Evaluating the Impact of Playout Buffer Policies on Cloud Gaming QoE

Xiaokun Xu
Computer Science
Worcester Polytechnic Institute
Worcester, USA
xxu11@wpi.edu

Mark Claypool
Computer Science
Worcester Polytechnic Institute
Worcester, USA
claypool@wpi.edu

***Abstract*—Cloud gaming relies upon smooth delivery of frames and low delay for a good player Quality of Experience (QoE). While playout buffering has long been used in traditional streaming systems – e.g., streaming video and VoIP – to smooth out variations in delay and available bandwidth, algorithms for playout buffering for cloud game streams are under-researched. In particular, QoE models for cloud-based game streams differ from those for traditional media, suggesting algorithms that have been widely used and proposed may need to be adjusted for cloud gaming. This paper investigates playout buffer policies on cloud gaming QoE by employing a trace-driven simulation framework that allows for a head-to-head comparison of policies. By integrating established QoE models, our study quantifies how these technical parameters affect perceived gaming quality. Our results reveal that the effectiveness of a buffer policy is highly dependent on the network conditions, with adaptive strategies showing potential benefits in environments with high jitter, while more conservative policies may suffice under stable network conditions.**

***Index Terms*—cloud-game streaming, quality of experience, playout buffer.**

I. Introduction

Cloud-based game streaming (cloud gaming) renders games on powerful remote servers and streams them to low-end devices, offering access to a wide range of game titles without the need for the game client to have high-end hardware or do game installation. However, network congestion, variable delays, and fluctuating bandwidth can cause delayed responses and frame stuttering, degrading QoE. To counteract this, client-side playout buffers are used to smooth out network-induced jitter by temporarily storing frames, then delivering them evenly-paced as long as there are frames in the buffer. This technique introduces a trade-off: while a buffer can help absorb network-induced frame jitter and reduce the frequency of playback interruptions – the larger the buffer, the smoother the playout – the buffer also adds to end-to-end latency – the larger the buffer, the less responsive the interaction.

Previous research on playout buffers has primarily focused on traditional media (e.g., video-on-demand streaming) and interactive audio/video (e.g., video conferencing). Studies in video streaming have shown that buffer-induced delay and playback interruptions equally degrade user experience [1]–[4], while research on interactive media has highlighted that the effects of delay and jitter depend on the nature of user interactions [5]–[7]. However, cloud gaming has distinct QoE requirements – where both high responsiveness and smooth playback are crucial – indicating a need for a focused investigation into how different playout buffer policies perform under varying network conditions.

In this paper, we explore how playout buffer policies affect cloud gaming QoE by analyzing delay and frame jitter using trace-driven simulation. We simulated diverse buffering scenarios with over 300 traces from real cloud gaming systems – with network conditions ranging from smooth to choppy – to compare policies head-to-head and determine optimal settings through brute-force exploration. Using established QoE models, we compare the trade-offs between buffering and frame interrupts, finding that adaptive strategies excel in high-jitter environments while conservative policies suffice under stable conditions. These insights can help inform system upgrades and network improvements and also guide developers in game designs and architectures that work well in cloud-based streaming environments for better overall QoE.

The remainder of this paper is organized as follows: Section II reviews related work on network performance and QoE in cloud gaming and streaming media; Section III outlines our methodological framework and experimental setup used to evaluate playout buffer policies; Section IV presents analysis of the performance metrics and discuss how different buffer policies affect QoE under various network conditions; Section V discusses the limitations of our study and potential avenues for future research; and Section VI summarizes our conclusions.

II. Related work

A. Jitter and Video Streaming

Prior research shows that packet-level delay jitter degrades QoE by disrupting video transmission and causing frame jitter. For example, Orosz et al. [8] linked QoS metrics such as jitter, packet loss, and reordering to Mean Opinion Score (MOS), while Guan-Ming et al. [9] noted that wireless networks often lack the stability for smooth streaming due to jitter. Rao et al. [10] further identified delay jitter as a key factor in quality degradation. However, these findings may not fully apply to gaming, which demands higher interactivity and real-time responsiveness.

B. Playout Buffer Policies and Video Streaming

Playout buffers in video streaming smooth packet arrival variations and reduce playback interruptions by temporarily storing frames. Strategies range from fixed buffers – which trade off latency against interruption risk – to adaptive approaches that adjust buffer size in real time to minimize stutter without excessive delay [4], [11]–[14]. These studies highlight the delicate balance required to optimize QoE, and that static solutions are less suited for interactive environments.

C. Playout Buffer Policies in Cloud-based Game Streaming

Traditional video streaming buffers smooth playback by storing data, but may not suit cloud gaming where even slight delays disrupt real-time feedback. Balancing jitter absorption with low latency is needed for cloud gaming, as games and other forms of interactive media have degraded QoE under high delays. Although some studies have examined network impairments in cloud gaming [15], [16], systematic investigations into tailored playout buffer policies are scarce, underscoring the need for detailed analysis of buffering strategies that maintain low latency while ensuring smooth frame delivery.

In our work, we address this gap by employing a trace-driven playout buffer simulator which allows us to directly compare a variety of buffering strategies under the exact same network conditions. Our evaluation uses cloud gaming QoE metrics to compare tradeoffs that are specific to cloud-based game streams.

III. Methodology

To investigate the effects of playout buffer policies on cloud game QoE, we took traces gathered from a user study: participants used an open source cloud-game streaming system and played several different commercial games over a range of induced network perturbations, all while the system gathered frame playout traces. Subsequently, these traces were fed into our playout buffer simulator that provided key metrics – delay and interrupts – for policy evaluation.

The overall methodology is as follows:

(1) **Select games.** While game genres provide a broad categorization, they are insufficient for assessing QoE, which also depends on visual effects (spatial/temporal complexity), camera perspective, and interactivity [17]–[20]. To address these factors, we selected four games with diverse camera types and spatial/temporal characteristics: Bloons Tower Defense 6 (BTD6) [21] (Figure 1d) is a top-down tower defense game; Hollow Knight [22] (Figure 1c) is a 2D side-scrolling platformer; Hades [23] (Figure 1b) is a rogue-like fighting game with an isometric view; and Counter-Strike: Global Offensive (CS:GO) [24] (Figure 1a) is a first-person shooter game with a training course. In each game, participants completed a standardized mission – positioning towers in BTD6, progressing through a tutorial in Hollow Knight, engaging in continuous combat in Hades, and navigating the training course in CS:GO – to ensure a consistent basis for evaluating QoE across varied interactive environments.

(a) CS:GO (b) Hades

(c) Hollow Knight (d) Bloons TD 6

Fig. 1: Screenshots of games selected for the four user studies (S1 to S4).

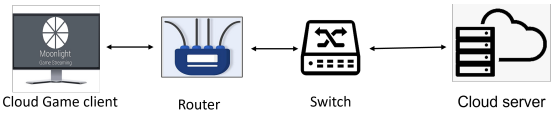

Fig. 2: Testbed for user study.

(2) **Setup testbed.** A laboratory setting was used to isolate participants and their game systems from uncontrolled variation, depicted in Figure 2. We built a cloud gaming setup using the open-source systems Moonlight [25] and Sunshine [26], with Sunshine as the host and encoder, and Moonlight as the decoding client. The system streamed at 60 f/s and 1080p, with controlled network conditions to study their impact on performance. All servers and clients ran on PCs exceeding Moonlight and Sunshine's recommended specifications to ensure that any frame jitter or delay resulted solely from network conditions. The cloud-game client was a Windows 10 PC (Intel i7 eight-core @ 2.0 GHz, 64 GB RAM, Gb/s Ethernet, 1920x1080 LED monitor at 60 Hz) connected via Moonlight, while the server had identical hardware and streamed using Sunshine. A Raspberry Pi 4 (1.5 GHz quad-core, 8 GB RAM, Ubuntu 20.04 LTS, Linux kernel 5.4) configured with `netem` acted as a network router between the client and server through a Gb/s switch. With both systems on the same LAN, baseline ping round-trip times were consistently around 1 ms.

(3) **Induce frame jitter.** Frame playout interrupts, quantified by their magnitude and frequency, were controlled using a Raspberry Pi router configured with netem [27]. Custom distribution files manipulated packet delays to achieve the desired

TABLE I: Target frametime jitter conditions.

Parameters	LAN	Low	High
frequency (/s)	0	0.5	3.5
magnitude (ms/s)	0	20	150
delay (ms)	1	30	100

TABLE II: LLR policy simulation results.

LLR Settings		Low Jitter			High Jitter		
Low	**High**	**IF**	**IM**	**Delay**	**IF**	**IM**	**Delay**
1	1	0.283	18.8	17.6	3.375	132.7	18.2
1	2	0.283	18.8	17.6	3.375	132.7	18.2
1	**3**	**0.283**	**18.8**	**17.7**	**3.253**	**129.6**	**20.6**
1	4	0.283	18.8	17.7	3.219	128.8	22.9
1	5	0.283	18.8	17.8	3.193	128.1	25.1
1	6	0.283	18.8	18.2	3.169	127.5	27.4
2	2	0.265	18.5	18.0	3.170	127.4	22.9
2	3	0.265	18.5	18.0	3.170	127.4	22.9
2	4	0.265	18.5	18.1	3.131	126.5	25.3
2	5	0.265	18.5	18.2	3.116	126.1	27.5
2	6	0.265	18.5	18.3	3.100	125.6	29.7
3	3	0.256	18.1	19.8	3.094	125.1	28.0
3	4	0.256	18.1	19.8	3.094	125.1	28.0
3	5	0.256	18.1	19.9	3.071	124.5	30.5
3	6	0.256	18.1	19.9	3.061	124.2	32.7
4	4	0.245	17.8	22.2	3.044	123.5	33.2
4	5	0.245	17.8	22.2	3.044	123.5	33.2
4	6	0.245	17.8	22.2	3.029	123.1	35.7
5	5	0.236	17.6	25.6	3.013	122.4	38.5
5	6	0.236	17.6	25.6	3.013	122.4	38.5
6	6	0.227	17.3	29.4	2.985	121.4	43.8

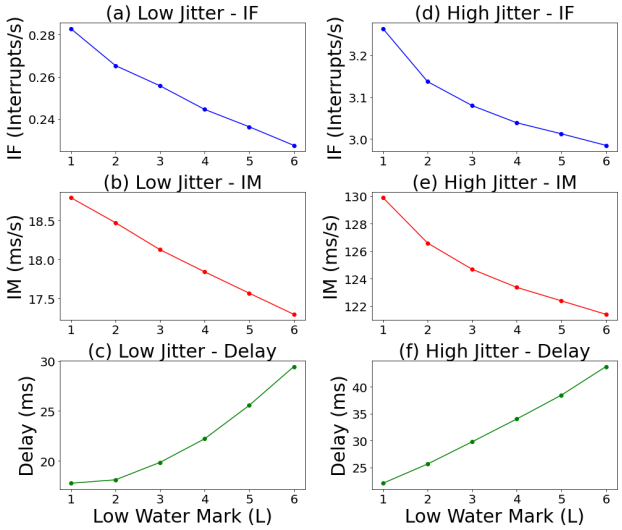

Fig. 3: LLR buffer policy results: fixed high water marker (H=6), low water mark in range 1-6.

interrupt characteristics, while Presentmon [28] recorded the time between displayed frames during gameplay.

(4) **Select parameters.** The study used two levels of frame jitter – low and high – determined through pilot tests of various interrupt frequency and magnitude combinations (see Table I).

(5) **Recruit users and conduct study.** The study was approved by our University's Institute Review Board (IRB), and volunteers were recruited via university mailing lists and game-centric groups. Each 50-second round contributed to an hour-long session per participant, who received $10 and were eligible for class credit, as appropriate. Participants signed consent forms, familiarized themselves with the setup through practice sessions, and then played short rounds for each game condition while the system recorded frame display timings. The order of games and frame jitter conditions was randomly shuffled for each participant, who completed all rounds for one game before moving to the next. Participants could pause between rounds as needed and were free to quit at any time.

(6) **playout buffer simulation.** After each gameplay session, frame traces captured by Presentmon were converted into input files for our playout buffer emulator, *Cushion*, which simulates frame arrival using inter-frame timing data. By comparing these simulations with subjective QoE ratings, we isolate the effects of playout buffer policies on cloud gaming performance. We simulated two policies: the Low Latency Video Renderer Algorithm (LLR) [29] – adapted from WebRTC used for Google's Stadia system [30] and other streaming-media players – and Queue Monitoring (QM) [31], a videoconferencing-centric approach balancing interrupts and delay.

For LLR, Cushion uses a watermark-based method with three parameters: a low water mark (L) to trigger playout, a high water mark (H) beyond which extra frames are discarded, and a target frame rate (F). The system starts in "fill" mode until L is reached, then switches to fixed-rate playout; if the buffer exceeds H, it enters "drain" mode until it drops back to L. We tested L and H values from 1 to 6.

The QM policy employs a "clawback" mechanism where late frames build up the buffer until persistent excess triggers aggressive discarding. Each buffer element has a threshold (in frame times) that decreases with buffer size using a fixed decay factor of 2. A counter for each position is incremented every frame, and if any counter exceeds its threshold, a frame is discarded and all counters are reset. We tested thresholds of 60, 120, 600, 900, 1800, and 3600.

(8) **Analyze data.** Our data analysis focuses on using the objective performance metrics derived from the playout buffer simulations – playback delay, interruption frequency, and interruption magnitude – as inputs to previously-derived QoE models.

IV. Analysis

This section examines the impact of playout buffer policies on jitter and delay (Section IV-A), followed by a comparison of the QoE outcomes across different playout buffer strategies and network conditions (Section IV-B).

A. Jitter and Delay

Frametime jitter, which causes playback interrupts (e.g., gaps larger than the frame time – such as 60 Hz with a frametime of 16.7 ms – will have a gap for frametimes larger than 33.3 ms), is used as an indicator of visual quality in streaming video [4]. Interrupts can be measured by both their frequency and magnitude. Interrupt frequency (IF) is the number of frame gaps per second, and interruption magnitude (IM) is the total extra delay per second.

TABLE III: QM policy simulation results.

QM Policy	Low Jitter			High Jitter		
Settings	IF	IM	Delay	IF	IM	Delay
60	0.261	18.3	19.1	3.009	122.1	44.6
120	0.259	18.2	19.6	2.992	121.5	49.3
600	**0.256**	**18.1**	**20.4**	**2.967**	**120.4**	**62.9**
900	0.256	18.1	20.6	2.965	120.4	63.5
1800	0.256	18.1	20.9	2.960	120.1	68.3
3600	0.255	18.1	21.1	2.957	119.8	73.0

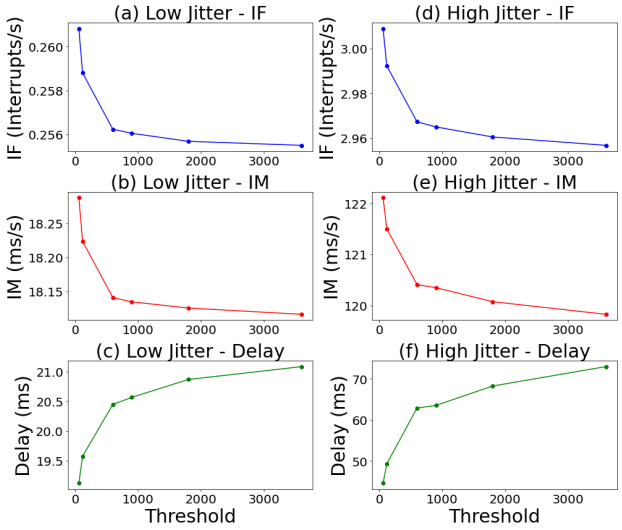

Fig. 4: QM policy simulation results.

1) LLR Performance Across Different Buffer Settings: Since consistent delay does not impact video smoothness, we focus on the LLR policy under low and high jitter. Table II shows how varying the Low Water Mark (L) and High Water Mark (H) affects IF, IM, and playback delay. The default LLR configuration uses a low water mark of 1 frame and a high water mark of 3 frames. From the table, we observe the following key trends:

1) Increasing the Low Water Mark (L) significantly reduces IF and IM – especially in high jitter – by ensuring sufficient buffering before playback starts, although this comes at the cost of increased delay.
2) Increasing the High Water Mark (H) further reduces IF and IM, but its impact is less pronounced compared to increasing L; the best performance in terms of interrupts only is achieved when both L and H are raised.
3) Both L and H, when increased, lead to higher playback delay, highlighting a trade-off between smoothness and interactivity.

Figure 3 shows the same data as Table II in graph form, focusing on $H = 6$ values (since L has a greater impact than H) with y-axes scaled separately for low and high jitter. Under low jitter, IF and IM are naturally low, so changes in L yields only minor improvements without much extra delay. In high jitter conditions, increasing L notably reduces IF and IM by absorbing frame delivery fluctuations; however, larger buffer sizes also lead to rapidly increasing playback delay, making balancing between delay and smoothness more critical in high-jitter environments.

2) Queue Monitoring Performance Across Different Buffer Settings: Table III shows the Queue Monitoring (QM) buffer policy evaluation. QM dynamically adjusts buffering based on two parameters: a threshold (the first number) that triggers frame discarding to prevent excessive queuing, and a decay factor (fixed at 2) that controls how aggressively the buffer is reduced. The table details the effects of different QM settings on IF, IM, and playback delay under both low and high jitter, with the default configuration indicated by the bold row (setting 600).

From the table, we observe the following key trends:

1) Raising the threshold allows more frames to accumulate, reducing IF and IM by increasing the buffer's tolerance to fluctuations, but it also increases playback delay. This effect is most pronounced under high jitter, where low thresholds lead to frequent interruptions. Although IF and IM decrease with higher thresholds, gains diminish beyond 60-120, indicating that larger buffers yield minimal additional interruption reduction while incurring greater delay.
2) Higher thresholds lead to longer playback delays as frames remain queued longer, especially under high jitter. For example, increasing the threshold from 60 to 3600 raises the delay from 44.6 ms to 73.0 ms, indicating that while higher thresholds reduce interruptions, they also increase latency, degrading user experience in latency-sensitive gaming.

Figure 4 shows the same data as Table III with y-axes scaled separately for low and high jitter. Under low jitter, threshold changes have little impact on IF and IM, and playback delay increases gradually. In contrast, under high jitter, higher thresholds markedly reduce interruptions but cause rapid increases in delay, illustrating a strong trade-off between smoothness and latency.

3) Comparison of LLR and QM Policies: The simulation results for both LLR and Queue Monitoring (QM) policies reveal key differences in how each buffering strategy affects IF, IM and playback delay under low and high jitter conditions.

1) In low jitter conditions, LLR slightly outperforms QM by achieving IF = 0.23 and IM = 17.3 ms versus QM's IF = 0.26 and IM = 18.1 ms, indicating smoother playout with fewer interruptions. Conversely, under high jitter, QM achieves marginally better performance with IF =

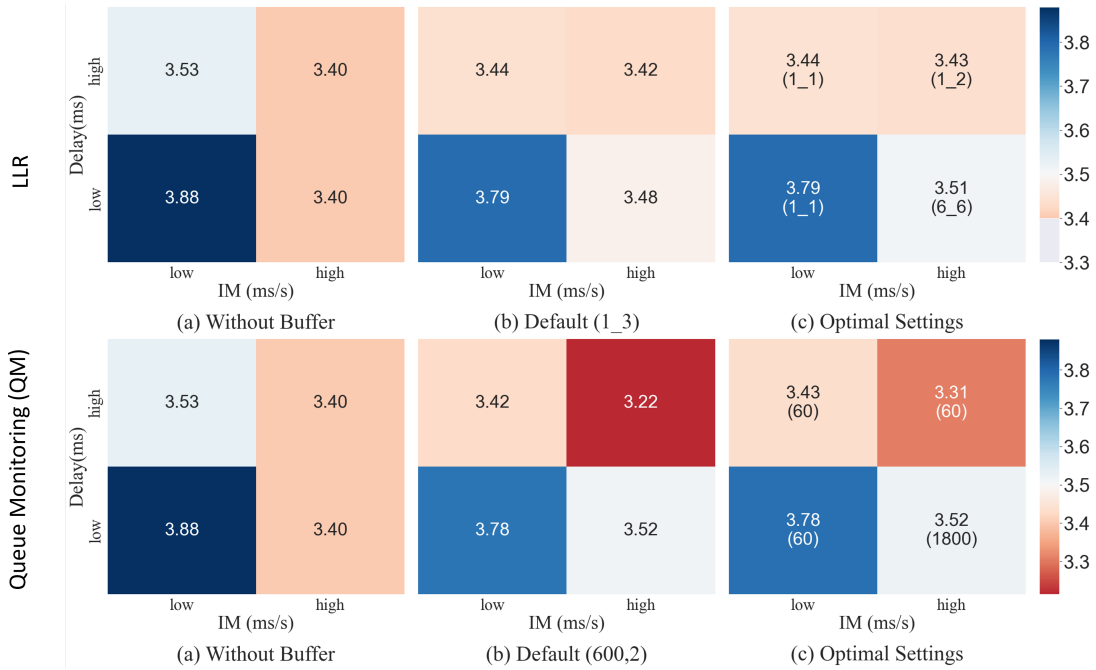

Fig. 5: QoE comparison of interruption magnitude (IM) and delay (ms) across different buffering strategies. (a) Without a buffer, (b) buffer policy with default settings, (c) buffer policy with optimal settings. For IM low and high, delay low and high, refer to Table I

2.96 and IM = 119.8 ms compared to LLR's IF = 2.99 and IM = 121.4 ms, suggesting that QM handles severe jitter slightly more effectively.

2) One key difference between the policies is playback delay. Under low jitter, QM achieves a significantly lower delay (21.1 ms) compared to LLR (29.4 ms), making it more efficient for latency-sensitive cloud gaming. However, in high jitter conditions, LLR maintains a lower delay (43.8 ms) than QM (73.0 ms). This suggests that while QM can slightly reduce interruptions in high jitter, its aggressive buffer reduction causes frames to be retained longer, resulting in higher latency.
3) **I**n low jitter conditions: LLR is better at reducing interruptions, while QM significantly reduces delay. If the primary concern is minimizing interruptions, LLR is the better choice. However, if low latency is the priority, QM performs much better.
   In high jitter conditions: QM still reduces interruptions slightly better than LLR, but at the cost of excessive delay. If maintaining low delay is more important, LLR is the superior option because it keeps playback delay significantly lower while still achieving reasonable smoothness.

B. QoE-based Results

Previous analysis compares interrupts and delay but does not quantify their impact on QoE. Xu and Claypool [32] found that interrupt frequency (IF) has little correlation with QoE in cloud gaming, so our evaluation focuses on interrupt magnitude (IM) and delay. We adopt their linear model (Q_{IM}) for IM:

$$Q_{IM} = -0.004 * IM + 4 \tag{1}$$

where Q_{IM} represents the QoE value (ranging from 1 to 5) based on IM (in ms/s).

For the delay model (Q_d), Liu and Claypool [33] proposed a similar linear model based on a controlled user study where participants experienced different levels of delay while playing cloud-based games. The model is given by:

$$Q_d = -0.004 * delay + 4 \tag{2}$$

where Q_d represents the QoE value (ranging from 1 to 5) based on delay (in milliseconds).

To compute the final QoE (Q_{final}) for each buffering configuration, we take the minimum value between Q_{IM} and Q_d:

$$Q_{final} = min(Q_{IM}, Q_d) \tag{3}$$

This approach reflects the worst-case impact principle, meaning that the most limiting factor between interrupt magnitude and delay determines the user's perceived QoE. If interrupt magnitude is high but delay is low, QoE will be constrained by IM, and vice versa.

Figure 5 uses heatmaps to depict the results. The heatmap illustrates the impact of different buffering policies (no buffer, default, and optimal settings) on QoE under varying jitter. The x-axis represents interruption magnitude (IM), and the y-axis shows playback delay (base delay plus buffering delay – 30 ms for low jitter and 100 ms for high jitter). Each square indicates

a specific jitter-delay combination and its corresponding QoE, with a blue-red gradient where blue signifies higher QoE and red lower QoE. Each large section corresponds to a different buffering policy:

1) Without Buffer: No playout buffer mechanism is applied.
2) Buffer policy with default settings: Using LLR (1_3) or QM (600), representing standard buffer settings.
3) Buffer policy with optimal settings: Using LLR (1_1, 6_6) or QM ((60), (3600)), which represents the best settings for each policy based on simulation results over all settings.

In low jitter, both buffering policies reduce QoE compared to no buffering – dropping from 3.88 (low delay) and 3.53 (high delay) – indicating that in stable networks, added latency harms QoE more than jitter reduction helps. In high jitter, LLR consistently improves QoE by mitigating interruptions without excessive latency, whereas QM's aggressive buffering introduces too much delay, resulting in a QoE drop.

When comparing default and optimal settings, buffering effectiveness differs between QM and LLR. QM generally introduces excessive delay, making a minimal buffer (setting 60) optimal for reducing delay while mitigating jitter. In contrast, LLR's default setting offers a balanced trade-off between reducing interruptions and limiting delay, making it more reliable in unpredictable network conditions, though further fine-tuning could improve performance under specific jitter and delay scenarios.

In summary, the LLR policy generally enhances QoE for cloud gaming in jittery conditions by balancing smoothness and responsiveness—critical for interactive gameplay—while effectively reducing interruptions with minimal added delay. In contrast, the QM policy, designed for video streaming where uninterrupted playback outweighs low latency, tends to introduce excessive delay in cloud gaming, especially under high jitter, thereby lowering QoE. This difference stems from their intended applications; LLR was optimized for Google's cloud gaming platform to manage network variations and latency constraints, whereas QM prioritizes buffering stability. As a result, LLR is the more effective choice for maintaining good QoE for cloud gaming in jittery network environments.

V. Limitations and Future Work

One limitation of our study is the simplified QoE model used in analysis, which defines final QoE as the minimum of IM and delay. While this captures the dominant factor, it overlooks possible confounding effects when both jitter and delay are high, or mitigating effects when one is high and the other low. Future work should develop a more comprehensive model with weighted, nonlinear interactions between delay and jitter that considers subjective user tolerance and input latency.

Another limitation is that Cushion currently supports only two policies – LLR and QM – even though other strategies, such as JitBright [34], offer different trade-offs between delay and smoothness. Future work could expand Cushion to include additional buffer policies, enabling a broader evaluation of strategies and more comprehensive guidance for optimizing cloud gaming QoE.

Cushion currently lacks key frame support, which is crucial for handling frame drops and resynchronization in real-world scenarios. Without key frames, simulated buffering may not accurately reflect actual behavior, potentially affecting result accuracy. Future work should add key frame handling to improve simulation realism and better represent video degradation in cloud gaming.

VI. Conclusions

Cloud-based game streaming presents unique challenges compared to traditional gaming due to network-induced frame jitter and delay. While playout buffer policies are commonly used in video streaming to smooth playback, their role in cloud gaming is less understood. This study examines how different policies affect QoE under various network conditions by simulating LLR and Queue Monitoring (QM) buffering policies with the Cushion framework, analyzing their effectiveness in mitigating jitter while minimizing delay.

Our results reveal that buffering strategies significantly affect QoE, with their impact depending on the network conditions. In low jitter scenarios, both LLR and QM tend to decrease QoE due to the additional delay introduced by buffering. However, in high jitter conditions, buffering can improve QoE, especially when the added delay remains minimal. The LLR buffer policy, designed specifically for cloud gaming, strikes a better balance between smoothness and interactivity compared to QM, originally developed for video streaming, which introduces excessive buffering delay, negatively affecting gameplay in high-delay conditions.

When comparing default and optimal buffer settings, QM's threshold of 60 performs best in most scenarios as it introduces the least delay, while LLR's default setting of 1_3 provides a more balanced trade-off, making it a reliable choice across different network conditions. However, for extreme jitter conditions, optimized LLR settings outperform its default configuration, demonstrating the potential for adaptive buffering strategies.

Overall, playout buffer policies play a crucial role in cloud gaming QoE, influencing how well the system balances smooth frame delivery and interactive responsiveness. Our findings suggest that cloud gaming platforms should carefully tune buffering mechanisms based on network conditions and even based on game genre to achieve better QoE. Future research could further explore adaptive buffering strategies that dynamically adjust settings based on real-time network variations and game type, leading to a better QoE for cloud gaming under a wide-range of network conditions.

REFERENCES

[1] L. Zhang, L. Zheng, and K. Soo Ngee, "Effect of Delay and Delay Jitter on Voice/Video over IP," *Computer Communications*, vol. 25, no. 9, pp. 863–873, 2002. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0140366401004182

[2] A. Tatematsu, Y. Ishibashi, N. Fukushima, and S. Sugawara, "QoE Assessment in Haptic Media, Sound and Video Transmission: Influences of Network Latency," in *Proceedings of the IEEE International Workshop Technical Committee on Communications Quality and Reliability (CQR)*, Vancouver, British Columbia, Canada, 2010, pp. 1–6.

[3] M. Claypool and J. Tanner, "The effects of jitter on the peceptual quality of video," in *Proceedings of the Seventh ACM International Conference on Multimedia (Part 2)*, Orlando, FL, USA, 1999, p. 115–118. [Online]. Available: https://doi.org/10.1145/319878.319909

[4] J. Allard, A. Roskuski, and M. Claypool, "Measuring and Modeling the Impact of Buffering and Interrupts on Streaming Video Quality of Experience," in *Proceedings of the 18th International Conference on Advances in Mobile Computing & Multimedia (MoMM)*, Chiang Mai, Thailand, Dec. 2020.

[5] H. Ahmadi, S. Khoshnood, M. R. Hashemi, and S. Shirmohammadi, "Efficient Bitrate Reduction using a Game Attention Model in Cloud Gaming," in *Proceedings of the IEEE International Symposium on Haptic Audio Visual Environments and Games (HAVE)*, Ottowa, ON, Canada, 2013, pp. 103–108.

[6] M. Manzano, J. A. Hernández, M. Urueña, and E. Calle, "An Empirical Study of Cloud Gaming," in *Proceeding of the 11th Annual Workshop on Network and Systems Support for Games (NetGames)*, 2012, pp. 1–2.

[7] M. Claypool, D. Finkel, A. Grant, and M. Solano, "Thin to Win? Network Performance Analysis of the OnLive Thin Client Game System ," in *Proceedings of the 11th ACM Network and System Support for Games (NetGames)*, Venice, Italy, Nov. 2012.

[8] P. Orosz, T. Skopkó, Z. Nagy, P. Varga, and L. Gyimóthi, "A Case Study on Correlating Video QoS and QoE," in *IEEE Network Operations and Management Symposium (NOMS)*, Krakow, Poland, 2014, pp. 1–5.

[9] S. Guan-Ming, S. Xiao, B. Yan, W. Mea, A. V. Vasilakos, and H. Wang, "QoE in Video Streaming over Wireless Networks: Perspectives and Research Challenges," *Wireless Networking*, 2015.

[10] N. Rao, A. Maleki, F. Chen, W. Chen, C. Zhang, N. Kaur, and A. Haque, "Analysis of the Effect of QoS on Video Conferencing QoE," in *15th International Wireless Communications & Mobile Computing Conference (IWCMC)*, 2019, pp. 1267–1272.

[11] K. Fujimoto, S. Ata, and M. Murata, "Adaptive Playout Buffer Algorithm for Enhancing Perceived Quality of Streaming Applications," *Telecommunication Systems*, vol. 25, no. 3–4, p. 259–271, Mar. 2004.

[12] 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.

[13] M. Yuang, S. Liang, Y. Chen, and C. Shen, "Dynamic video playout smoothing method for multimedia applications," in *Proceedings of ICC/SUPERCOMM '96 - International Conference on Communications*, vol. 3, 1996, pp. 1365–1369 vol.3.

[14] Y. Cinar, P. Pocta, D. Chambers, and H. Melvin, "Improved Jitter Buffer Management for WebRTC," *ACM Trans. Multimedia Comput. Commun. Appl.*, vol. 17, no. 1, Apr. 2021. [Online]. Available: https://doi.org/10.1145/3410449

[15] H. S. Rossi, N. Ögren, K. Mitra, I. Cotanis, C. Åhlund, and P. Johansson, "Subjective Quality of Experience Assessment in Mobile Cloud Games," in *IEEE Global Communications Conference (GLOBECOM)*, Rio de Janeiro, Brazil, Dec. 2022, pp. 1918–1923.

[16] M. Suznjevic, I. Slivar, and L. Skorin-Kapov, "Analysis and QoE Evaluation of Cloud Gaming Service Adaptation under Different Network Conditions: The Case of NVIDIA GeForce NOW," in *Eighth International Conference on Quality of Multimedia Experience (QoMEX)*, Lisbon, Portugal, 2016, pp. 1–6.

[17] S. Schmidt, S. Zadtootaghaj, and S. Moller, "Towards the Delay Sensitivity of Games: There is More than Genres," in *Proceedings of the International Conference on Quality of Multimedia Experience (QoMEX)*, Erfurt, Germany, May 2017.

[18] M. Claypool, "Motion and Scene Complexity for Streaming Video Games," in *Proceedings of the 4th ACM International Conference on the Foundations of Digital Games (FDG)*, Florida, USA, Apr. 2009.

[19] M. Claypool and K. Claypool, "Perspectives, Frame Rates and Resolutions: It's all in the Game," in *Proceedings of the 4th ACM International Conference on the Foundations of Digital Games (FDG)*, Florida, USA, Apr. 2009.

[20] S. S. Sabet, S. Schmidt, S. Zadtootaghaj, C. Griwodz, and S. Möller, "Delay Sensitivity Classification of Cloud Gaming Content," in *Proceedings of the International Workshop on Immersive Mixed and Virtual Environment Systems (MMVE)*, Istanbul, Turkey, 2020.

[21] Wikipedia contributors, "Bloons TD 6," 2018, [Accessed 12-Sep-2023]. [Online]. Available: https://en.wikipedia.org/wiki/Bloons_TD_6

[22] ——, "Hollow Knight," 2017, [Accessed 12-Sep-2023]. [Online]. Available: https://en.wikipedia.org/wiki/Hollow_Knight

[23] ——, "Hades (video game)," 2020, [Accessed 12-Sep-2023]. [Online]. Available: https://en.wikipedia.org/wiki/Hades_(video_game)

[24] ——, "Counter-strike: Global Offensive (CS:GO)," [Accessed 28-Nov-2023]. [Online]. Available: https://en.wikipedia.org/wiki/Counter-Strike:_Global_Offensive

[25] C. Gutman, "Moonlight PC v4.3.1," Nov. 2022, online: https://github.com/moonlight-stream.

[26] LizardByte, "Sunshine," Nov. 2022, online: https://github.com/LizardByte/Sunshine.

[27] Wikipedia contributors, "Network emulation," 2022, [Accessed 20-Oct-2022]. [Online]. Available: https://en.wikipedia.org/wiki/Network_emulation

[28] GameTechDev, "PresentMon V1.8.0," May 2022, online: https://github.com/GameTechDev/PresentMon.

[29] T. C. Authors, "low_latency_video_renderer_algorithm," Jan. 2020, online: https://tinyurl.com/mpe5sucp.

[30] Google, "Google Stadia Post-game Survey," 2020, (Accessed Oct 15, 2021). [Online]. Available: https://stadia.google.com/

[31] D. L. Stone and K. Jeffay, "An empirical study of delay jitter management policies," *Multimedia Syst.*, vol. 2, no. 6, p. 267–279, Jan. 1995. [Online]. Available: https://doi.org/10.1007/BF01225244

[32] X. Xu and M. Claypool, "User Study-based Models of Game Player Quality of Experience with Frame Display Time Variation," in *Proceedings of the 15th ACM Multimedia Systems Conference*, Bari, Italy, 2024, p. 210–220. [Online]. Available: https://doi.org/10.1145/3625468.3647625

[33] S. Liu and M. Claypool, "The Impact of Latency on Navigation in a First-Person Perspective Game," in *Proceedings of the ACM CHI Conference on Human Factors in Computing Systems*, New Orleans, LA, USA, 2022. [Online]. Available: https://doi.org/10.1145/3491102.3517660

[34] Y. Zhao, Q. Wu, G. Lv, F. Yang, J. Zhang, F. Peng, Y. Liu, Z. Li, Y. Chen, H. Guo, and G. Xie, "JitBright: towards Low-Latency Mobile Cloud Rendering through Jitter Buffer Optimization," in *Proceedings of the 34th Edition of the ACM Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV)*, Bari, Italy, 2024, p. 36–42.