Hindawi
Security and Communication Networks
Volume 2022, Article ID 4992957, 13 pages
https://doi.org/10.1155/2022/4992957

Research Article

Adaptive Bandwidth Prediction and Smoothing Glitches in Low-Latency Live Streaming

Dapeng Wu[1,2,3] **Linfeng Cui**[1,2,3] **Tong Tang**[1,2,3] **and Ruyan Wang**[1,2,3]

[1]*School of Communication and Information Engineering, Chongqing University of Posts and Telecommunications, Chongqing 400065, China*
[2]*Advanced Network and Intelligent Interconnection Technology Key Laboratory of Chong Qing Education Commission of China, Chongqing 400065, China*
[3]*Chongqing Key Laboratory of Ubiquitous Sensing and Networking, Chongqing 400065, China*

Correspondence should be addressed to Tong Tang; tangtong@cqupt.edu.cn

Received 17 January 2022; Accepted 13 April 2022; Published 9 May 2022

Academic Editor: Qi Li

HTTP adaptive streaming (HAS) technologies such as dynamic adaptive streaming over HTTP (DASH) and common media application format (CMAF) are now used extensively to deliver live streaming services to large numbers of viewers. However, in dynamic networks, inaccurate bandwidth prediction may result in the wrong request of bitrate, and short-term network fluctuations may produce glitches, causing unnecessary bitrate switching, thereby degrading clients' Quality of Experience (QoE). To tackle this, we propose adaptive bandwidth prediction and smoothing glitches in low-latency live streaming (called APSG) in this article. Concretely, firstly, the size of random bandwidth fluctuations is exploited as the weight of exponentially weighted moving average (EWMA) for adaptive bandwidth prediction; in addition to bandwidth prediction and buffer occupancy, glitches phenomena under a stable network environment are taken into account to enhance the viewing experience of clients. Finally, experimental results show that compared to traditional ABR algorithms under a stable network environment, APSG could reduce the number of bitrate switches and latency by up to 72.6% and 27.3%, respectively; under a dynamic network environment, APSG could reduce the number of bitrate switches and latency by up to 53.8% and 23.6%, respectively.

1. Introduction

In recent years, the development of mobile networks and streaming technologies has enabled clients to watch live streaming on their mobile devices at any time, with video accounting for 67% of global traffic in 2016 and expected to reach 80% by 2022, according to Cisco's Annual Visual Networking Index Report [1]. Today, live streaming platforms, such as Huya and Douyu, attract millions of active clients, and video content providers have become more interested in live streaming as client's engagement will directly increase commercial revenue. This trend means that high-quality videos with fewer switches, lower latency, less rebuffering, and higher bitrate need to be provided to clients.

In DASH, the video is divided into multiple segments, each with a duration of approximately 2 to 10 seconds, and encoded at different bitrates and resolutions [2]. On the client's side, the ABR algorithm takes into account network environments or buffer occupancy to pick the right bitrate for the clients and fetch it from the server. In a traditional video on demand (VoD) scenario, DASH has a large end-to-end latency due to the fact that the entire segment is completely downloaded before it is added to the playback buffer and queued for playback. If the buffer content is empty, then the rebuffering will occur. In live streaming, the latency is generated by the process of capturing video from the anchor to the server and decoding it by the client. The latency is proportional to the size of the segment, and if the duration of the segment is reduced to achieve the purpose of reducing the latency, then the number of requests and the round-trip time (RTT) will increase significantly. In order to achieve target latency without reducing the duration of the segments, CMAF is a method [3]. CMAF can divide the segments into smaller chunks and then transmit them by

HTTP. When the coding of the chunk is completed, it will be sent to the client, and the remaining chunks of the segment will be sent without additional requests; it is unnecessary to send them to the client only after the coding of the entire segment is completed. CMAF significantly reduces latency.

CMAF could reduce latency but bring new challenges; bandwidth measurement becomes nonnegligible. The biggest difference between live streaming and VoD is that live content is generated in real time. Bandwidth measurement usually uses the size of the segment divided by the download time of the segment in the VoD. However, HTTP does not provide a download start time for each chunk within the segment. If the requested chunk is not encoded, it is necessary to wait until the chunks coding is completed to send it to the client. During this period, there must be idle times between the two chunks. VoD's bandwidth measurement method will underestimate the download rate [4–6]. The ABR algorithm will choose a low bitrate, directly reducing the client's QoE. To solve this problem, Bentaleb et al. proposed the first solution to calculate the bandwidth through the sliding window moving average (SWMA) bandwidth measurement method [7]. When the chunk download rate is close to the average download rate of the segment, this chunk must be disregarded. Although the problem of bandwidth underestimation is solved, the bandwidth will be overestimated, and it is more likely to rebuffer when watching live streaming. In order to solve the problem of bandwidth overestimation, Ozcelik and Ersoy considered the whole segment and subtracted the download end time of the consecutive chunks [8]. If the value is less than the average download time of the segment, then it is considered that there are no idle times between the two chunks. These chunks are used to approximate the download time of the remaining chunks within the segment and reduce the impact of idle times.

Once the exact bandwidth has been measured, the two most important parts are bandwidth prediction and bitrate selection. Traditional bandwidth prediction methods based on time series models are weighted to historical data, which can be estimated online in real-time. Fixed parameters or weights are difficult to apply to all network situations, a smaller number of samples may produce unstable prediction values when the bandwidth changes drastically, and the correlation between premature historical data and current bandwidth is weak. When the bandwidth is at a certain point, there is a glitches phenomenon (i.e., the stable network suddenly changes and then gets back to the original network state), and the bitrate changes accordingly. This unnecessary bitrate switching will directly affect the client's viewing experience.

To address the above issues, firstly, this article proposes an adaptive bandwidth prediction method, which calculates the network stability factor based on historical data, designs the weight values based on the network stability factor, and obtains the adaptive bandwidth prediction values by EWMA. Then APSG algorithm is proposed for smoothing glitches. The bandwidth is differentiated into a stable and dynamic network environment by calculating the network stability factor. The glitches phenomenon is smoothed under a stable network environment. This article takes into account both bandwidth prediction and buffer occupancy and adopts corresponding strategies to select the appropriate bitrate. APSG algorithm is implemented under the DASH.js reference player [9] and extensive experiments have been done under different network environments. The experiments enable APSG to compare with two traditional algorithms in terms of live latency, average bitrate, and the number of bitrate switches.

The rest of this article is organized as follows. Related work is provided in Section 2, followed by the details for the APSG scheme in Section 3. Section 4 presents the experimental evaluation, and Section 5 concludes the article.

2. Related Work

In the past decade, many ABR algorithms have been proposed, which can be divided into four main categories: (1) available bandwidth-based adaptive bitrate algorithms; (2) playback buffer-based adaptive bitrate algorithms; (3) mixed adaptive bitrate algorithms; and (4) data-driven adaptive bitrate algorithms.

(1) Available bandwidth-based adaptive bitrate algorithms: in this type of scheme, the most important thing is to accurately predict bandwidth. Jiang et al. proposed an algorithm called FESTIVE, which mitigates bandwidth jitter caused by stop-and-wait mechanisms by optimizing video chunks scheduling and uses harmonic mean to predict bandwidth [10]. The PANDA proposed by Li et al. uses an EWMA with a weight of 0.2 for bandwidth prediction [11]. Bentaleb et al. implemented a CMAF-based bandwidth measurement algorithm for live streaming and recursive least-squares- (RLS-) based bandwidth prediction [7]. However, the bandwidth overestimation problem occurs when the idle time increases. Ozcelik and Ersoy further addressed the problem of bandwidth overestimation due to idle times based on [7] and used an EWMA method with a weight of 0.9 to calculate the available bandwidth for the next segment [8]. van der Hooft et al. proposed an HTTP/2-based algorithm that discards unimportant frames within a segment when the selected bitrate does not match the available bandwidth [12]. Existing work has shown that selecting the next bitrate based on inaccurate bandwidth prediction values can cause low-quality video or playback rebuffering.

(2) Playback buffer-based adaptive bitrate algorithms: in this type of scheme, clients use the playout buffer occupancy as a criterion to select the next segment bitrate during video playback. Huang et al. proposed a buffer-based bitrate selection algorithm called BBA, which selects the bitrate based on a linear function aimed at maximizing the average video quality and avoiding unnecessary rebuffering events [13]. Spiteri et al. designed a buffer-based online control algorithm that uses Lyapunov optimization techniques to minimize rebuffering and maximize

video quality [14]. Essentially, these two algorithms are mapping the current buffer occupancy. Huang et al. developed a QoE model, including rebuffering, the number of bitrate switches, and video quality, and formulated the problem as a nonlinear stochastic optimal control problem [15]. A dynamic buffer-based controller is designed for DASH using control theory to determine the bitrate of each segment. Qin et al. proposed a framework for PIA by further analyzing ABR video streaming based on proportional-integral-derivative (PID) control and combining several ABR algorithms to address various business requirements [16]. Both of these approaches use a PID controller to control the buffer occupancy. The adaptive bitrate algorithm based on the playback buffer has many limitations, the most serious being that in low-latency live streaming scenarios, the size of the buffer that can be used is drastically reduced, especially under long-term bandwidth fluctuations; there are problems of overall low QoE, unstable selection of bitrates, and too many bitrate switches.

(3) Mixed bitrate algorithms: in this type of scheme, clients select bitrate based on the combination of metrics, including available bandwidth and buffer occupancy. Pioneering this critical work was Yin et al., who modelled bitrate adaptation as a stochastic optimal control problem, proposing a model predictive control (MPC) approach to model cache dynamics and then select the bitrate by optimizing the overall QoE function based on bandwidth prediction and current buffer occupancy as inputs [17], but MPC is sensitive to bandwidth prediction errors and network jitter. A fuzzy logic-based bitrate adaptive algorithm and prediction mechanism was proposed that takes into account buffer occupancy and the prediction of available network bandwidth in order to be able to respond proactively to requests [18]. Reference [19] considered the joint decision of two factors and minimized video bitrate switching. Yarnagula et al. designed a segment-aware rate adaptation (SARA) algorithm by considering segment size to predict the time to download the next segment [20].

(4) Data-driven adaptive bitrate algorithms: CS2P proposed by Sun uses a hidden Markov model to design a prediction model by analyzing the evolution trajectory of download rate [21]. In [22, 23], an ABR algorithm is based on deep reinforcement learning. With the powerful approximation ability of the neural network, the best mapping between various states and bitrate selection is learned. However, when encountering untrained network environments, the overall QoE will be very poor. Another disadvantage is that it is difficult to reproduce these ABR algorithms based on deep reinforcement learning [24, 25].

The approach used in this paper is based on a joint decision between bandwidth prediction and current buffer occupancy. On the one hand, the adaptive bandwidth prediction method is used to improve bandwidth prediction accuracy; on the other hand, the APSG method is used to solve the glitches phenomenon, thus significantly improving the quality of service experience for clients.

3. Proposed APSG

In this article, APSG is designed to improve bandwidth prediction accuracy and eliminate glitches caused by bandwidth fluctuations under different network environments. Figure 1 shows the components of the APSG in DASH.js [9], which contains five parts: (1) bandwidth measurement module; (2) bandwidth prediction module; (3) ABR control module; (4) logger module; and (5) playback speed control module. This section will introduce each module in the DASH.js player and elaborate on the details. The list of notations used in APSG is given in Table 1.

3.1. APSG Process. This article selects the appropriate bitrate for clients to match the current network environments. Firstly, the download rate is measured based on the history of the segment after removing the idle times; then, the proposed prediction method is used to get the predicted value of the download rate of the next segment. Then the current buffer occupancy and bandwidth prediction are combined to jointly determine the bitrate of the next segment to be requested and subsequently place the downloaded video in the playback buffer and determine the buffer status and whether the playback speed control module needs to be invoked.

3.2. APSG Design. The next section of this article describes the core functional design and implementation details of the APSG.

3.2.1. Bandwidth Measurement Module. The module uses a heuristic bandwidth measurement method that is able to reduce the impact of idle times on the calculation of download rates. As mentioned earlier, the chunks are encoded and transmitted at the same time, chunks that are not encoded to completion become unavailable, and unavoidable idle time is generated between two consecutive chunks. Therefore, this article uses the bandwidth measurement method of [8]. The method is shown as follows.

Firstly, this article expresses the size and download time of each chunk in a segment as a sequence: $L_i = \{x_i^1, x_i^2, x_i^3, \ldots, x_i^n\}$; there are n chunks in each segment, t_i^j denotes the download end time of the ith segment's jth chunk, and s_i^j denotes the size of the ith segment's jth chunk. We can know the download end time but do not know the download start time of each chunk, so we use the download end time of consecutive chunks subtracted from each other as the download time of each chunk. The average arrival time of successive chunks in each segment is as follows:

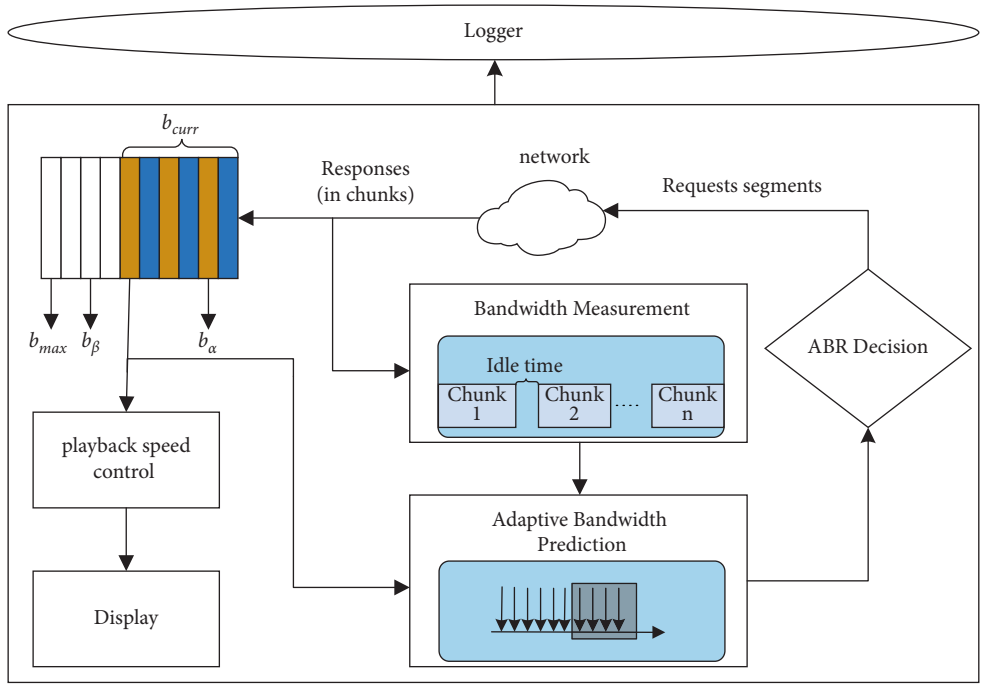


Figure 1: APSG overview in DASH.js reference play.

Table 1: List of notations.

Notation	Meaning
B_i	Download rate of the *ith* segment
$\bar{t}_i$	Average download time of the *ith* segment
t_i^j	Download time of the *jth* chunk of the *ith* segment
x_i^j	The *jth* chunk of the *ith* segment in the extra sequence
s_i^j	Size of the *jth* chunk of the *ith* segment
T_i	Download time of the *ith* segment
T_i'	Download time of the *ith* segment without idle times
ρ_i	Approximate coefficient of the *ith* segment
α	Network stability factor
θ	Network stability factor threshold
$\widehat{B}_i$	The predicted bandwidth of the *ith* segment
r_i	Bitrate of the *ith* segment
k	k segments in the video sequence

$$\bar{t}_i = \frac{t_i^n - t_i^1}{n-1}. \tag{1}$$

Secondly, these chunks are considered to be unaffected by the encoding side and has already been encoded at the time of request without idle times if the chunks are downloaded faster than the average arrival time. Place these chunks in an additional sequence L_i':

$$L_i' = \left(x_i^j\right), \text{ s.t.t}_i^j - t_i^{j-1} \le \bar{t}_i \forall x_i^j \in L. \tag{2}$$

Then use the chunks in L_i' to calculate an approximate download rate ρ_i:

$$\begin{aligned} \rho_i &= \frac{\sum_{x_i^j \in L'} s_i^j}{T_i'}, \\ T_i' &= \sum_{x_i^j \in L'} t_i^j - t_i^{j-1}. \end{aligned} \tag{3}$$

Once the approximate download rate is obtained, the total size of the remaining chunks and ρ_i are used to estimate the effective download time, reducing the effect of idle times in this step:

$$T_i = \frac{\sum_{x_i^j \in (L/L')} s_i^j}{\rho_i}. \tag{4}$$

Therefore, the average download rate of the *ith* segment is calculated according to $T_i' + T_i$ of (3) and (4):

$$B_i = \frac{\sum_{x_i^j \in L} s_i^j}{T_i' + T_i}. \tag{5}$$

3.2.2. Bandwidth Prediction Module. This module dynamically and adaptively predicts bandwidth in the APSG. It consists of two phases: Phase 1, which calculates the size of network fluctuation; Phase 2, which adaptively predicts the bandwidth of the next segment based on phase 1.

At phase 1, firstly, it is necessary to distinguish whether the network environments are transient jitter or long-term changes, and this article introduces a network stability factor α, the ratio of the standard deviation to the mean of a set of data to indicate the size of the fluctuation. α is calculated by the actual download rate of the last m segments of the sliding window and constructing a set $\{B_{i-m+1}, B_{i-m+2}, \cdots, B_i\}$. The network stability factor is calculated as shown in the following equation:

$$\alpha = \frac{\sqrt{\sum_{j=i-m+1}^{i} \left(B_j - 1/m \sum_{j=i-m+1}^{i} B_j\right)^2}}{1/m \sum_{j=i-m+1}^{i} B_j}. \tag{6}$$

α characterizes the dispersion of the download rate of the nearest m segments. When the value of α is small, it means that there is little fluctuation in bandwidth during the period, but there may be glitches, and those transient changes can seriously affect the client's viewing experience, so the network stability factor is designed to find and eliminate glitches. The network stability factor threshold θ is defined to measure the fluctuation of the bandwidth. The current network environment is considered to be in a stable network environment if α belongs to $(0, \theta)$; otherwise, it is considered to be in a dynamic network environment.

Bandwidth prediction for chunked video streams is not easy, and to solve this problem, traditional bandwidth prediction methods are as follows:

(1) Segment-based last bandwidth: the last successfully downloaded segment is used to predict the next segment

(2) Sliding Window Moving Average (SWMA) [7]: using the last three successfully downloaded segments, find their average

(3) Exponentially Weighted Moving Average (EWMA) [8]: exponentially weighted average bandwidth of the last four segments

(4) Harmonic mean: the total size of the last five segments is divided by the total download rate

The four prediction methods mentioned above all share a common feature, where SWMA and Harmonic have a fixed window size and EWMA has fixed weights, so there cannot predict different network environments. For example, when bandwidth fluctuations are small, there is high prediction accuracy, but when the fluctuations become larger, too early measurements are less relevant to the current network environment. The player will take a long time to download the selected chunks of a high prediction value and therefore may run out of content in the buffer during the download process, causing rebuffering. In VoD scenarios, the ABR algorithm usually has enough cached content to absorb errors, but the playback buffer is small in live streaming scenarios; the wrong request of bitrate causes rebuffering, which can seriously affect the client's viewing experience. Fixed parameters are difficult to apply to all network environments.

So, in phase 2, an adaptive bandwidth prediction method is designed in this article. An initial weight α^j is set for each segment based on the network stability factor, $j \in \{0, 1, \cdots, m-1\}$; m is the window size. Normalizing these weights to their geometric sum, the final weights for each segment are expressed as follows:

$$\alpha_j = \frac{\alpha^j (1-\alpha)}{1-\alpha^m}. \tag{7}$$

Considering that there is a strong correlation between the bandwidth prediction value and the bandwidth fluctuation, this article needs to choose the appropriate weight and prediction formula according to the size of the network stability factor. The bandwidth prediction formula is expressed as follows:

$$\widehat{B}_{i+1} = \begin{cases} \sum_{j=0}^{m} \alpha_j B_{i-j}, & 0.6 < \alpha < 1, \\ \sum_{j=0}^{m} \alpha_{m-j} B_{i-j}, & \alpha > 1, \\ \frac{1}{m} \sum_{j=0}^{m} B_{i-j}, & \text{others}. \end{cases} \tag{8}$$

When the network stability factor is less than 0.6, it means that the network is in a stable state. The detrimental effects of the glitch's phenomenon can be well reduced using m segments average for prediction. When α is in the other two ranges, it means that the network changes greatly or drastically, the average value of historical data as the bandwidth prediction values may result in a wrong request of bitrate. The segment closer to the current moment has a greater impact on predicting the next segment, so it is given a greater weight value. From (7), it can be obtained that the size of the weights varies with the drastic changes in the network, and we can make predictions adaptively according to the stability of the network.

3.2.3. ABR Control Module. The ABR algorithm is the core of the process and the bitrate chosen directly determines the client's viewing experience. In different scenarios, ABR algorithms have different objectives. In the VoD scenario, there is no buffering requirement. To ensure a better viewing experience for the clients, it will increase or decrease step by step rather than changing suddenly because the buffer is large enough to allow this. However, live streaming requires fewer switches, lower latency, less rebuffering, and higher bitrate within the constraints of a small buffer, and an appropriate bitrate is very difficult to choose.

As with typical bitrate adaptation algorithms, the APSG selects the most appropriate one from the set of available bitrates. The downloaded segments are placed in the playback buffer, and this article defines four buffer thresholds and the maximum playback buffer, which are b_I b_α, b_β, $b_{\max}$. b_{curr} represents the current buffer occupancy. As shown in Figure 2, all thresholds are defined in terms of time. The proposed algorithm uses the joint decision of buffer occupancy and bandwidth prediction to select the bitrate for the next segment.

The details of the APSG algorithm are shown in Algorithm 1. R: $\{r^0, r^1, \cdots, r^{\max}\}$ defined as the set of bitrates available for video in the server. The buffer threshold b_I b_α, b_β, $b_{\max}$, the network stability factor threshold θ, and the set of bitrates available R are initialized. These parameters do not change during each experiment. The bitrate of the next segment is chosen with knowledge of the following parameters: Download the bitrate of the current segment (r^{curr}), the current buffer occupancy(b_{curr}), network stability factor α, the size corresponding to the next available segment of bitrate $s_{i+1}(f) = \{s_{i+1}(0), s_{i+1}(1), \cdots, s_{i+1}(\max)\}$, and the adaptive bandwidth prediction value $\widehat{B}_i$ calculated by (8).

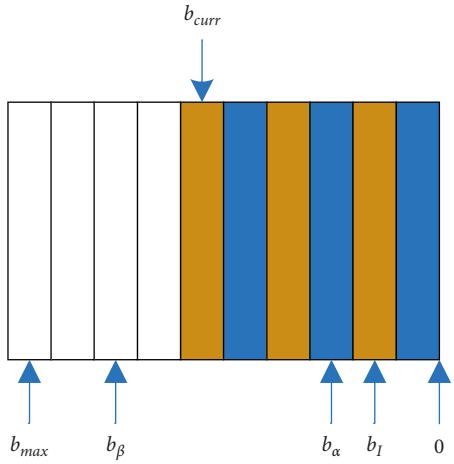


Figure 2: Video playback buffer model.

According to the current buffer occupancy and playback time, the proposed algorithm is divided into four stages, which are described as follows:

Stage 1 ($b_{curr} \le b_I$ or $i < 4$): there is nothing in the playback buffer when the video starts to play; the lowest bitrate is chosen to ensure low initial latency and also to fill the buffer quickly to avoid rebuffering. The lowest bitrate is chosen during the initial playback stage of the video for two purposes: firstly, the currently known sample values for download rates are too few to use the proposed bandwidth prediction method; secondly, it prevents clients from giving up watching the video due to rebuffering.

Stage 2 ($b_{curr} \le b_\alpha$): the algorithm enters stage 2 when the current buffer occupancy is less than b_α. At this stage, the selected bitrate is incremented by one level and is no longer played at the lowest bitrate, improving the quality of the video.

Stage 3 ($b_\alpha < b_{curr} \le b_\beta$): when the playback buffer occupancy is between b_α and b_β, this is the desired stage in this article. At this stage, the dynamic changes in the network environment can be used to determine whether the current network is in a stable state, and if it is in a stable state, the glitches phenomenon can be eliminated based on the choice of bitrate, reducing unnecessary switching of bitrate and improving the viewing experience of the client.

Stage 4 ($b_\beta < b_{curr} \le b_{\max}$): when the playback buffer occupancy exceeds b_β, There is a risk of video content overflow, resulting in a lost frame. The video download is paused until the playback buffer occupancy returns to stage 3 in the traditional method. The bandwidth resources are wasted, and the proposed bandwidth prediction method cannot be used because the download rate cannot be obtained during the pause. So, a playback speed control module is used to avoid causing overflow and wasted bandwidth resources.

3.2.4. Logger Module. The module regularly records various metrics such as bitrate, buffer occupancy, rebuffering, measured bandwidth values, and predicted bandwidth values.

3.2.5. Playback Speed Control Module. The client has a catch-up function that adjusts the playback rate to pull the player back to the target real-time edge. The player can keep itself close to the target latency by controlling the playback rate, which relies on the assumption that changes in playback rate are not significant enough for the end client at 25% or less [26]. It, therefore, speeds up or slows down the playback rate within a range of (0.75, 1.25) depending on the difference between the target latency and the current latency. To determine the actual value in this range, this article relies on the default implementation in the DASH.js player, which uses a sigmoid function.

4. Experimental Result

4.1. Experimental Design

4.1.1. Test Set. As with the existing CMAF-based live servers, this article uses the video sequence Big Buck Bunny [27], encoded using x264 [28] into three different bitrates{360p@200 Kbps, 480p@600 Kbps, 720p@1000 Kbps}. The encoded video was then segmented using MP4Box [29] into 0.5-second segments for DASH and into 0.5-second segments with 33-millisecond chunks for CMAF-DASH. The resulting segments/chunks were used in the DASH.js framework. The whole video sequence intercepted the first 300 seconds of Big Buck Bunny.

4.1.2. Test Platform. In order to build an end-to-end live streaming system, two Ubuntu 20.04 virtual machines were built using personal computers, the first running the DASH.js player in the Google Chrome browser (v97) [30]. Another virtual machine is used to enable the CMAF wrapped FFmpeg encoder and put it into the server. To simulate the network, the network bandwidth is controlled on the server PC using the TC [31] network traffic shaping tool, a module of the Linux kernel, whose control principle is to restrict the transmission of packets at the transport layer so that the data traffic can be shaped.

4.1.3. Bandwidth Configuration. This article is a live streaming scenario with a single client exclusive link bandwidth, and the bandwidth variation is modelled by two scenarios: a stable network and a dynamic network, the details of which are depicted in Figure 3. In a stable network, there are two glitches at 50 and 110 seconds and a sudden decrease and increase in bandwidth at 220 and 270 seconds, respectively. There are two glitches during the dynamic network, and the network fluctuates dramatically after 150 seconds.

Data:
R: $\{r^0, r^1, \cdots, r^{\max}\}$: Set of available bitrates
b_I, b_α, b_β, $b_{\max}$: Buffer thresholds
θ: Network stability factor threshold
INPUT:
r^{curr}: Bitrate of the most recently downloaded segment
b_{curr}: Current playback buffer occupancy (in seconds)
α: Network stability factor
$s_{i+1}(f) = \{s_{i+1}(0), s_{i+1}(1), \cdots, s_{i+1}(\max)\}$ are the $(n+1)th$ segment sizes for bitrates $\{r^0, r^1, \cdots, r^{max}\}$
$\widehat{B}_{i+1}$: The Adaptive bandwidth prediction of the $(i+1)th$ segment
Initialization
if $i < 4$ or $b_{curr} < b_I$;
then
$l_{i+1} = r^0$;
else
if $b_{curr} < b_\alpha$;
then
$l_{i+1} = r^1$;
else if $b_{curr} < b_\beta$;
then
if $\alpha < \theta$;
then
$l_{i+1} = l_i$;
else
$l_{i+1} = \max\{r^f | r^f \in R, s_{i+1}(f)/\widehat{B}_{i+1} \le b_{curr} - b_I\}$;
else if $b_{curr} > b_\beta$;
then
$l_{i+1} = \max\{r^f | r^f \in R, s_{i+1}(f)/\widehat{B}_{i+1} \le b_{curr} - b_\alpha\}$;
end
end
Result:
l_{i+1}: The bitrate of the next segment to be downloaded

Algorithm 1: Adaptive prediction and smoothing glitches algorithm.

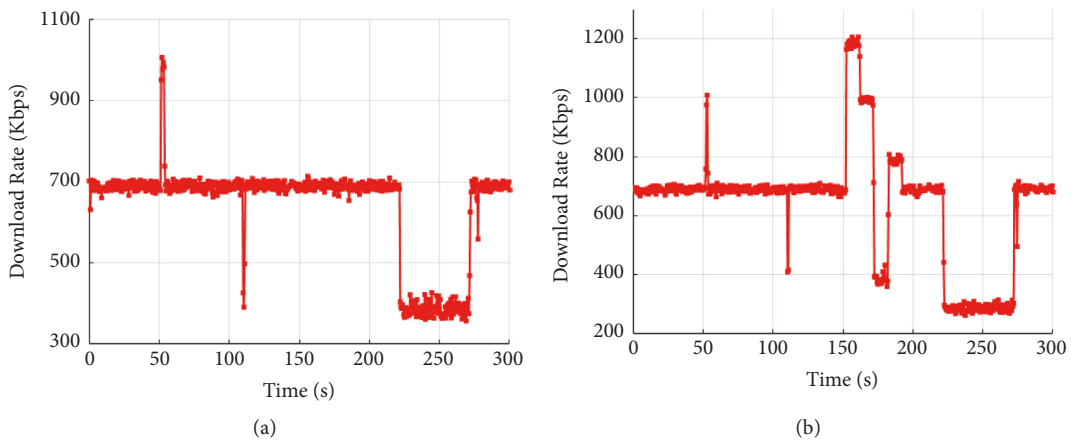


Figure 3: Bandwidth configuration. (a) Stable network; (b) dynamic network.

4.1.4. ABR Comparison Algorithm. The proposed APSG was compared with two traditional algorithms, the available bandwidth-based bitrate adaptation algorithm FESTIVE [10] and the buffer-based bitrate adaptation algorithm (BBA) [13]. Bandwidth is predicted by an average of 5 historical data, and then the maximum bitrate less than the

predicted value is selected in FESTIVE. The BBA uses a linear mapping function where the bitrate is chosen based on the buffer occupancy.

4.1.5. Performance Metrics. The following performance metrics are prediction errors and QoE parameters.

(1) Prediction errors: the bandwidth prediction errors model is based on Root Mean Square Error (RMSE), which is calculated based on the difference between the bandwidth prediction and the actual bandwidth measurement, as follows:

$$RMSE = \sqrt{\frac{1}{k}\sum_{i=1}^{k}\left(\frac{\widehat{B}_i - B_i}{B_i}\right)^2}. \quad (9)$$

(2) QoE parameters: after each segment is downloaded, this article considers three evaluation metrics of video average bitrate, the number of bitrate switches, and latency to analyze the performance of the proposed algorithms; the rebuffering problem was not considered because these three algorithms did not show cache underflow during the experiment.

Bitrate is one of the most important metrics of the client viewing experience; the higher bitrate brings a better viewing experience for the clients.

$$Q_i^v = \frac{1}{k}\sum_{i=0}^{k} r_i. \quad (10)$$

From the client's perspective, frequent bitrate switching is undesirable. Video is easily abandoned because bitrate switches from high to low; the following formula determines whether the bitrate switching occurs:

$$Q_i^s = \sum_{i=1}^{k-1}\left|r_i - r_{i-1}\right|. \quad (11)$$

VoD streaming has more relaxed latency requirements and can use large playback buffers, whereas live streaming cannot. To maintain interactivity, the most important requirement is low latency. The latency parameter T_d can be obtained directly in the DASH.js player.

In summary, the QoE model is as follows:

$$QoE_i = \mu Q_i^v - \gamma Q_i^s - \varsigma T_d, \quad (12)$$

where μ, γ, ς are the weights used to calculate the QoE. Each weight is given the following values: μ = segment duration; since this article focuses on bitrate switching, a large weight is given to γ, $\gamma = 20$; the latency is expressed in milliseconds in Dash.js, $\varsigma = 5000$. To simplify the representation of QoE, a normalized QoE with a value between 0 and 1 is used, N-QoE (QoE/QoEMAX).

4.2. Bandwidth Prediction Accuracy. In this section, the accuracy of traditional bandwidth prediction methods [10] is compared with that of the adaptive prediction methods in this article. Under the same network environment and parameters, the same ABR algorithm and playback speed control module is used in two prediction methods. Ten experiments were conducted to take the average of each ABR algorithm. However, this article cannot conclude from the observations that the proposed prediction method is better than the traditional method, so there are other aspects to prove that the proposed prediction is more accurate.

Bandwidth prediction error can be calculated by (9), which can directly measure the prediction accuracy at each time. Figure 4 shows the error values at each moment.

According to Figure 4, it can be seen that adaptive prediction produces lower errors in the stable networks at 50 and 110 seconds and 220 and 270 seconds, while at other times, these two errors overlap, shown in blue, because the mean prediction method is used when the network stability factor is less than θ. The same results can be clearly seen under a dynamic network. The proposed prediction method is calculated to reduce the error by 2% compared to the traditional mean prediction error under a stable network and by 5% under a dynamic network. If the network changes more and more dramatically, then the proposed prediction method will be more effective. The proposed prediction method is able to withstand small network changes as well as large ones, in contrast to traditional algorithms that are slower to respond to bandwidth changes because they only consider a fixed sample of historical measurements. Therefore, there is evidence that our prediction method outperforms the traditional method.

4.3. The Glitches Phenomenon. In this section, we verify that APSG can eliminate the glitches phenomenon. Figure 5 shows the bitrate selected results of the three algorithms in the stable network with bandwidth shown in Figure 3(a).

As can be seen from Figures 5(a) and 5(b), the lowest bitrate was chosen as the initial playback bitrate of all three algorithms at the beginning of playback. The glitches phenomenon was eliminated at 50 and 110 seconds, and the same bitrate as the previous segment was selected by the APSG algorithm. In Figure 3(a), the network suddenly changes immediately back to the original network environment, the bitrate selected by the FESTIVE algorithm is switched at 50 and 110 seconds, and small fluctuations occurred in 280 seconds, with bandwidth falling below 600 Kbps, resulting in lower bitrate requested. Figure 5(c) shows that the bitrate is selected by the BBA algorithm and it can be seen that although the glitches are eliminated, it is switched several times between 220 and 270 seconds when the network changes. In a live streaming scenario, the buffer is quite small. Once the network environment changes, the content of the buffer will continue to increase or decrease, resulting in fluctuations around the threshold, and the selected bitrate will be switched many times.

Figure 6 shows the bitrate selected results of the three algorithms in the dynamic network with bandwidth shown in Figure 3(b). As can also be seen in Figures 6(a) and 6(b), the proposed algorithm APSG is able to remove the glitches phenomenon and avoid bitrate switching due to transient

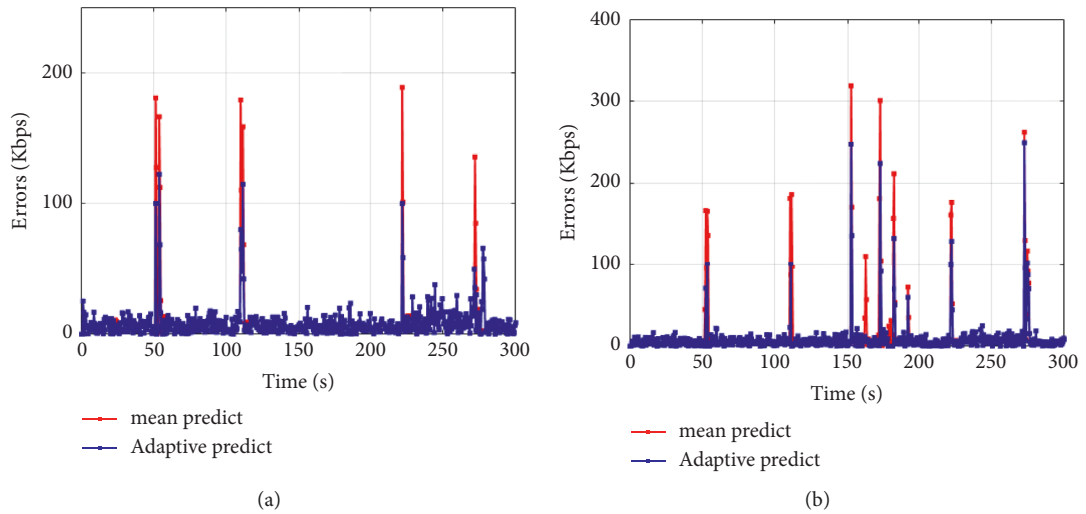


Figure 4: Prediction errors for different prediction methods under two network environments. (a) Stable network prediction errors; (b) dynamic network prediction errors.

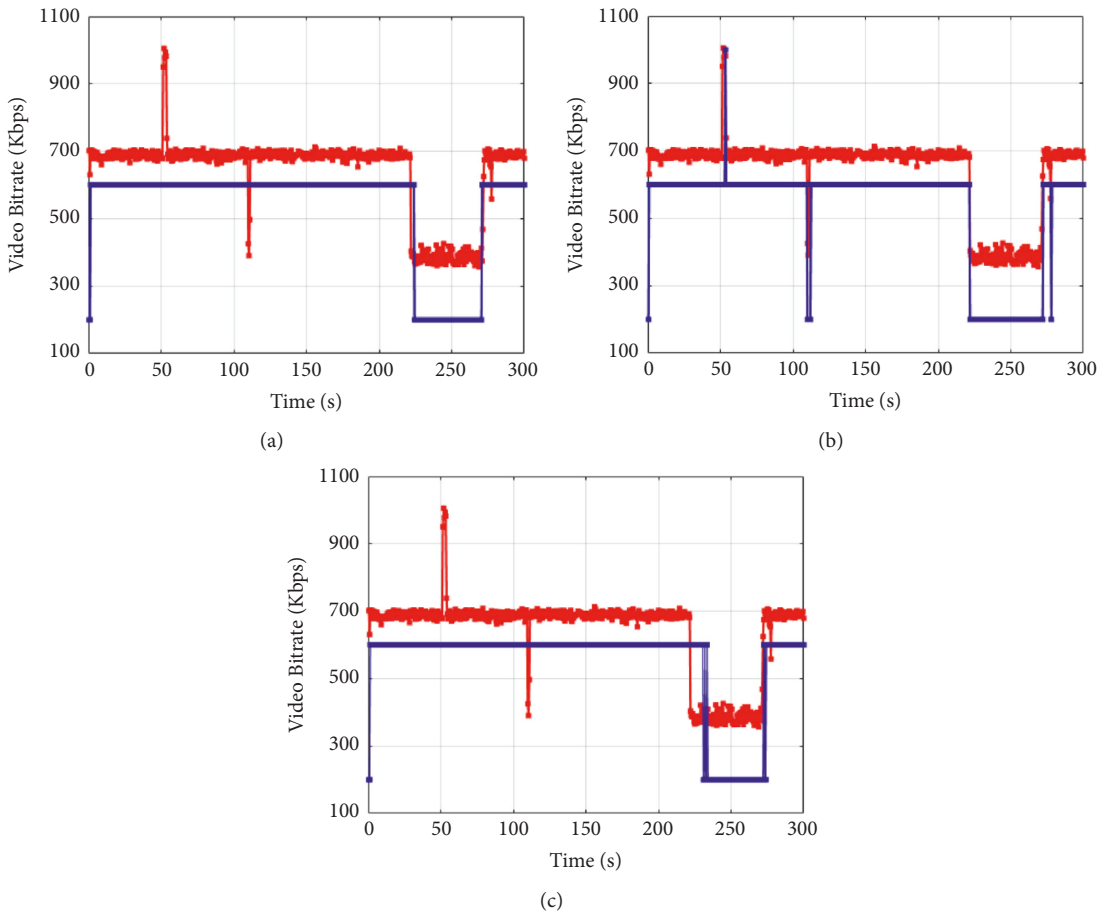


Figure 5: The bitrate selection results during the stable network; the red line represents the bandwidth and the blue line represents the selected bitrate. (a) APSG; (b) FESTIVE; (c) BBA.

bandwidth fluctuations. At about 160 seconds, the network environment becomes worse and the bitrate selection by the FESTIVE algorithm decreases; however, the APSG algorithm still maintains the original bitrate. This is because the bitrate changes less during this period, and the proposed bandwidth prediction method is closer to the real network

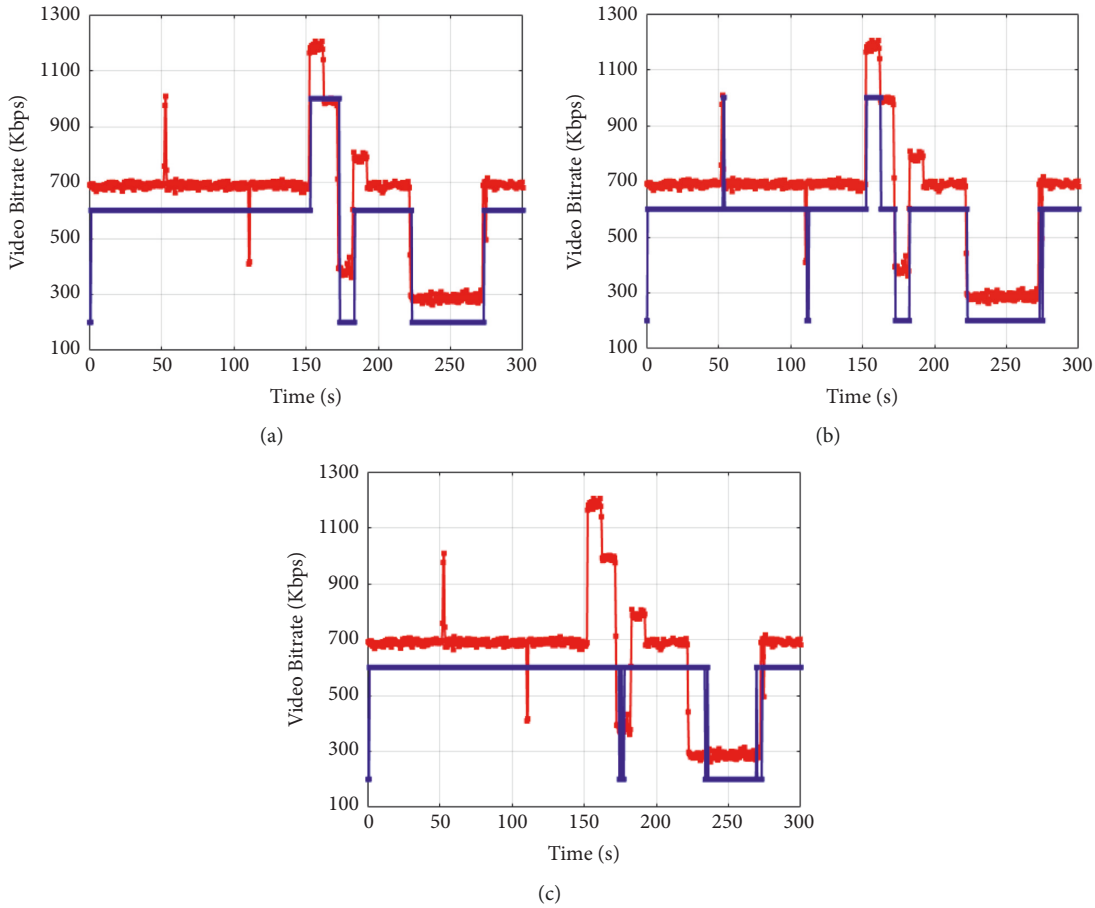


Figure 6: The bitrate selection results during the dynamic network; the red line represents the bandwidth and the blue line represents the selected bitrate. (a) APSG; (b) FESTIVE; (c) BBA.

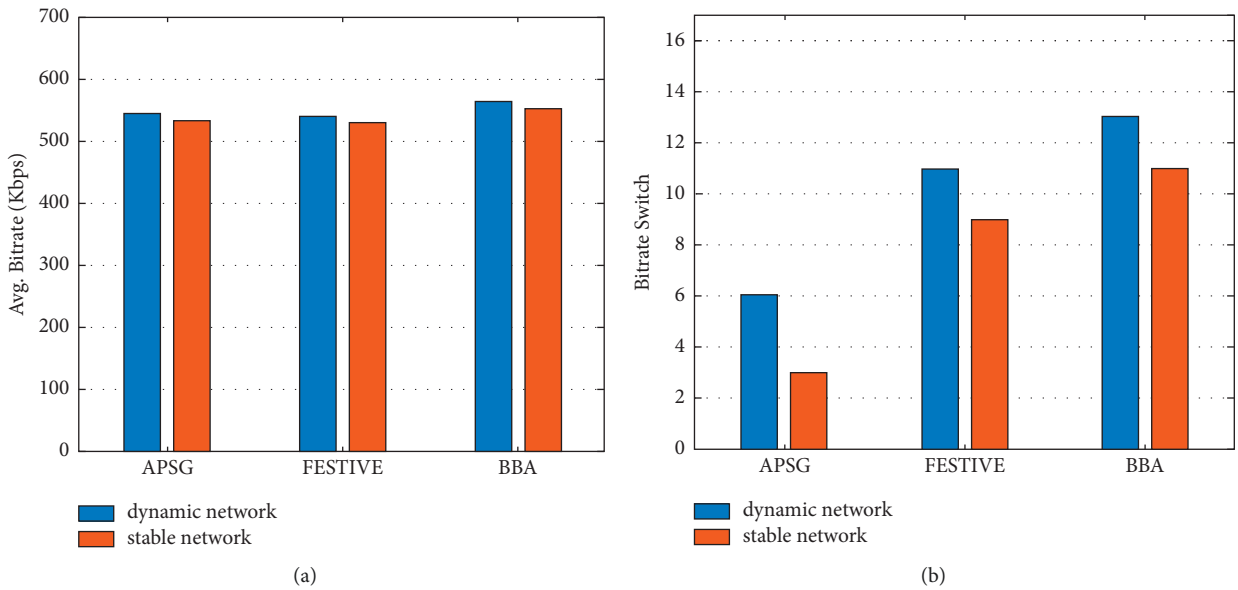
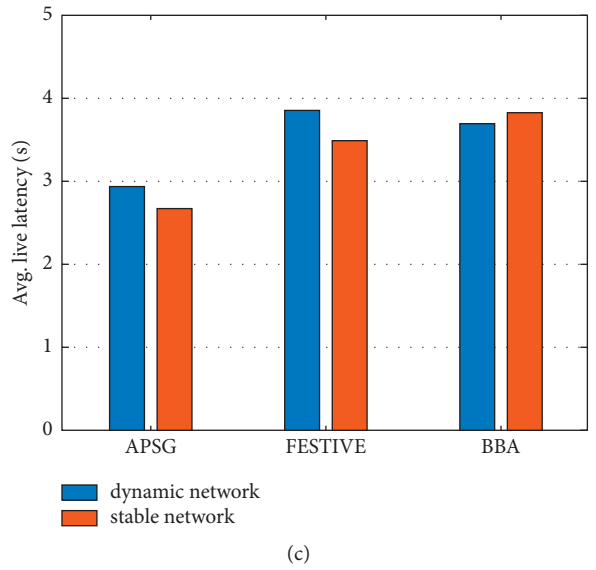


Figure 7: Continued.



(c)

Figure 7: Average bitrate, number of switches, and live latency for different ABR and different networks over 10 runs. (a) Average bitrate; (b) average number of switches; (c) average live latency.

Table 2: Average bitrate, live latency, and QoE with its metrics.

	Avg.Bitrate (Kbps)	Avg.Live Latency (s)	Avg. Switches	Avg.N-QoE
		Stable network		
APSG	531.334	2.68	3	0.98
FESTIVE [10]	528.667	3.47	9	0.85
BBA [13]	552.000	3.69	11	0.85
		Dynamic network		
APSG	544.697	2.94	6	0.98
FESTIVE [10]	540.667	3.85	11	0.86
BBA [13]	562.000	3.83	13	0.85

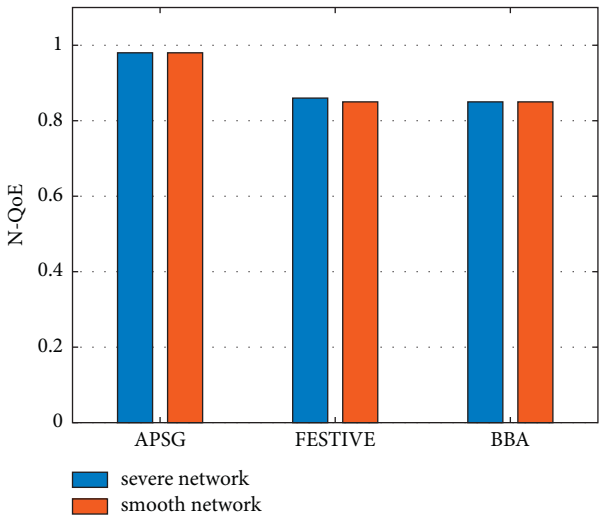


Figure 8: Average N-QoE.

environments, so there is no bitrate switching until the bandwidth is reduced again in 170 seconds. For Figure 7(c), the bitrate selection is similar to the stable network, where the appropriate bitrate is selected based on the buffer occupancy, which tends to ignore the reasonable use of network bandwidth from the playback cache perspective, resulting in wasted bandwidth resources.

4.4. QoE Performance. This section shows the comparison and summary of APSG and two traditional algorithms in QoE metrics. Table 2 and Figure 7 are QoE indicators of the three algorithms under two network environments. In terms of average bitrate, the APSG algorithm compared with the FESTIVE algorithm has little improvement but less than the BBA algorithm, the overall change is not big, and video quality can be considered to remain unchanged. In terms of switching, the APSG algorithm reduced the number of switches relative to the FESTIVE algorithm and the BBA algorithm by 6 and 8 under a stable network and by 5 and 7 under a dynamic network, respectively. In terms of latency, the APSG algorithm reduces it by 0.79 seconds and 1.15 seconds under a stable network and by 0.91 seconds and 0.75 seconds under a server network, respectively. Because this article references the playback speed control module of the DASH.js player, it is able to speed up or slow down the playback speed within range. As can be seen from the above results, unnecessary switching is reduced on the basis of guaranteed bitrate by the APSG algorithm.

Figure 8 shows the N-QoE of different algorithms under different network environments. It can be seen that the algorithm proposed has a higher QoE, while the other two traditional algorithms have a lower overall QoE due to the excessive number of switches. The number of bitrate switches is a key concern in this article, and the QoE obtained by the two traditional algorithms would be lower if the item is given a higher weight.

5. Conclusion

In this article, an adaptive bitrate scheme called APSG is proposed to improve prediction accuracy and eliminate the glitches phenomenon caused by bandwidth fluctuations. The ABR decision relies on three main components: (1) bandwidth measurement with idle time removed; (2) adaptive bandwidth prediction based on the size of network fluctuation; (3) a joint decision algorithm based on bandwidth prediction and buffer occupancy. Results showed that compared to traditional ABR algorithms stable network environment, APSG could reduce the number of bitrate switches and latency by up to 72.6% and 27.3%, respectively, under a dynamic network environment; APSG could reduce the number of bitrate switches and latency by up to 53.8% and 23.6%, respectively, achieving a better video service experience.

Although this work shows good performance in removing glitches, there is room for improvement. Next, we will consider making full use of the various network conditions' scenarios to improve the model and consider the human subjective factor to improve the quality of experience.

Data Availability

The datasets of this work are available from the corresponding author upon reasonable request.

Conflicts of Interest

The authors declare no conflicts of interest in publishing this article.

Acknowledgments

This work was supported by National Natural Science Foundation of China under Grants U20A20157, 61771082 and 61871062, the Science and Technology Research Program of Chongqing Municipal Education Commission under Grant KJQN201900604, the General Program of Chongqing Natural Science Foundation under Grant cstc2021jcyj-msxmX0032, the Natural Science Foundation of Chongqing, China (cstc2020jcyj-zdxmX0024), and the University Innovation Research Group of Chongqing (CXQT20017).

References

[1] V. Cisco, "Cisco visual networking index: forecast and trends, 2017–2022[J]," *White Paper*, vol. 1, no. 1, 2018.

[2] A. Bentaleb, B. Taani, A. C. Begen, C. Timmerer, and R. Zimmermann, "A survey on bitrate adaptation schemes for streaming media over HTTP," *IEEE Communications Surveys & Tutorials*, vol. 21, no. 1, pp. 562–585, 2019.

[3] K. Hughes and D. Singer, *Information Technology–Multimedia Application Format (MPEG-A)–Part 19: Common media Application Format (CMAF) for Segmented media*, pp. 23000–19, ISO/IEC, Geneva, Switzerland, 2017.

[4] D. Wu, R. Bao, Z. Li, H. Wang, H. Zhang, and R. Wang, "Edge-cloud collaboration enabled video service enhancement: a hybrid human-artificial intelligence scheme," *IEEE Transactions on Multimedia*, vol. 23, pp. 2208–2221, 2021.

[5] D. Wu, X. Han, Z. Yang, and R. Wang, "Exploiting transfer learning for emotion recognition under cloud-edge-client collaborations," *IEEE Journal on Selected Areas in Communications*, vol. 39, no. 2, pp. 479–490, 2021.

[6] T. Tang, L. Li, X. Wu et al., "TSA-SCC: text semantic-aware screen content coding with ultra low bitrate," *IEEE Transactions on Image Processing*, vol. 31, pp. 2463–2477, 2022.

[7] A. Bentaleb, C. Timmerer, A. C. Begen, and R. Zimmermann, "Performance analysis of ACTE," *ACM Transactions on Multimedia Computing, Communications, and Applications*, vol. 16, no. 2s, pp. 1–24, 2020.

[8] I. M. Ozcelik and C. Ersoy, "Low-latency live streaming over HTTP in bandwidth-limited networks," *IEEE Communications Letters*, vol. 25, no. 2, pp. 450–454, 2021.

[9] JS. Dash, "DASH Reference Player," 2019, https://reference.dashif.org/dash.js/.

[10] J. Jiang, V. Sekar, and H. Zhang, "Improving fairness, efficiency, and stability in HTTP-based adaptive video streaming with festive," *IEEE/ACM Transactions on Networking*, vol. 22, no. 1, pp. 326–340, 2014.

[11] Z. Li, X. Zhu, J. Gahm et al., "Probe and adapt: rate adaptation for HTTP video streaming at scale," *IEEE Journal on Selected Areas in Communications*, vol. 32, no. 4, pp. 719–733, 2014.

[12] J. van der Hooft, S. Petrangeli, T. Wauters et al., "HTTP/2-Based adaptive streaming of HEVC video over 4G/LTE networks," *IEEE Communications Letters*, vol. 20, no. 11, pp. 2177–2180, 2016.

[13] Te-Y. Huang, R. Johari, N. McKeown, M. Trunnell, and M. Watson, "A buffer-based approach to rate adaptation: evidence from a large video streaming service," in *Proceedings of the 2014 ACM Conference on SIGCOMM (SIGCOMM '14)*, pp. 187–198, NY, USA, October 2014.

[14] K. Spiteri, R. Urgaonkar, and R. K. Sitaraman, "BOLA: near-optimal bitrate adaptation for online videos," *IEEE/ACM Transactions on Networking*, vol. 28, no. 4, pp. 1698–1711, 2020.

[15] W. Huang, Y. Zhou, X. Xie, D. Wu, M. Chen, and E. Ngai, "Buffer state is enough: simplifying the design of QoE-aware HTTP adaptive video streaming," *IEEE Transactions on Broadcasting*, vol. 64, no. 2, pp. 590–601, 2018.

[16] Y. Qin, R. Jin, S. Hao et al., "A control theoretic approach to ABR video streaming: a fresh look at PID-based rate adaptation," *IEEE Transactions on Mobile Computing*, vol. 19, no. 11, pp. 2505–2519, 2020.

[17] X. Yin, A. Jindal, V. Sekar, and B. Sinopoli, "A control-theoretic approach for dynamic adaptive video streaming over HTTP," *ACM SIGCOMM - Computer Communication Review* in *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication (SIGCOMM '15)*, vol. 45, no. 4, pp. 325–338, NY, USA, October 2015.

[18] A. Sobhani, A. Yassine, and S. Shirmohammadi, "A video bitrate adaptation and prediction mechanism for HTTP adaptive streaming," ACM trans. Multimedia comput," *Commun. Appl*, vol. 13, no. 2, 2017.

[19] C. Wang, R. Amr, and M. Zink, "SQUAD: a spectrum-based quality adaptation for dynamic adaptive streaming over HTTP," in *Proceedings of the 7th International Conference on Multimedia Systems (MMSys '16)*, Association for Computing Machinery, NY, USA, May 2016.

[20] H. K. Yarnagula, P. Juluri, S. K. Mehr, V. Tamarapalli, D. Medhi, and D. Medhi, "QoE for mobile clients with segment-aware rate adaptation algorithm (SARA) for DASH video streaming," *ACM Transactions on Multimedia Computing, Communications, and Applications*, vol. 15, no. 2, pp. 1–23, 2019.

[21] Y. Sun, "CS2P: improving video bitrate selection and adaptation with data-driven throughput prediction," in *Proceedings of the 2016 ACM SIGCOMM Conference*, Florianopolis Brazil, August 2016.

[22] 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 (SIGCOMM '17)*, pp. 197–210, NY, USA, August 2017.

[23] N. Kan, C. Li, C. Yang, W. Dai, J. Zou, and H. Xiong, "Uncertainty-aware robust adaptive video streaming with bayesian neural network and model predictive control," in *Proceedings of the 31st ACM Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV '21)*, pp. 17–24, NY, USA, July 2021.

[24] Y. Tian, T. Li, J. Xiong, M. Z. A. Bhuiyan, J. Ma, and C. Peng, "A blockchain-based machine learning framework for edge services in IIoT," *IEEE Transactions on Industrial Informatics*, vol. 18, no. 3, pp. 1918–1929, 2022.

[25] J. Xiong, R. Bi, Y. Tian, X. Liu, and D. Wu, "Toward lightweight, privacy-preserving cooperative object classification for connected autonomous vehicles," *IEEE Internet of Things Journal*, vol. 9, no. 4, pp. 2787–2801, 2022.

[26] M. Kalman, E. Steinbach, and B. Girod, "Adaptive media playout for low-delay video streaming over error-prone channels," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 14, no. 6, pp. 841–851, 2004.

[27] Ton, "Big Buck Bunny," 2020, https://peach.blender.org/.

[28] x264, "VideoLAN, a project and a non-profit organization," 2020, https://www.videolan.org/developers/x264.html.

[29] Gpac, "MP4Box," 2020, https://gpac.wp.imt.fr/mp4box/.

[30] Chromium, "Chromium Web Browser 2020," 2020, https://www.chromium.org/.

[31] Ubuntu, "TC Module 2020," 2020, http://manpages.ubuntu.com/manpages/xenial/man8/tc.8.html.