A FUZZY-BASED ADAPTATION CONTROLLER FOR LOW LATENCY LIVE VIDEO STREAMING

Yunlong Li[⋆]*, Xinfeng Zhang*[‡]*, Shanshe Wang*[⋆]*, Siwei Ma*[⋆]

[⋆] National Engineering Laboratory for Video Technology, Peking University, Beijing, China
[‡] University of Chinese Academic of Sciences, Beijing, China

ABSTRACT

HTTP adaptive streaming with chunked transfer encoding can provide low latency live video streaming experience. However, providing high and smooth quality video streaming experience under low latency conditions poses a great challenge for Adaptive BitRate (ABR) algorithm design due to the shorter time to react to bandwidth fluctuations. To address the problem, we design a **F**uzzy logic controller based ABR for low latency live video streaming with **C**hunked **T**ransfer **E**ncoding (FCTE). We take player buffer size, throughput mean, and throughput standard deviation as inputs to make bitrate decisions. By making full use of network information and dealing with uncertainties using fuzzy language, FCTE can make quick and robust bitrate decisions in the complicated network environment. We evaluate FCTE over five network scenarios released by Twitch. FCTE achieves average QoE improvement by 129%, 338%, and 127% compared with three representative algorithms: L2A, LoL, and Stallion, respectively.

***Index Terms*—** DASH, CTE, Low Latency, Live Video Streaming, FLC

1. INTRODUCTION

Live video streaming is expected to grow to 13% of total Internet traffic by 2021[1]. It becomes more latency-sensitive to provide a better experience with the emergence of new live streaming scenarios such as live sports and interactive live broadcast. The latency is the time delay between the video capture and rendering, which is proportional to video segment duration[2] in HTTP Adaptive Streaming (HAS). Typically, video segment duration is 2-10 seconds and therefore introduces 10-50 seconds latency. Many streaming solutions have been proposed in the last decade to decouple segment duration from latency[3, 4, 5, 6, 7, 8, 9]. Finally, it becomes a widely recognized solution to package a video segment in the Common Media Application Format (CMAF) [10] specification and then deliver through chunked transfer encoding (CTE) [11], where video segments are split and transmitted in small non-overlapping parts, called chunks. This solution manages to reduce live streaming latency to 1-5 seconds, as a benefit of delivering video segments while they are still being packaged. By far, three solutions have been standardized based on this scheme: (i) DASH Low-Latency (DASH-LL) [12]; (ii) Low-latency HTTP Live Streaming (LHLS) [1]; (iii) Low-Latency HTTP Live Streaming (LLHLS) [13]. From high-level perspective, a common feature across all these solutions is that they require the media to be encoded and delivered in chunks, which is called chunked low latency live video streaming.

However, chunked live video streaming faces two major challenges. First, throughput measurement and prediction are more difficult. Video chunks available in the server will be sent out in bursts after receiving the fetch signal, whereas the remaining chunks will be sent out only when they are available. This behavior introduces an idle period between video chunks. It is difficult to estimate the chunk transmission duration because chuck transmission start time is not available in traditional HTTP protocol and cannot be replaced by previous chuck transmission end time for there may be the idle period. Second, it is difficult to design the ABR algorithm. Most existing ABR algorithms work well in non-low-latency scenario, where the player buffer can be 10 seconds. However, there is a limited video buffer (eg., 1s) to compensate for throughput fluctuations and incorrect bitrate selections in low latency scenario, which makes it a much more difficult task to design the ABR algorithm.

In this paper, we propose FCTE with the following contributions. First, FCTE filters the correct chunk transmission duration from the video chunk's arriving time according to [14] and then predict throughput mean, standard deviation, and trend. We take multiple throughput prediction metrics into fuzzy logic controller because we find that only throughput mean cannot correctly guide video bitrate selection in a volatile network environment (eg., mobile network). Second, FCTE takes throughput metrics and video player buffer size into the fuzzy logic controller. Through fuzzification, fuzzy engine (guided by expert experience in the form of fuzzy rules), and defuzzification, FCTE outputs an aggressive factor to guide bitrate selection. Third, FCTE chooses the next segment bitrate according to the aggressive factor and throughput mean. Finally, experiments over five streaming scenarios are conducted and FCTE outperforms three representative ABR algorithms: L2A [8], LoL [7], and Stallion [9] in quality of

experience (QoE).

The rest of this paper is organized as follows. Section 2 describes the proposed algorithm of Fuzzy Logic Controller, followed by performance evaluation in Section 3. Section 4 concludes the paper.

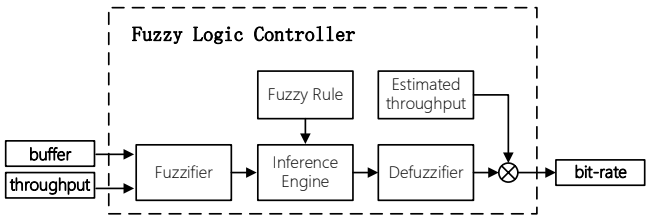


Fig. 1. Diagram of Fuzzy Logic Controller

2. THE PROPOSED ALGORITHM OF FUZZY LOGIC CONTROLLER

The proposed Fuzzy Logic Controller diagram is shown in Figure 1. As a control system based on fuzzy logic, buffer and throughput metrics are taken as inputs and the bitrate decision is output. *Inputs* and *Fuzzy Logic Controller Design* are described in detail below.

2.1. Inputs

Normalized Throughput Metrics. Throughput mean, standard deviation, and trend are taken into consideration. We filter correct chunk transmission speed from the video chunk's arriving time according to [14] and then predict throughput mean, standard deviation and trend. For a video sequence with N bitrate versions stored on the server, normalized throughput mean is defined,

$$BM_i = \begin{cases} \frac{\widehat{bm}_i - Q_n}{Q_{n+1} - Q_n} & n \neq N \\ 0 & n = N \end{cases} \quad BM_i \in [0,1], \qquad (1)$$

$\widehat{bm}_i$ is the estimated throughput mean of segment i ($Q_n \leq \widehat{bm}_i < Q_{n+1}$). Q_n is video bitrate level n ($1 \leq n \leq N$). BM_i is the normalized throughput mean.

The normalized throughput standard deviation is defined,

$$BV_i = \frac{\widehat{bt}_i * \widehat{bv}_i}{Q_{step}}, \qquad (2)$$

$\widehat{bt}_i$ is the throughput standard deviation estimation and $\widehat{bv}_i$ is the throughput trend estimation. Q_{step} is the average video bitrate interval.

Buffer Size. In low-latency live video streaming, video segments are downloaded sequentially. However, a video segment may be delayed when it is not available on the server. The buffer size transfer function can be defined as,

$$u_i = max\{u_{i-1} - t_{dowload} - t_{delay}, 0\} + \tau, \qquad (3)$$

$t_{dowload}$ is the download time of segment i and t_{delay} is the delayed time before segment i is available on the server. τ is the video segment duration. Video freeze event occurs when $u_{i-1} - t_{dowload} - t_{delay} < 0$ and video freeze time will be recorded for QoE evaluation.

2.2. Fuzzy Logic Controller Design

The proposed Fuzzy Logic Controller consists of three main parts: a fuzzier, a fuzzy engine, and a defuzzier.

Fuzzier: The fuzzier converts numerical values into linguistic language, which is used to deal with uncertainties. We define $F(x)$ as the fuzzier function which decomposes inputs into one or more fuzzy subsets. $F(x)$ is detailed in Table 1. The proposed Fuzzy Logic Controller consists of three inputs and one output. We define O_i as the output, which is the aggressive factor of the next bitrate decision. If O_i is small, conservative bit rate selection will be made and if O_i is large, aggressive bit rate selection will be made. NL, NS, ZO, PS, PL are short for Negative Large, Negative Small, Zero, Positive Small, Positive Large, respectively.

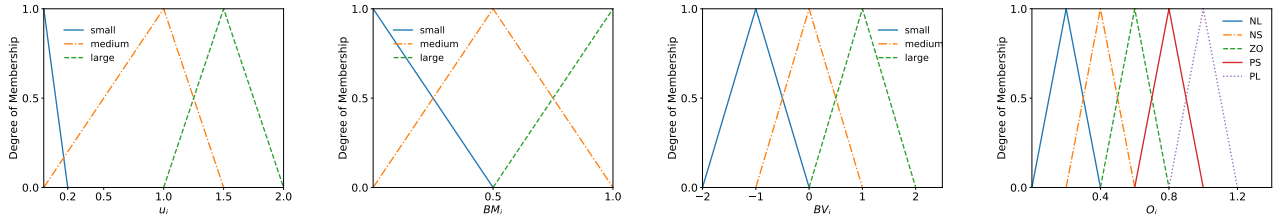
Membership functions are required in this step, which are defined and modified according to expert experience. The proposed membership functions are depicted in Figure 2.

Table 1. Fuzzier Functions

Numerical Variables	Fuzzy Language
$F(BM_i)$	{Small, Medium, Large}
$F(BV_i)$	{Negative, Zero, Positive}
$F(u_i)$	{Small, Medium, Large}
$F(O_i)$	{Negative Large(NL)/Small(NS), Zero(ZO), Positive Small(PS)/Large(PL)}

Fuzzy Engine and Fuzzy Rules: In fuzzy logic control, fuzzy engine is the connection between input and output, which follows fuzzy rules. Fuzzy rules are defined according to expert experience, which exit in the format of $if A, then B$. For example, if the buffer size is large, normalized throughput mean is large and the standard deviation is positive, then the aggressive degree should be positive large, which means the fuzzy logic controller should choose a larger bitrate. Complete fuzzy rules based on expert experience is presented in Table 2 - 4.

Defuzzier and Bitrate Selection: $f_{sub}(x)$ presents the degree of x belongs to the fuzzy subset, which is defined as,



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

Fig. 2. Membership of inputs and output: (**a**) buffer size (**b**) normalized throughput mean (**c**) normalized throughput standard deviation (**d**) aggressive degree

$$f_{\text{sub}}(O_i) = \begin{cases} \frac{\sum r_i}{3} & sub = NL \\ & i = \{1, 2, 10\} \\ \frac{\sum r_i}{4} & sub = NS \\ & i = \{4, 11, 13, 19\} \\ \frac{\sum r_i}{10} & sub = ZO \\ & i = \{3, 5, 6, 7, 8, 12, 14, 16, 20, 22\} \\ \frac{\sum r_i}{6} & sub = PS \\ & i = \{9, 15, 17, 21, 23, 25\} \\ \frac{\sum r_i}{4} & sub = PL \\ & i = \{18, 24, 26, 27\}. \end{cases} \tag{4}$$

There are many algorithms to make defuzzification, such as Center of Area (COA), Center of Maximization, and Mean of Maximization. In this work, we choose COA. Then aggressive degree O_i can be calculated as,

$$O_i = \frac{\sum_{F(O_i)} f_{F(o_i)}(O_i) * c(F(O_i))}{\sum_{F(o_i)} f_{F(o_i)}(O_i)}, \tag{5}$$

where $c(F(O_i)$ means the middle of the triangle's base.

We define two threshold O_{low} and O_{high}. Bitrate above and below the throughput mean can be found ($Q_{n-1} \leq \widehat{bm}_i < Q_n$). Finally, bitrate decision can be calculated as,

$$b_i = \begin{cases} Q_{n-1} & O_i < O_{low} \\ Q_{n+1} & O_i \geq O_{high} \\ Q_n & \text{else.} \end{cases} \tag{6}$$

Table 2. Linguistc Rules (buffer size = Small)

BM / BV	Negative	Zero	Postive
Small	NL ($r1$)	NL ($r2$)	ZO ($r3$)
Medium	NS ($r4$)	ZO ($r5$)	ZO ($r6$)
Large	ZO ($r7$)	ZO ($r8$)	PS ($r9$)

Table 3. Linguistc Rules (buffer size = Medium)

BM / BV	Negative	Zero	Postive
Small	NL ($r10$)	NS ($r11$)	ZO ($r12$)
Medium	NS ($r13$)	ZO ($r14$)	PS ($r15$)
Large	ZO ($r16$)	PS ($r17$)	PL ($r18$)

Table 4. Linguistc Rules (buffer size = Large)

BM / BV	Negative	Zero	Postive
Small	NS ($r19$)	ZO ($r20$)	PS ($r21$)
Medium	ZO ($r22$)	PS ($r23$)	PL ($r24$)
Large	PS ($r25$)	PL ($r26$)	PL ($r27$)

3. PERFORMANCE EVALUATION

Performance evaluations are carried out on five network traces defined by Twitch's ACM MMSys 2020 Grand Challenge (Twitch's Challenge for short).

3.1. Evaluation Environment

Video Player: Video player is forked from dash.js (v3.0.1) [15] and is modified to pre-request partially prepared video segments.

Server: The video streaming server is realized by python, which can stream a video segment before it is entirely received.

Video Traces: We use "BigBuckBunny" [16] sequence as the video trace, of which a section with 10 minutes duration is utilized and is divided into 1s duration segments. The video is encoded by H.264/AVC codec in three bitrate levels: 200kbps, 600kbps, 1000kbps.

Network Traces: Experiments are carried out on five network traces: Cascade, Intra Cascade, Spike, Slow Jitters, and Fast Jitters, which simulate various situations that a video player may encounter during playback.

QoE function: we defined our QoE model according to Yin [17] and Yi [18] as,

$$QoE = \sum_{i=1}^{I} (\alpha Q_i - \beta R_i - \gamma L_i) - \sum_{i=1}^{I} \mu \left| Q_i - Q_{i-1} \right|, \tag{7}$$

Q_i is the video quality which is represented by video bitrate in this work. R_i is the rebuffering time and L_i is the live latency. $\alpha, \beta, \gamma, \mu$ are penalty factors which reflect user's preferences. We choose the same penalty factors as in LoL [7].

ABR Algorithms: We evaluate the following ABR algorithms,

- FCTE: The proposed Fuzzy Logic Controller based ABR algorithm.

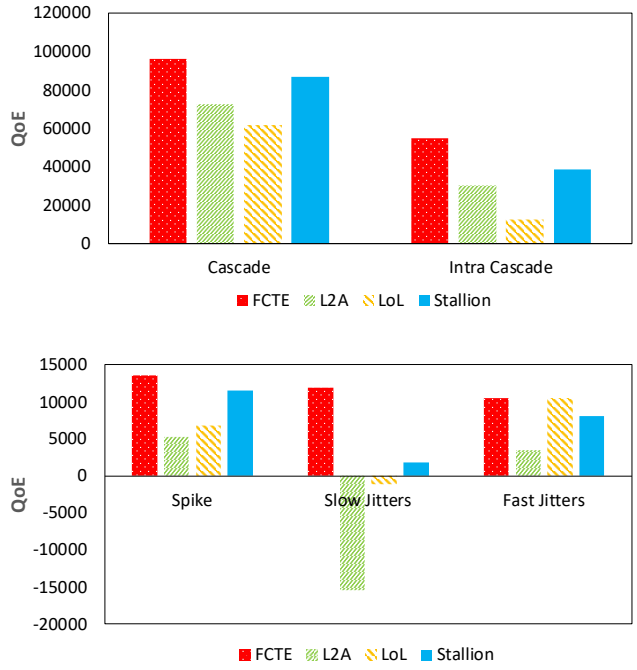


Fig. 3. QoE performance of FCTE and three representative ABR algorithms over five network scenarios

- L2A [8]: The winning solution of Twitch's Challenge on Adaptation Algorithm for Near-Second Latency, which is based on Online Convex Optimization.
- LoL [7]: The runner-up solution of Twitch's Challenge, which is a reinforcement learning-based algorithm.
- Stallion [9]: The 3rd place solution of Twitch's Challenge, which uses a sliding window to measure the mean and standard deviation of both the bandwidth and latency to improve upon bandwidth and latency estimation.

3.2. Evaluation Results

QoE performance of FCTE and three representative ABR algorithms are depicted in Figure 3. For each network trace, three simulations are conducted and the average QoE is calculated as the final result. Overall, FCTE achieves average QoE improvement by 129%, 338%, 127% with L2A, LoL, and Stallion, respectively. FCTE achieves the most outstanding performance under Slow Jitters. This may be because FCTE makes full use of throughput information and thus can adapt well in a fluctuating network environment.

Detailed QoE performance is presented in Table 5. Overall, L2A and LoL have a poor performance in Cascade, Intra Cascade, Spike, and Slow Jitters scenarios, because they have a long video freeze time compared with our FCTE and Stallion. In Cascade, FCTE outperforms Stallion by 10.49%. This is because FCTE improves average bitrate and reduces bitrate switches, and at the same time maintains the close latency and rebuffer performance. In Intra Cascade, FCTE outperforms Stallion by 42.51% where FCTE largely improves average bitrate and reduces rebuffer time, while maintaining the close latency performance. However, bitrate switches of FCTE are much larger than Stallion, which is because that Stallion keeps streaming in low bitrate without switching up for a long time when the network situation is improved. In Fast Jitters, FCTE performs slightly worse than LoL for a longer latency. This is because FCTE is much sensitive to video freezes and thus keeps a larger player buffer which increases latency.

Table 5. Streaming metrics of ABRs over five nerwork scenarios

ABR	Avg. bitrate	Avg. Latency	Total Rebuffer	Switches
CASCADE (150s)				
FCTE	684.64	1.56	2.50	5
L2A	690	1.80	21.70	17
LoL	672.53	1.25	33.23	10.86
Stallion	630	1.50	2.50	7.50
INTRA_CASCADE (135s)				
FCTE	526.67	1.71	8.80	13
L2A	460	2.30	21	19
LoL	521.58	2.04	53.24	5
Stallion	365	1.65	7	4
SPIKE (30s)				
FCTE	640.26	1.89	3.51	4
L2A	590	1.50	10.40	4
LoL	646.97	1.77	10.45	4
Stallion	540	2	2.50	4
SLOW JITTERS (30s)				
FCTE	665.02	1.74	2	14
L2A	460	2.30	21	19
LoL	642.82	2.13	15.44	10.71
Stallion	365	1.65	7	4
FAST JITTERS (11.6s)				
FCTE	1000	1.20	0	1
L2A	1000	1.04	7	1
LoL	1000	1	0	1
Stallion	835	1.40	0	2.50

4. CONCULSION

Chunked HTTP adaptive streaming facilitates low latency live streaming. However, existing ABR schemes need to be tweaked for low latency scenario. In this work, we propose FCTE: a Fuzzy Logic Controller based ABR algorithm for low latency live video streaming with chunked transfer encoding. We evaluate FCTE with three representative ABR algorithms: L2A, LoL, and Stallion over Twitch's five network traces. FCTE achieves average QoE improvement by 129%, 338%, 127% with L2A, LoL, and Stallion, respectively. Note that FCTE is only experimented in artificially defined network traces, where bitrate selections are relatively simple. In the future, we plan to investigate how to design an ABR for real network traces, especially for mobile networks.

5. REFERENCES

[1] "Low-on-latency. [online] https://github.com/nustreaming/acm-mmsys-2020-grand-challenge," .

[2] K. Durak, M. N. Akcay, Y. K. Erinc, B. Pekel, and A. C. Begen, "Evaluating the Performance of Apple's Low-Latency HLS," Sept. 2020, pp. 1–6.

[3] Mariem Ben Yahia, Yannick Le Louedec, Gwendal Simon, Loutfi Nuaymi, and Xavier Corbillon, "HTTP/2-based Frame Discarding for Low-Latency Adaptive Video Streaming," *ACM TOMCCAP*, vol. 15, no. 1, pp. 18:1–18:23, Feb. 2019.

[4] V. Swaminathan and S. Wei, "Low latency live video streaming using HTTP chunked encoding," in *IEEE 13th MMSP*, 2011, pp. 1–6.

[5] Jeroen van der Hooft, Cedric De Boom, Stefano Petrangeli, Tim Wauters, and Filip De Turck, "An HTTP/2 push-based framework for low-latency adaptive streaming through user profiling," in *NOMS*, 2018, pp. 1–5.

[6] Yunlong Li, Shanshe Wang, Xinfeng Zhang, Chao Zhou, and Siwei Ma, "High efficiency live video streaming with frame dropping," in *ICIP*, 2020, p. 1226–1230.

[7] May Lim, Mehmet N. Akcay, Abdelhak Bentaleb, Ali C. Begen, and Roger Zimmermann, "When they go high, we go low: low-latency live streaming in dash.js with LoL," in *ACM MMsys*, 2020, pp. 321–326.

[8] Theo Karagkioules, Rufael Mekuria, Dirk Griffioen, and Arjen Wagenaar, "Online learning for low-latency adaptive streaming," in *ACM MMsys*, 2020, pp. 315–320.

[9] Craig Gutterman, Brayn Fridman, Trey Gilliland, Yusheng Hu, and Gil Zussman, "Stallion: video adaptation algorithm for low-latency video streaming," in *ACM MMsys*, 2020, pp. 327–332.

[10] "Iso/iec. 23000-19:2020 information technology – multimedia application format (mpeg-a) – part 19: Common media application format (cmaf) for segmented media. [online] available: https://www.iso.org/standard/79106.html," .

[11] "Akamai whitepaper. ultra-low-latency streaming using chunkedencoded and chunked-transferred cmaf. [online] available: https://www.akamai.com/us/en/multimedia/documents/white-paper/ low-latency-streaming-cmaf-whitepaper.pdf," .

[12] "Dash-if/dvb report on low-latency live service with dash, [online] https://dashif.org/docs/report.

[13] "Enabling low-latency hls, [online] https://www.wowza.com/blog/apple-low-latency-hls," .

[14] Abdelhak Bentaleb, Christian Timmerer, Ali C. Begen, and Roger Zimmermann, "Bandwidth prediction in low-latency chunked streaming," in *NOSSDAV*, Amherst, Massachusetts, 2019, pp. 7–13.

[15] "dash.js, [online] https://github.com/dash-industry-forum/dash.js," .

[16] "Bigbuckbunny, [online] https://peach.blender.org/," .

[17] "The acm multimedia 2019 live video streaming grand challenge," 2019.

[18] Xiaoqi Yin, Abhishek Jindal, Vyas Sekar, and Bruno Sinopoli, "A control-theoretic approach for dynamic adaptive video streaming over http," in *ACM SIGCOMM*, 2015, vol. 45, pp. 325–338.