

Evaluating the Impact of Playout Buffer Policies on Cloud Gaming QoE

Xiaokun Xu

xxu11@wpi.edu

Worcester Polytechnic Institute

Mark Claypool

Worcester Polytechnic Institute

Research Article

Keywords: cloud-game streaming, quality of experience, playout buffer.

Posted Date: October 21st, 2026

DOI: <https://doi.org/10.21203/rs.3.rs-11223288/v1>

License:   This work is licensed under a Creative Commons Attribution 4.0 International License.

[Read Full License](#)

Additional Declarations: No competing interests reported.

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.



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

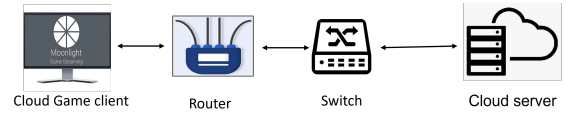


Fig. 2: Testbed for user study.

(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 interrupt characteristics, while Presentmon [28] recorded the

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: E-policy simulation results with initial buffer in range 1–11 frames.

E-policy Settings (initial buffer)	Low Jitter			High Jitter		
	IF	IM	Delay	IF	IM	Delay
1	0.232	15.8	30.8	1.93	104.1	347.2
3	0.215	15.1	32.9	1.92	103.0	351.8
5	0.196	14.5	38.5	1.90	102.5	358.2
7	0.178	14.0	46.8	1.89	101.9	365.8
9	0.153	13.5	56.8	1.87	101.5	373.5
11	0.139	13.0	69.0	1.87	101.0	381.9

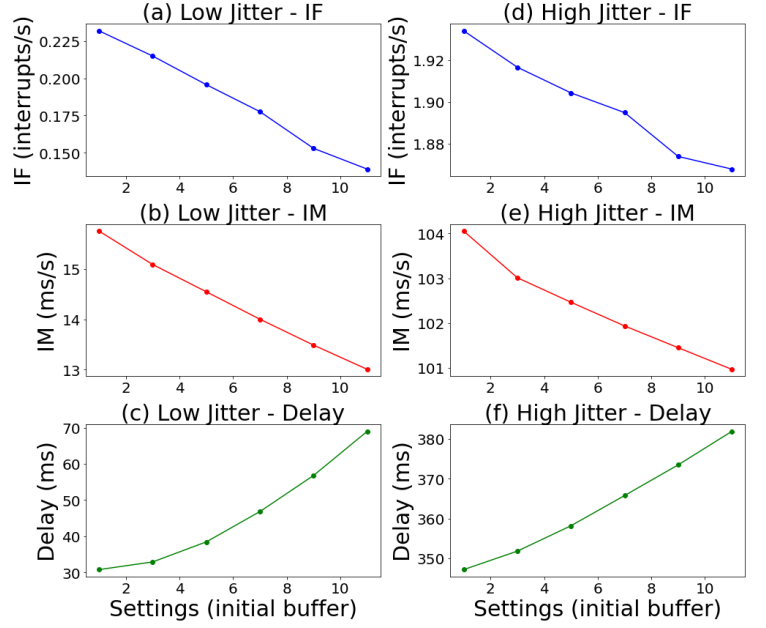


Fig. 3: E-policy simulation results with initial buffer in range 1–11 frames.

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 four policies: the I-policy baseline, the E-policy, Queue Monitoring (QM) [29], and the Low Latency Renderer Algorithm (LLR) [30], along with JitBright [31], an adaptive jitter-buffer strategy originally designed for cloud-rendering workloads.

As a baseline, we include a “no-buffer” configuration, referred to as the I-policy. This policy performs no active buffering or delay smoothing; instead, each frame is displayed

immediately upon arrival. The I-policy therefore represents an idealized low-latency pipeline that offers zero added playout delay but provides no protection against irregular frame arrivals. Under network jitter, this lack of buffering results in visible stutter, uneven motion, and occasional frame drops, making it a useful lower bound for understanding how more advanced buffering strategies trade latency for smoothness. All subsequent buffering approaches are evaluated relative to this baseline.

The E-policy maintains an initial buffer of frames and adjusts the playout schedule based on observed frame-time deviations. By accumulating an initial reservoir and then occasionally inserting extra delay, the E-policy attempts to absorb bursts of jitter while avoiding frequent frame drops. We evaluate a range of initial-buffer sizes (1–11 frames) to explore how the policy trades off interruption suppression against added delay.

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.

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.

TABLE III: QM policy simulation results.

QM Policy Settings	Low Jitter			High Jitter		
	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

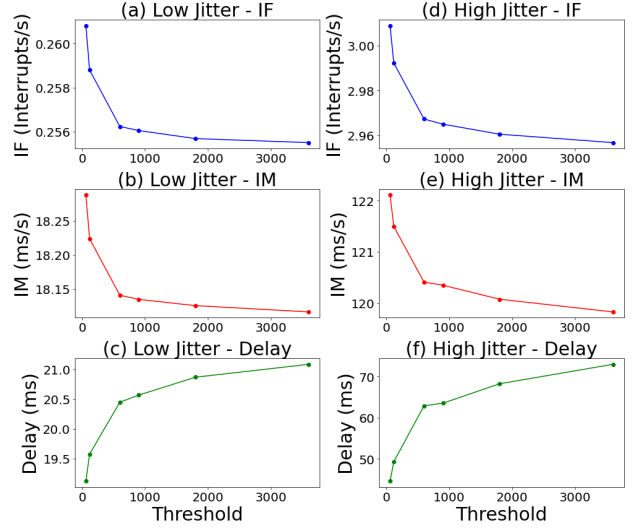


Fig. 4: QM policy simulation results.

The JitBright policy dynamically adjusts the playout buffer according to measured jitter and the estimated stability of incoming frames. Rather than relying on fixed watermarks or counter-based thresholds, JitBright models both short-term and long-term variation in inter-frame arrival times and selects a playout delay that minimizes the predicted risk of interruption while keeping latency low. The algorithm maintains a target buffer level computed from recent variance in frame arrivals and updates the playout position gradually using a smoothing factor that stabilizes adjustments over time. For our evaluation, we tested JitBright using scaling parameters (sp) of 0.5, 0.6, 0.7, 0.8, 0.9, and 1.0, each representing a different aggressiveness in adapting the playout delay. We also evaluated the algorithm under six network-capacity limits—15 Mb/s, 25 Mb/s, 35 Mb/s, 45 Mb/s, 55 Mb/s, and 65 Mb/s—to capture its behavior across constrained and well-provisioned network conditions.

(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.

1) *I-policy Baseline:* Before evaluating the buffering strategies, we establish a baseline using the I-policy, a no-buffer configuration in which each frame is displayed immediately upon arrival. As described earlier, this approach introduces no additional playout delay and therefore reflects the raw frame-timing behavior of the network. The corresponding interrupt frequency (IF), interrupt magnitude (IM), and baseline delay for each network condition are shown in Table I. These values represent the lower bound on smoothness, since the I-policy performs no smoothing or reordering and is fully exposed to fluctuations in frame arrival.

The simulations of E-policy, QM, LLR, and JitBright report only the *extra* playout delay added by the buffer algorithm. Thus, the total effective delay experienced by the player is the sum of the baseline delay from Table I and the additional buffering delay listed in the corresponding policy results.

2) *E-policy Performance Across Different Buffer Settings:* Table II and Figure 3 summarize the behavior of the E-policy as the initial buffer size increases from 1 to 11 frames. Under low jitter, larger initial buffers steadily reduce both IF and IM by absorbing more timing variability before frames are displayed. However, this improvement in smoothness comes at the cost of substantially increased delay: from roughly 31 ms at one frame up to nearly 69 ms at 11 frames. Under high jitter, IF and IM decrease only slightly as the buffer grows, while delay increases dramatically from 347 ms to almost 382 ms. Overall, the E-policy demonstrates that aggressive initial buffering can smooth frame delivery, but in interactive cloud gaming the resulting hundreds of milliseconds of delay are likely unacceptable. The smallest buffer size therefore represents the most reasonable operating point, and even there the additional delay is considerably larger than that incurred by the other policies.

TABLE IV: 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

3) *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

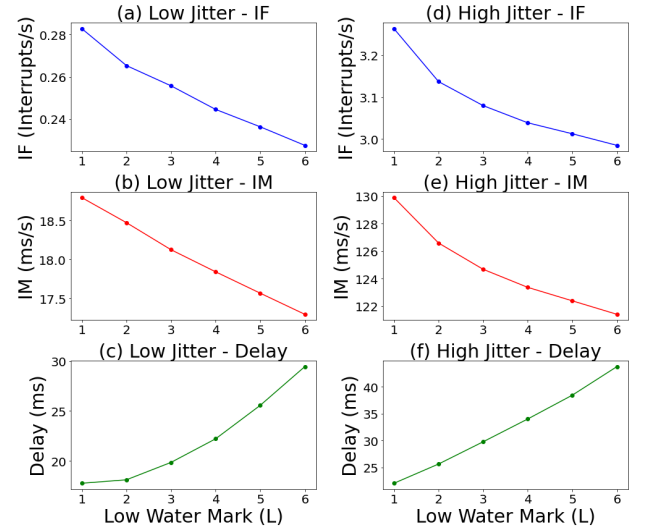


Fig. 5: LLR buffer policy results: fixed high water marker ($H=6$), low water mark in range 1-6.

between smoothness and latency.

4) *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 IV 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 5 shows the same data as Table IV 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.

5) *JitBright Performance Across Different Buffer Settings:* The JitBright policy differs from traditional buffer algorithms in that it dynamically adjusts its buffer delay based not only

TABLE V: JitBright policy simulation results with fixed capacity, sp in range 0.5–1.0.

JitBright (35 Mb/s) Settings (sp)	Low Jitter			High Jitter		
	IF	IM	Delay	IF	IM	Delay
0.5	0.218	17.0	34.1	2.968	120.6	49.3
0.6	0.218	17.0	34.1	2.968	120.6	49.3
0.7	0.227	17.3	29.6	2.978	121.2	46.3
0.8	0.236	17.6	25.7	2.999	121.9	43.3
0.9	0.245	17.8	22.5	3.018	122.7	40.1
1.0	0.256	18.1	20.3	3.044	123.6	37.0

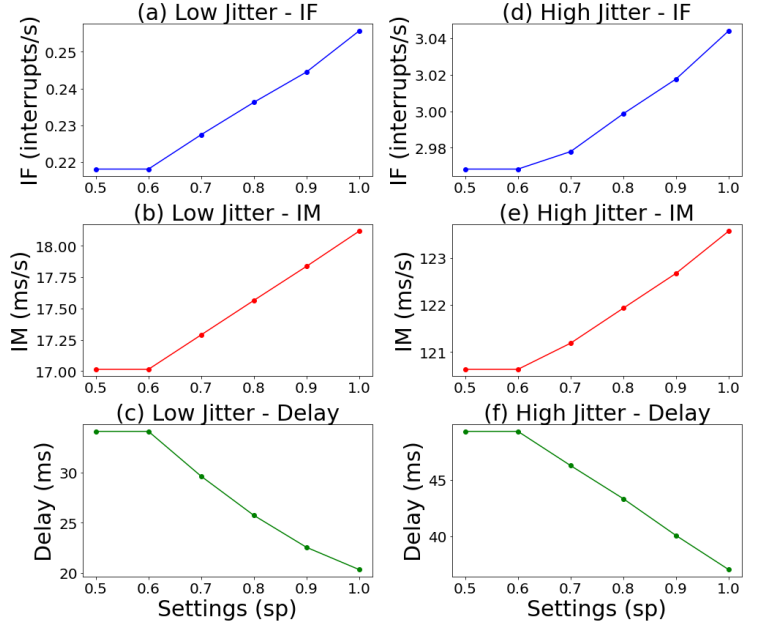


Fig. 6: JitBright policy simulation results with fixed capacity 35 Mb/s, sp in range 0.5–1.0.

on instantaneous frame arrivals, but also on longer-term statistical indicators such as observed jitter patterns and available streaming capacity. In a real cloud-gaming system, JitBright continuously updates its buffer target by monitoring changes in delivery variance and resource availability. However, because our simulation tool, Cushion, replays frame traces deterministically and cannot emulate real-time changes in network state or adaptive feedback loops, we evaluate JitBright using two complementary approaches that approximate its behavior.

JitBright with fixed capacity (35 Mb/s): Table V shows the performance of the JitBright playout buffer policy when the network capacity is fixed at 35 Mb/s. We select 35 Mb/s because prior measurements of several mainstream cloud-gaming platforms indicate that many commercial streams operate in the range of 25–35 Mb/s [32]. Under this fixed capacity, we vary the JitBright sp parameter from 0.5 to 1.0 to examine how its adaptive smoothing aggressiveness influences IF, IM, and playback delay, with the default configuration indicated by the bold row (sp 1.0).

From the table, we observe several key trends:

- 1) Increasing the sp value consistently lowers playback delay under both low and high jitter. Larger sp values make JitBright more permissive, reducing conservative buffering and enabling quicker playout. Under low jitter, delay drops from 34.1 ms at $sp=0.5$ to 20.3 ms at $sp=1.0$. A similar monotonic drop occurs under high jitter, from 49.3 ms to 37.0 ms.
- 2) Interrupt frequency increases gradually as sp increases. A smaller sp value results in stronger temporal smoothing, suppressing fluctuations at the cost of higher delay. As sp grows, the buffer intervenes less aggressively, leading to slightly more timing irregularities. Low-jitter

IF rises from 0.218 to 0.256, while high-jitter IF rises from 2.968 to 3.044.

- 3) Interrupt magnitude (IM) also increases with sp , especially under high jitter. We observe steady IM growth across all sp values. Under low jitter, IM increases from 17.0 to 18.1 ms/s; under high jitter, IM increases more noticeably from 120.6 to 123.6 ms/s.

Figure 6 visualizes the same trends with separate y-axis scaling. Across both jitter conditions, the results exhibit a clear and predictable trade-off: larger sp values reduce playback delay but incrementally increase IF and IM. This makes sp a direct control knob for balancing responsiveness and smoothness. The monotonic changes observed in all three metrics demonstrate that JitBright behaves consistently and controllably at a typical cloud-gaming bitrate such as 35 Mb/s.

JitBright with fixed sp (1.0): Table VI presents the performance of the JitBright playout buffer policy when the smoothing parameter is fixed at $sp = 1.0$ and the available network capacity varies between 15 Mb/s and 65 Mb/s. Unlike LLR and QM, JitBright explicitly incorporates network conditions—particularly throughput—into its adaptive buffering strategy. When capacity is high, JitBright reduces the internal buffer to preserve responsiveness; when capacity is constrained, it expands the buffer to absorb variability. By fixing sp at its most permissive value, this evaluation isolates how JitBright’s capacity-driven behavior affects IF, IM, and playback delay.

From the table, we observe several key trends:

- 1) Higher capacity leads to smaller JitBright buffer sizes, resulting in lower playback delay. As throughput increases, JitBright becomes more confident in the stability of frame arrivals and reduces buffering accordingly. This results in a distinct, steady reduction in delay: under

TABLE VI: JitBright policy simulation results with fixed sp, capacity in range 15–65 Mb/s.

JitBright (sp=1) Settings (capacity)	Low Jitter			High Jitter		
	IF	IM	Delay	IF	IM	Delay
15	0.218	17.0	34.1	2.968	120.6	49.3
25	0.227	17.3	29.6	2.978	121.2	46.3
35	0.256	18.1	20.3	3.044	123.6	37.0
45	0.265	18.5	18.6	3.076	124.9	34.2
55	0.283	18.8	18.3	3.133	126.6	32.1
65	0.298	19.0	17.1	3.268	128.6	30.3

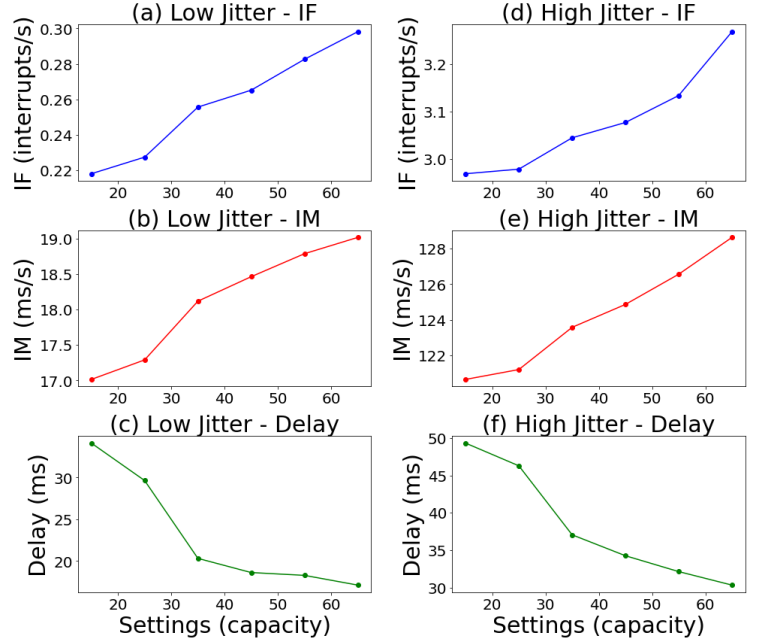


Fig. 7: JitBright policy simulation results with fixed sp, capacity in range 15–65 Mb/s.

low jitter, delay decreases from 34.1 ms at 15 Mb/s to 17.1 ms at 65 Mb/s; under high jitter, the decrease is from 49.3 ms to 30.3 ms.

- 2) Smaller buffers at high capacity cause IF to increase slightly. With reduced buffering, timing fluctuations that would otherwise be smoothed become more visible in the output stream. As a result, interrupt frequency rises gradually with capacity: from 0.218 to 0.298 under low jitter, and from 2.968 to 3.268 under high jitter.
- 3) Interrupt magnitude (IM) also rises with capacity, especially under high jitter. Larger buffers at low capacity suppress short-term timing spikes, but the smaller buffers used at high capacity allow more of these fluctuations to pass through. Consequently, IM grows from 17.0 to 19.0 ms/s (low jitter), and from 120.6 to 128.6 ms/s (high jitter).

Figure 7 visualizes the same trends, highlighting JitBright’s characteristic capacity-responsive behavior: larger capacity encourages more aggressive (lower-latency) playout with less smoothing, while smaller capacity triggers more conservative buffering that improves smoothness at the cost of delay. This trade-off is consistent and predictable across all tested conditions, demonstrating how JitBright leverages capacity awareness to balance responsiveness and stability in cloud game streaming.

6) *Comparison of Buffer Policies:* The simulation results across the I-policy baseline, E-policy, Queue Monitoring (QM), Low Latency Renderer (LLR), and JitBright highlight clear differences in how each playout buffer strategy balances smoothness (IF and IM) and responsiveness (playback delay) under both low and high jitter. These policies span a continuum

of design philosophies: I-policy provides no smoothing, E-policy relies on a large initial reservoir, LLR uses watermarks, QM applies threshold-based discarding, and JitBright adapts based on temporal statistics and inferred network capacity.

- 1) In low jitter conditions, all buffering strategies achieve similar QoE, even though E-policy reduces IF/IM the most. Under low jitter, the I-policy already experiences minimal interruptions; therefore, the QoE improvements from buffering are modest across all strategies. E-policy produces the lowest IF and IM—i.e., the smoothest playout—but its delay is substantially higher than the other policies. Because jitter is low, the reduced interruptions do not translate into noticeably higher QoE. As a result, E-policy, QM, LLR, and JitBright all achieve comparable QoE despite their differing internal behaviors. In this setting, the extra smoothness gained by E-policy does not compensate for the additional delay from a QoE perspective.
- 2) Under high jitter, E-policy provides the strongest smoothing but incurs the largest delay. When jitter is high, the I-policy suffers significantly from frequent, severe frame-time fluctuations. All buffering strategies improve IF and IM in this scenario, but E-policy produces the lowest interrupts of all methods. However, it does so by adding hundreds of milliseconds of buffering delay, far exceeding the delay introduced by QM, LLR, or JitBright. LLR and QM both reduce interruptions with more moderate delay costs, while JitBright provides consistent smoothing with the lowest delay among the adaptive strategies. Thus, E-policy yields the cleanest frametime delivery but at the highest latency penalty.

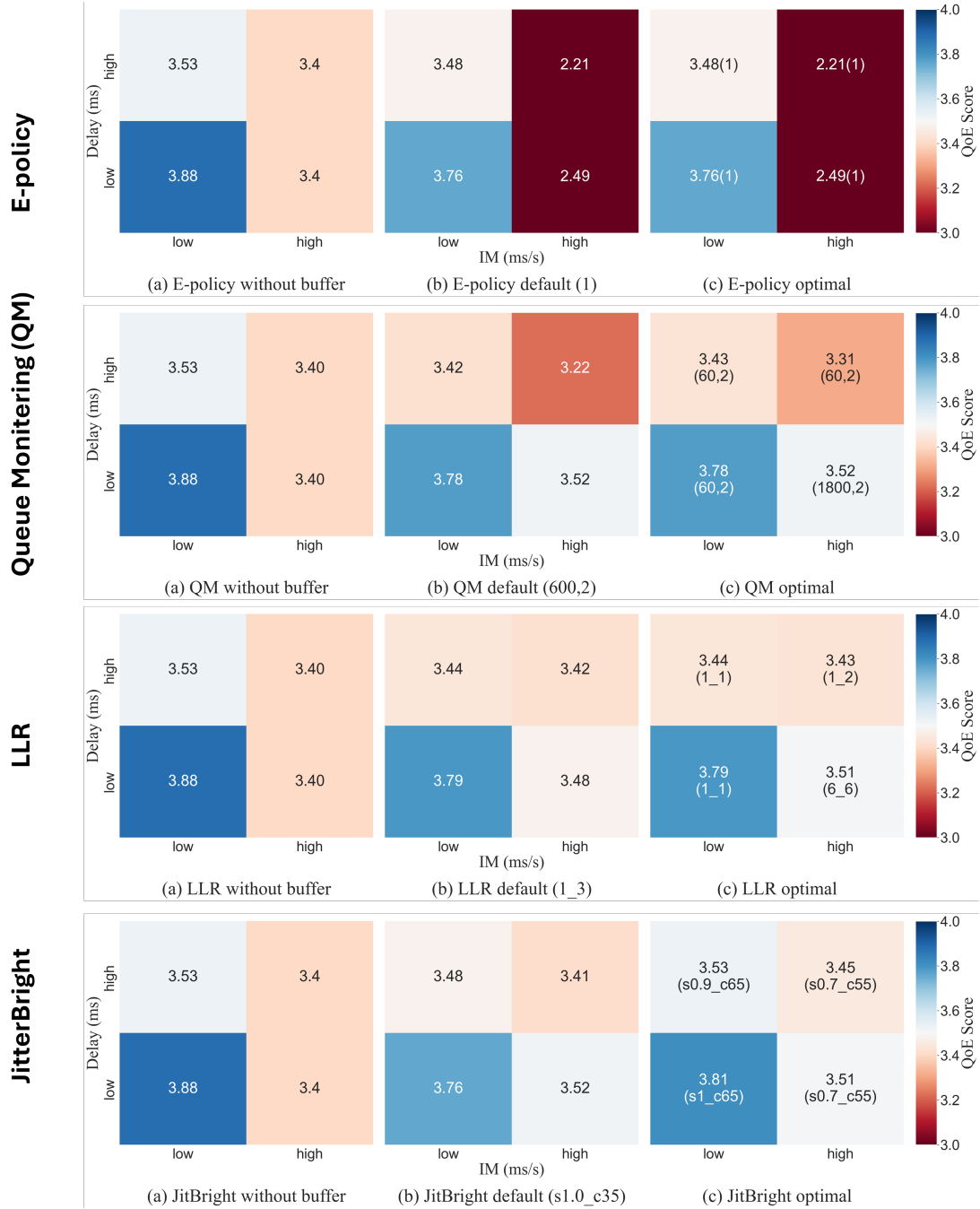


Fig. 8: QoE comparison across different buffering strategies.

- 3) Summary: I-policy is best for latency, E-policy is best for smoothing, LLR offers balanced performance, QM is delay-heavy, and JitBright is the most adaptable. In low jitter, buffering has little effect on QoE, and all policies—including E-policy—perform similarly. In high jitter, E-policy suppresses interrupts most effectively, LLR offers strong smoothing with moderate delay, and JitBright gives controllable and capacity-aware balancing of delay and smoothness. If minimal delay is essential,

the I-policy is optimal; if eliminating interrupts is the priority, E-policy is strongest; if overall QoE stability across diverse conditions is desired, JitBright is the most robust choice.

B. QoE-based Results

Previous analysis compares interrupts and delay but does not quantify their impact on QoE. Xu and Claypool [33] 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:

$$Q_{IM} = -0.004 * IM + 4 \quad (1)$$

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 [34] 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:

$$Q_d = -0.004 * delay + 4 \quad (2)$$

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 :

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

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 8 summarizes how QoE varies under different combinations of interruption magnitude (IM) and playback delay for the four buffering strategies evaluated in this chapter—E-policy, QM, LLR, and JitBright—each tested under three configurations: I-policy baseline (no buffer), default settings, and optimal settings derived from simulation data. The x-axis indicates IM (low or high), and the y-axis represents playback delay (low or high), where delay includes both baseline network delay and buffer-induced delay. The color scale ranges from red (low QoE) to blue (high QoE) and is kept consistent across all four strategies to enable direct comparison. This QoE analysis complements the earlier IF/IM/delay evaluation by showing how each buffering strategy resolves the fundamental trade-off between reducing interruptions (improving smoothness) and increasing buffering delay (hurting responsiveness). The heatmaps highlight how each policy navigates this trade-off under different network conditions.

1) *I-policy and E-policy*: In all four strategies, the leftmost heatmap corresponds to the I-policy (no buffer). Since no additional playout buffering is introduced, the QoE is determined entirely by the underlying network conditions. The highest QoE, 3.88, occurs when both delay and IM are low. Increasing delay while keeping IM low reduces QoE to 3.53, while high IM reduces QoE to 3.40 regardless of whether the delay is low or high. Thus, without buffering, either delay or interruption magnitude can become the dominant factor limiting QoE.

The E-policy introduces an initial buffer to reduce interruptions, but this can substantially increase playback delay. With the default setting, QoE remains relatively high when IM is low, reaching 3.76 under low delay and 3.48 under high delay.

However, under high IM, QoE drops sharply to 2.49 and 2.21. The optimal settings shown in the figure do not improve upon these values, indicating that the evaluated E-policy configurations cannot effectively resolve the delay-interruption trade-off under these conditions. In particular, although buffering may reduce interruption frequency or magnitude, the additional delay can become the dominant factor in the final QoE. Among the evaluated policies, E-policy therefore shows the greatest QoE degradation in the more challenging conditions.

2) *Queue Monitoring (QM)*: QM provides a better balance between interruption reduction and buffering delay than E-policy. With the default setting (600,2), QoE ranges from 3.22 to 3.78. Under low IM, the default configuration achieves QoE values of 3.42 and 3.78 for high- and low-delay conditions, respectively. Under high IM, QoE is 3.22 under high delay and 3.52 under low delay. Compared with E-policy, QM therefore avoids the severe QoE degradation observed in the high-IM conditions.

The optimal QM settings further illustrate the importance of adapting the buffering threshold to network conditions. A relatively small threshold (60,2) is selected for three of the four conditions, producing QoE values of 3.43, 3.31, and 3.78. For the low-delay/high-IM condition, a much larger threshold (1800,2) is selected, resulting in a QoE of 3.52. However, the differences between the default and optimal configurations are relatively small. This suggests that although tuning QM can improve the balance between interruption mitigation and delay, its QoE remains sensitive to the amount of buffering introduced.

3) *Low Latency Renderer (LLR)*: LLR provides consistently high QoE across the evaluated conditions. With its default setting (1_3), QoE ranges from 3.42 to 3.79. In particular, the default configuration achieves 3.44 and 3.79 under low IM and 3.42 and 3.48 under high IM. Compared with E-policy and QM, LLR avoids large QoE reductions while maintaining relatively high QoE across both delay and IM conditions.

The optimal LLR settings provide similar results, with QoE ranging from 3.43 to 3.79. Different watermark configurations are selected depending on the operating condition: (1_1) is selected for the low-IM conditions, while (1_2) and (6_6) are selected for the high-delay/high-IM and low-delay/high-IM conditions, respectively. The relatively small difference between the default and optimal results indicates that the default LLR configuration already provides a good balance between interruption reduction and added delay. Thus, LLR is comparatively insensitive to parameter tuning and maintains stable QoE across different network conditions.

4) *JitBright*: JitBright also maintains consistently high QoE across the evaluated conditions. With the default configuration (s1.0_c35), QoE ranges from 3.41 to 3.76. Under low IM, QoE reaches 3.48 under high delay and 3.76 under low delay, while under high IM it remains at 3.41 and 3.52. These results are comparable to those obtained with LLR and are substantially higher than E-policy in the high-IM conditions.

Unlike LLR, JitBright shows somewhat more benefit from condition-specific parameter selection. With the optimal settings, QoE ranges from 3.45 to 3.81. The selected smoothing parameter ranges from 0.7 to 1.0, while the selected capacity ranges from 55 to 65 Mb/s. The highest QoE, 3.81, is obtained under low delay and low IM with the s1_c65 configuration. Under high IM, JitBright achieves QoE values of 3.45 and 3.51, demonstrating that it can mitigate interruptions without introducing enough additional delay to substantially reduce QoE. Overall, the results indicate that JitBright adapts effectively to different delay-IM conditions and provides QoE comparable to LLR.

5) *Summary*: Overall, the results highlight the trade-off between reducing interruptions and introducing additional playback delay. The I-policy provides the highest QoE when network conditions are favorable, since buffering provides little benefit when IM is already low. E-policy shows the largest QoE degradation under challenging conditions because its added delay outweighs the benefits of interruption reduction. QM provides better QoE but is more sensitive to its buffering threshold.

LLR and JitBright provide the most consistent QoE across the evaluated conditions. LLR performs well even with its default configuration, while JitBright benefits somewhat more from condition-specific parameter tuning. These results suggest that cloud gaming buffering should be adaptive, remaining minimal under stable conditions and increasing only when interruption reduction can compensate for the additional delay.

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 our study does not differentiate QoE sensitivity across different game genres. Previous research suggests that game genres exhibit varying degrees of sensitivity to delay and jitter [35]–[37]. For example, fast-paced first-person shooters (FPS) and fighting games tend to be sensitive to delay and jitter, whereas turn-based strategy games or card games may be more resilient to both. Future research could explore genre-based QoE models, integrating game-specific buffering strategies that adjust buffer sizes dynamically based on the gameplay demands.

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 game streaming imposes stringent requirements on both latency and visual smoothness, making playout-buffer design a critical component of end-to-end QoE. While buffering techniques are well established in video streaming, their effects in cloud gaming—where responsiveness is equally important—remain less understood. This chapter evaluated the I-policy baseline and four buffering strategies: E-policy, QM, LLR, and JitBright, using the Cushion simulation framework. By examining how these policies influence interruption magnitude, playback delay, and ultimately QoE across a range of jitter and delay conditions, we gain deeper insight into how playout buffers shape interactive streaming performance.

Overall, our results show that playout buffer policies do influence QoE in cloud gaming, but the magnitude of improvement is generally modest. Across most network conditions, the QoE gains from buffering over the I-policy baseline are small—typically only 0.05–0.15 MOS, or roughly a 2–6% relative increase. The one notable exception occurs in the high-jitter, high-delay scenario, where buffering provides the largest benefit by mitigating the most severe frame-timing fluctuations; even there, however, the improvement remains limited to a few tenths of a MOS point.

A key reason for this limited QoE improvement is the nature of cloud gaming itself: interactions are frequent and time-critical, making users highly sensitive to added delay. Any buffering strategy that smooths frame timing must do so by adding latency, and even modest increases in delay can diminish responsiveness enough to offset the smoothness gains. This inherent tension sharply constrains how much a playout buffer can help in practice.

E-policy and QM reduce interruptions but introduce considerable latency, yielding minor or even negative QoE benefits except under extreme jitter. Aggressively tuned LLR attains slightly higher peak QoE, and JitBright delivers the most stable performance, yet both remain close to the simple baseline. Overall, buffering offers only limited visual smoothing and cannot markedly improve QoE given cloud gaming’s strict latency requirements, implying that platforms are better served by mitigating network jitter and delay rather than heavily depending on client-side buffering to hide timing fluctuations.

FUNDING

No funding was received for conducting this study.

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 perceptual 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. Roskusi, 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)*, Ottawa, 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_\(video_game\)](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] 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>
- [30] T. C. Authors, "low_latency_video_renderer_algorithm," Jan. 2020, online: <https://tinyurl.com/mpe5sup>.
- [31] 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.
- [32] X. Xu and M. Claypool, "Measurement of Cloud-Based Game Streaming System Response to Competing TCP Cubic or TCP BBR Flows," in *Proceedings of the 22nd ACM Internet Measurement Conference*, Nice, France, 2022, p. 305–316.
- [33] —, "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>
- [34] 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>
- [35] S. S. Sabet, S. Schmidt, S. Zadtootaghaj, C. Griwodz, and S. Möller, "Delay sensitivity classification of cloud gaming content," in *Proceedings of the 12th ACM International Workshop on Immersive Mixed and Virtual Environment Systems*, ser. MMVE '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 25–30. [Online]. Available: <https://doi.org/10.1145/3386293.3397116>
- [36] S. Schmidt, S. Zadtootaghaj, and S. Möller, "Towards the delay sensitivity of games: There is more than genres," in *2017 Ninth International Conference on Quality of Multimedia Experience (QoMEX)*, 2017, pp. 1–6.
- [37] M. Claypool and K. Claypool, "Latency and player actions in online games," *Commun. ACM*, vol. 49, no. 11, p. 40–45, Nov. 2006. [Online]. Available: <https://doi.org/10.1145/1167838.1167860>