N-QoE [11] adopted by prior work [6], [19]. Specifically, $R_{min} = 200, R_{max} = 6000$, and $\alpha_1 = 0.5, \alpha_2 = R_{max}, \alpha_3 = 0.02 \times R_{min}$ if $l_i < 1.6$ and $0.1 \times R_{max}$ if not, $\alpha_4 = R_{min}, \alpha_5 = 1$.

### *B. Bandwidth Measurement Results*

**Bandwidth Measurement.** We first demonstrate AAR's accurate bandwidth measurement. For the sake of fairness, we fix the ABR as RB, and present the average measurement errors in Table III. Note that we apply AAR's improved CMAF identification patch to all the baselines except Moof. We directly report DeeProphet's performance stated in [19].

We can find that AAR consistently outperforms all the baselines with 8%-80% improvement throughout all the datasets. Specifically, compared to the second and existing SOTA measurement DeeProphet, AAR still surpasses with 8%-9% error reduction, let alone other methods. However, DeeProphet needs extra setup on the server side, and the measurement result $c_i$ is stored at server's database, which can only be delivered to the client upon $seg_{i+1}$ request. As a result, it modifies the requested bitrate to be lower than the measured bandwidth from the server side to prevent such delay, which has altered the whole LLLS framework and is not practical to deploy. Surprisingly, we find I-moof can reduce at most 50% error compared to the original Moof, which validates the importance of our correct $moof + mdat$ search for the later measurement. AAST and Seg are relatively worse as they neglect the CMAF chunk level idle time. Default is the worst because its HTTP chunk filtering strategy is too aggressive, random noises would render some chunks to be removed for false idle time.

**Bandwidth Prediction.** To further demonstrate the importance of measurement, we also present the prediction errors using the trivial smooth average in Table III. The relative performance among baselines is similar to that in measurement. However, we find that AAR's prediction accuracy drops for 3G/HSDPA and Online dataset, because if the original network trace varies a lot, it won't help even with accurate measurement, we would need extra approaches to predict the bandwidth evolution such as the learning-based in [19], which is beyond the scope of our work. In general, AAR's measurement outperforms all baselines with extremely low error rate, which in turn benefits the prediction performance.

TABLE IV: ABR's QoE ↑ with different bandwidth measurements.

| Dataset | Mea \ ABR | AAR | Pensieve | LoL+ | L2ALL | STALLION | Dynamic | RB |
|---|---|---|---|---|---|---|---|---|
| FCC | AAR | 5.23±2.00 | 3.08±1.59 | 3.66±1.55 | 3.31±1.52 | 2.70±1.60 | 3.81±1.50 | 3.28±1.52 |
| | Fleet | 5.20±2.01 | 1.00±2.06 | 3.42±1.69 | 1.37±2.20 | 2.51±1.73 | 3.97±1.51 | 1.63±2.08 |
| Oboe | AAR | 1.40±0.25 | 1.01±0.15 | 1.18±0.21 | 1.16±0.21 | 1.10±0.21 | 1.17±0.20 | 1.16±0.21 |
| | Fleet | 1.23±0.27 | 1.00±0.18 | 1.18±0.24 | 1.17±0.27 | 1.10±0.21 | 1.12±0.22 | 1.14±0.26 |
| HSDPA | AAR | 1.84±0.41 | 1.52±0.33 | 1.54±0.38 | 1.53±0.38 | 1.17±0.46 | 1.59±0.33 | 1.57±0.38 |
| | Fleet | 1.83±0.42 | 1.24±0.45 | 1.40±0.43 | 1.32±0.63 | 1.00±0.55 | 1.45±0.34 | 1.31±0.61 |
| Online | AAR | 2.20±0.38 | 1.07±0.35 | 1.86±0.81 | 1.91±0.53 | 1.38±1.14 | 2.07±0.46 | 1.87±0.57 |
| | Fleet | 2.22±0.39 | 1.00±0.66 | 1.94±0.75 | 1.49±0.96 | 1.72±0.59 | 2.04±0.45 | 1.71±0.76 |

### C. ABR QoE Results

For a fair comparison, we fix the bandwidth measurement as AAR. The aggregate statistics for QoE scores and individual QoE components are shown in Fig. 5. Note that we normalize the metric value by the worst baseline. Overall, AAR outperforms all the baselines with 39%-104% QoE improvement, which mainly stems from high bitrates and low rebuffering simultaneously. In comparison, the SOTA baseline in LLLS LoL+ is inferior mainly in rebuffering, especially for 3G/HSDPA and Online due to the varying bandwidth. Dynamic ranks second since it tends to pick the lowest bitrate among all the ABR rules, which favors the rebuffering and latency QoE components. L2ALL shares similar performance as RB while STALLION sometimes gets the worst result, which means simply improving future bandwidth estimation provides limited QoE improvement. Pensieve performs the worst for Oboe and Online dataset due to the unseen traces during training, therefore the picked bitrate is lower than available bandwidth. This may apply to other learning-based ABRs that depend on the training sample distribution like BDQ [12]. In general, AAR outperforms existing ABR baselines with higher QoE, bitrates and lower rebuffing.

### D. Ablation Study

To validate the effectiveness of each module of AAR, we combine different ABRs with different bandwidth measurements in Table IV. Note that we only pick the representative Fleet measurement for comparison, because DeeProphet requires extra setup and alters our LLLS framework, despite its performance. For each dataset, we normalize the QoE with the minimum value across 2 measurements and 7 ABRs. The results demonstrate AAR's measurement generally improves ABR's performance and vice versa. As for Online dataset, Fleet+AAR's ABR performs better than AAR+AAR, and LoL+ and STALLION also present better results combined with Fleet rather than AAR, it's mainly because most Online traces contain high bandwidth, and Fleet generally overestimates the bandwidth (refer to Fig. 2 (b)), which causes ABRs to select higher bitrates. However, the highest bitrate is 6000Kbps which does not even make up 50% of the Online dataset (See Table II). Therefore high bitrates may not trigger stalling at all and QoE rather improves.

To demonstrate our novel max-min objective improves overall QoE, we replace the min solution $d^u_{i,j}$ with regular $d_{i,j}$, and by contrast we adopt RMPC's bandwidth prediction error module $pc \leftarrow \frac{pc}{1+|\Delta pc|}$, denoted as AAR-1 (RMPC). To demonstrate the rationale for $\Delta d_i = |d^*_i - d_i|$ that guarantees adaption, we instead use $\Delta d_i = |d^*_i - d^u_i|$ to estimate the upper bound $d^u_{i,j}$, denoted as AAR-2. For better comparison, we also use the buffer prediction error $\frac{|b-b^*|}{b^*}$ to evaluate ABR's modeling performance. The QoE and B-Error results are in Table V. We can find that AAR outperforms AAR-1 and AAR-2 with QoE improvement of 3%-18% and B-Error reduction of 0.6%-2.6%. AAR-1 ranks the bottom because our max-min objective is the essential guarantee for robustness against all possible LLLS uncertainty. Despite AAR-2's higher QoE than AAR-1, its B-Error is still high because of the inappropriate estimation of $d^u_{i,j}$, as validated by Theorem 2.

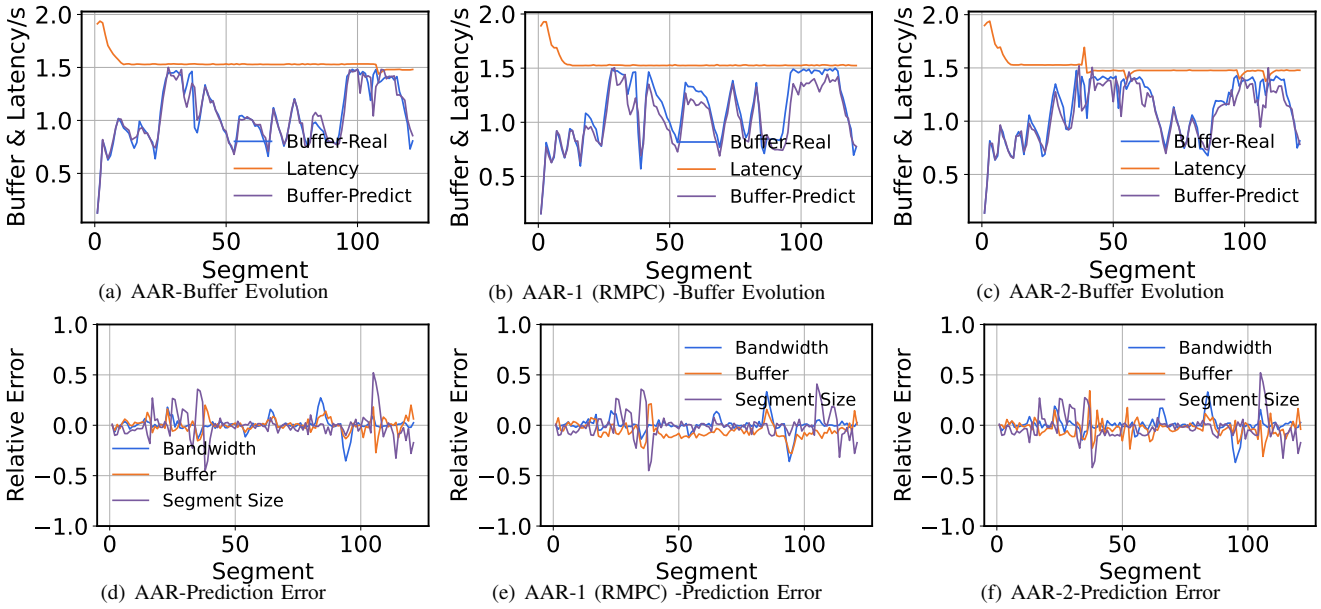
TABLE V: QoE ↑ and B-Error ↓ ablation study of AAR's ABR.

| Dataset | Metric \ ABR | AAR | AAR-1 (RMPC) | AAR-2 |
|---|---|---|---|---|
| FCC | QoE | 1.14±0.43 | 1.00±0.39 | 1.13±0.42 |
| | B-Error | 6.67%±1.61% | 7.37%±1.94% | 7.27%±1.88% |
| Oboe | QoE | 1.12±0.20 | 1.00±0.24 | 1.11±0.21 |
| | B-Error | 4.38%±0.33% | 6.44%±0.72% | 6.42%±0.68% |
| 3G/HSDPA | QoE | 1.18±0.26 | 1.00±0.25 | 1.18±0.27 |
| | B-Error | 5.39%±0.48% | 7.66%±0.85% | 7.58%±0.81% |
| Online | QoE | 1.04±0.17 | 1.00±0.16 | 1.01±0.14 |
| | B-Error | 4.79%±1.00% | 7.39%±1.56% | 6.84%±1.21% |

To better perceive the difference, we present intuitive examples of buffer evolution and prediction errors in Fig. 6. We can find in (a) that AAR's predicted buffer is close to the ground truth value, even when the segment size varies around segment 40 in (d), thanks to our novel max-min objective that adjusts the download time estimation based on historical deviation. In contrast, AAR-1 performs worse because RMPC only considers bandwidth uncertainty by design principles, therefore the size fluctuation around segment 40 in (e) leads to inferior buffer prediction in (b). AAR-2 adopts $\Delta d_i = |d^*_i - d^u_i| > 0$ even with low deviation around segment 50 in (f), which renders varying download time and buffer estimation. Therefore the latency also fluctuates in (c) due to playback speed adjustment to guarantee the buffer threshold (1.5s).

### E. Overhead Analysis

Regarding bandwidth measurement, different from existing methods like DeeProphet [19] and CLBE [24] that alters streaming protocols or introduces additional time delay, AAR




(a) AAR-Buffer Evolution (b) AAR-1 (RMPC) -Buffer Evolution (c) AAR-2-Buffer Evolution

(d) AAR-Prediction Error (e) AAR-1 (RMPC) -Prediction Error (f) AAR-2-Prediction Error

Fig. 6: Buffer evolution (a)-(c) and corresponding relative prediction error (d)-(f).

**Does Not** modify the client architecture and only slightly changes the server side, which does not bring any additional time overhead, because the Flag parameter is attached in the HTTP chunk header, which is due to be sent out in the first place. In addition, the cached CMAF chunk number can be easily obtained from the server side. AAR's ABR also induces no additional overhead even with a max-min objective, because we have derived a closed form solution from Theorem 1 without any optimization. In summary, AAR improves bandwidth measurement accuracy and ABR QoE with near zero overhead, demonstrating AAR's feasibility in real world deployment.

## V. Related Work

**Bandwidth Measurement.** We have introduced two representative measurement schemes in Section II-A. Alternatively, there are also cross-layer measurements such as DeeProphet [19] that leverages the TCP cwnd from server's transport layer to decide which of the packets are sent consecutively, reducing TCP blocking time. However, this induces additional setup overhead and the measured results can't be delivered to the clients before the segment request and ABR decision, which greatly hinders real-world deployment. CLBE [24] also proposes to use the captured packet information at the client side to compute packet level bandwidth. However, it requires queries from third-party module via WebSocket, which can induce additional time overhead, and a single packet's bandwidth still suffers deviation from noises the same way as Fleet.

**ABR Algorithm.** The history of ABR starts with heuristic methods like rate-based and RMPC [10], followed by deep learning-based schemes like Pensieve [23] and Comyco [25]. Karma [26] improves QoE by learning the causality among past observations, returns and actions. Jade [27] instead proposes to optimize QoE by aligning with user's scores via RLHF [28]. SODA [29] proposes to optimize a time-based QoE with theoretical guarantees. More recently, LLLS ABRs are proposed to optimize latency and playback speed like LoL+ [6] in Section II-B. L2ALL [21] instead solves an online convex optimization problem to derive optimal bitrate. STALLION [22] improves bandwidth measurement with the mean and deviation for better decisions. Tightrope [12] proposes the BDQ framework that leverages reinforcement learning to control both bitrates and playback speed. However, it's only implemented in offline simulator and lacks compatibility with real world streaming. SLVS [30] also proposes to generate a parameter table in a simulator according to bandwidth distribution for hybrid ABR to adjust the bandwidth weights.

## VI. Conclusion

In this work, we present two challenges in LLLS with respect to inaccurate bandwidth measurement and various LLLS uncertainty. Based on the insights, we propose the AAR framework consisting of two modules. On the one hand, we propose to attach a Flag parameter to identify the burst CMAF chunks to accumulate consecutive HTTP chunks for smooth bandwidth via size weighted average. We also improve the CMAF boundary identification to further guarantee accuracy. On the other hand, we propose a novel max-min ABR objective to guarantee the lower bound of QoE. We propose theorems to derive the min solution and guarantee adaption. To address the max objective, we further propose a novel LLLS model and apply MPC to search for the optimal bitrate. Real world experiments demonstrate that AAR outperforms 7 measurement and 6 ABR baselines with significant improvement. The ablation study also validates the effectiveness of each module of AAR.

## REFERENCES


[1] Cisco, "Vni complete forecast highlights," 2021. [Online]. Available: https://www.cisco.com/c/dam/m/en_us/solutions/service-provider/vni-forecast-highlights/pdf/Global_2021_Forecast_Highlights.pdf

[2] J. Li, Z. Li, R. Lu, K. Xiao, S. Li, J. Chen, J. Yang, C. Zong, A. Chen, Q. Wu *et al.*, "Livenet: a low-latency video transport network for large-scale live streaming," in *Proceedings of the ACM SIGCOMM 2022 Conference*, 2022, pp. 812–825.

[3] I. O. for Standardization and I. E. Commission, "Multimedia application format (mpeg-a)–part19: Common media application format (cmaf) for segmented media. standard iso/iec 23000-19:2018," 2018. [Online]. Available: https://www.iso.org/standard/71975.html,2018

[4] R. Fielding and J. Reschke, "Hypertext transfer protocol – http/1.1, rfc 7230." 2014. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc7230,2014

[5] Dash-Industry-Forum, "lldash," 2020. [Online]. Available: https://github.com/Dash-Industry-Forum/dash.js/wiki/Low-Latency-streaming

[6] A. Bentaleb, M. N. Akcay, M. Lim, A. C. Begen, and R. Zimmermann, "Catching the moment with lol+ in twitch-like low-latency live streaming platforms," *IEEE Transactions on Multimedia*, vol. 24, pp. 2300–2314, 2021.

[7] Y. Li, X. Zhang, C. Cui, S. Wang, and S. Ma, "Fleet: Improving quality of experience for low-latency live video streaming," *IEEE Transactions on Circuits and Systems for Video Technology*, 2023.

[8] A. Bentaleb, C. Timmerer, A. C. Begen, and R. Zimmermann, "Bandwidth prediction in low-latency chunked streaming," in *Proceedings of the 29th ACM workshop on network and operating systems support for digital audio and video*, 2019, pp. 7–13.

[9] P. K. Yadav, A. Bentaleb, M. Lim, J. Huang, W. T. Ooi, and R. Zimmermann, "Playing chunk-transferred dash segments at low latency with qlive. association for computing machinery, new york, ny, usa, 51–64," 2021.

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

[11] "Qoe evaluation for cmaf-based low-latency streaming," 2020. [Online]. Available: https://github.com/twitchtv/acm-mmsys-2020-grand-challenge/blob/master/NQoE.pdf

[12] L. Sun, T. Zong, S. Wang, Y. Liu, and Y. Wang, "Tightrope walking in low-latency live streaming: Optimal joint adaptation of video rate and playback speed," in *Proceedings of the 12th ACM Multimedia Systems Conference*, 2021, pp. 200–213.

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

[14] "Grand challenge on adaptation algorithms for near-second latency," 2020. [Online]. Available: https://2020.acmmmsys.org/lll_challenge.php

[15] (2022) bbb-videos. [Online]. Available: https://dash.akamaized.net/akamai/bbb_30fps/

[16] F. C. Commission, "Raw data - measuring broadband america," 2016. [Online]. Available: https://www.fcc.gov/reports-research/reports/

[17] H. Riiser, P. Vigmostad, C. Griwodz, and P. Halvorsen, "Commute path bandwidth traces from 3g networks: analysis and applications," in *Proceedings of the 4th ACM Multimedia Systems Conference*, 2013, pp. 114–118.

[18] "Oboe trace," 2022. [Online]. Available: https://github.com/USC-NSL/Oboe

[19] K. Chen, B. Wang, W. Wang, X. Li, and F. Ren, "Deeprophet: Improving http adaptive streaming for low latency live video by meticulous bandwidth prediction," in *Proceedings of the ACM Web Conference 2023*, 2023, pp. 2991–3001.

[20] "Aast," 2022. [Online]. Available: https://github.com/Dash-Industry-Forum/dash.js/pull/3660

[21] T. Karagkioules, R. Mekuria, D. Griffioen, and A. Wagenaar, "Online learning for low-latency adaptive streaming," in *Proceedings of the 11th ACM Multimedia Systems Conference*, 2020, pp. 315–320.

[22] C. Gutterman, B. Fridman, T. Gilliland, Y. Hu, and G. Zussman, "Stallion: video adaptation algorithm for low-latency video streaming," in *Proceedings of the 11th ACM Multimedia Systems Conference*, 2020, pp. 327–332.

[23] H. Mao, R. Netravali, and M. Alizadeh, "Neural adaptive video streaming with pensieve," in *Proceedings of the conference of the ACM special interest group on data communication*, 2017, pp. 197–210.

[24] C. Shende, C. Park, S. Sen, and B. Wang, "Cross-layer network bandwidth estimation for low-latency live abr streaming," in *Proceedings of the 14th Conference on ACM Multimedia Systems*, 2023, pp. 183–193.

[25] T. Huang, C. Zhou, R.-X. Zhang, C. Wu, X. Yao, and L. Sun, "Comyco: Quality-aware adaptive video streaming via imitation learning," in *Proceedings of the 27th ACM international conference on multimedia*, 2019, pp. 429–437.

[26] B. Xu, H. Chen, and Z. Ma, "Karma: Adaptive video streaming via causal sequence modeling," in *Proceedings of the 31st ACM International Conference on Multimedia*, 2023, pp. 1527–1535.

[27] T. Huang, R.-X. Zhang, C. Wu, and L. Sun, "Optimizing adaptive video streaming with human feedback," in *Proceedings of the 31st ACM International Conference on Multimedia*, 2023, pp. 1707–1718.

[28] P. F. Christiano, J. Leike, T. Brown, M. Martic, S. Legg, and D. Amodei, "Deep reinforcement learning from human preferences," *Advances in neural information processing systems*, vol. 30, 2017.

[29] T. Chen, Y. Lin, N. Christianson, Z. Akhtar, S. Dharmaji, M. Hajiesmaili, A. Wierman, and R. K. Sitaraman, "Soda: An adaptive bitrate controller for consistent high-quality video streaming," in *Proceedings of the ACM SIGCOMM 2024 Conference*, 2024.

[30] G. Zhang, K. Liu, M. Xiao, B. Wang, and V. Aggarwal, "An intelligent learning approach to achieve near-second low-latency live video streaming under highly fluctuating networks," in *Proceedings of the 31st ACM International Conference on Multimedia*, 2023, pp. 8067–8075.