Event Sequence-driven Generalized and Accurate End-to-end Streaming Latency Measurement

Boqing Shi and Li Zhang

Beijing University of Posts and Telecommunications, Beijing, China

Abstract. Video streaming latency is critical for service providers for network problem diagnosing and further data analytics to provide better user experience to end users. However, there lacks a generalized approach for measuring end-to-end latency and corresponding quality of service metrics in cloud-based video streaming systems. Therefore, we proposed a new approach to measure the end-to-end latency, along with a list of valuable metrics for streaming service providers, such as video encoding/decoding latency and jitter buffer latency. Our system implements such an approach by instrumenting the video streaming infrastructure at both the server side and client side by labeling event identifiers and corresponding timestamps. Compared with traditional approaches that require manual efforts to train a deep learning model or demands ad-hoc hardware, our system significantly reduces the overhead for involving humans in the system pipeline, and also provides comprehensive streaming-related metrics. The evaluation results show the different kinds of metrics contributed to the end-to-end latency, which we think helps the community further improve their streaming systems.

1 Introduction

Streaming has assumed a pivotal role in the contemporary information age, facilitating the transmission of audio and video content via the Internet. This technology enables real-time global information dissemination and entertainment consumption, revolutionizing the means by which we access information and amusement. It has evolved into an omnipresent digital media experience, encompassing various forms such as cloud gaming, live broadcasts, and short video platforms like TikTok. In the realm of streaming media, the latency indicator holds immense importance, as it directly impacts the user experience. Lower latency equates to swifter loading speeds and real-time delivery, thereby providing a seamless viewing experience. Consequently, latency testing stands as an indispensable task for every streaming system.

In traditional non-interactive video streaming scenarios, the primary concern revolves around the rendering delay of the video. This delay refers to the time it takes for the video to render from the push stream source to the pull stream destination. However, as shown in Figure 1 interactive scenarios encompass more

EAI MLICOM 2023, December 17, Beijing, People's Republic of China

DOI 10.4108/eai.17-12-2023.2342317

user interactions compared to straightforward video streaming. Consequently, when conducting delay testing, it becomes imperative to account for the additional delay introduced by these user interactions. Regrettably, as of now, there is no standardized delay testing scheme specifically tailored to interactive scenarios. Much of the existing research has opted to focus on controlling the additional latency introduced by network factors [5, 7, 9–11]. This approach, while common, comes with the drawback that it does not provide an exact measurement of the end-to-end latency. The mainstream solution in the industry adopts high-speed camera shooting. This approach simultaneously captures the hand gesture and the display, allows us to determine the moment when the user initiates the click and receive the corresponding response frame, subsequently calculating the time difference as the end-to-end latency. However, it's worth noting that this method necessitates additional hardware and human intervention. Moreover, it yields only a single set of time differences per test, making it operationally challenging and data acquisition laborious. Recent advancements in this field involve latency prediction through model training. In this setup, an AI robot autonomously triggers interactions and identifies response frames on the receiving screen, subsequently quantifying the latency. Although this method offers automation, it is not entirely precise. Additionally, the model requires retraining for each unique scenario, rendering it challenging to reuse.

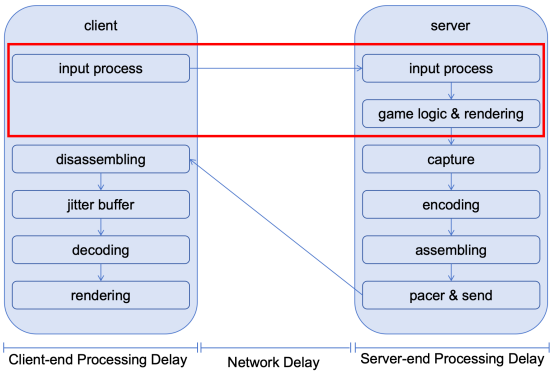

Fig. 1. Workflow of interactive video streaming scenarios

To address the challenge of lacking an end-to-end latency measurement scheme in interactive scenarios, this system introduces a system-level approach. This method allows for straightforward latency testing of the interactive streaming media system by implementing a few key transformations. When a user engages in an interaction, the system records both the interaction event's unique identifier and the initiation time. The interaction events are then transmitted to the server. Simultaneously, the server processes the interaction event, displaying its corresponding identifier on the screen, and subsequently sends it back to the user

along with the associated image. The client, in turn, identifies the first frame featuring the event identifier as the response frame. By measuring the time difference between the rendering of this frame and the initiation time of the event, the system derives the end-to-end latency.

This method eliminates the need for additional hardware, making it a more convenient and reusable solution. Moreover, it defines a set of indicators based on the end-to-end latency in interactive scenarios, enabling a more precise evaluation of interaction quality. When applied in a real interactive streaming scenario, this system efficiently collects and processes data in real-time. It produces comprehensive statistical results, encompassing traditional indicators, newly defined metrics, and supplementary segmentation delay information, facilitating thorough delay detection for various streaming applications.

The primary contributions of this work are as follows: 1) The proposal of a system-level delay measurement scheme for interactive video streaming scenarios. 2) The definition of a comprehensive set of evaluation indicators based on end-to-end latency, allowing for a thorough assessment of the streaming system's interaction quality. 3) The validation of the significance of end-to-end latency through experimental data, coupled with a concise analysis of the influence of network and server resource factors on end-to-end latency.

2 Background and Related Work

2.1 Edge Facilitated Streaming Services

The evolution of technologies like 5G and edge computing has led to the deployment of edge data centers and the migration of cloud services closer to users. This transition aims to offer customers services with reduced latency and enhanced user experiences. Concurrently, the advancement of neural networks and game engines has substantially increased the processing demands on mobile devices, elevating the quality of applications. In order to experience the latest applications, users have to buy more expensive devices, or have to endure problems such as heating of the devices and freezes of pictures. Nonetheless, this trend also opens up opportunities for the development of cloud gaming and cloud-based mobile services. By leveraging Cloud as a Service, users can offload resource-intensive mobile applications onto virtual server-based devices. This approach handles tasks such as video rendering and neural network inference, while the mobile device functions solely as an endpoint for receiving computed results from the server. Consequently, this enhances the overall user experience.

However, even with these advancements, existing video streaming systems still encounter issues like freezes, jitters, and image quality degradation in response to user interactions. These issues cannot be solely quantified and analyzed through user sensory feedback. This gives rise to several challenges. First, it becomes difficult to ascertain whether the services offered by the provider meet the required Quality of Service (QoS) standards. Second, if a high-level scheduling system need to factor in streaming QoS for task scheduling or dynamic configuration adjustments, it is currently infeasible to make real-time enhancements to

streaming quality. Finally, users lack an intuitive and effective means of identifying instances of freezing and jitter in the services they use, and they lack precise data to report these issues to the service provider.

2.2 End-to-end latency vs. Render Latency

In our system, we define end-to-end latency as the time elapsed from the moment a user initiates an operational command until the first video frame containing the response is received. On the other hand, render latency is defined as the time it takes for the server to render a frame and the client to subsequently receive this frame. Traditional measurement schemes used in non-interactive video streaming scenarios, such as WebRTC [3], predominantly rely on rendering delay. However, when these schemes are applied in interactive contexts, the resulting measurements tend to underestimate the actual delay experienced by users. This is because they do not account for the time required for upstream network transmission and the processing latency associated with operational events on the server side. To better reflect the true Quality of Service (QoS) and provide more accurate measurements, it is advisable to employ end-to-end latency, which aligns more closely with real-world conditions.

2.3 Measuring the Latency of Streaming Systems

Table 1. Comparison of measurement delays, measurement objects, and measurement methods for different jobs. 'd' in the header represents delay, 'c' represents the client, 's' represents the server, "full" represents the overall delay, and "part" represents segmentation delay.

reference	d_full	d_s		d_c		d_net	platform	delay measurement method
		full	part	full	part			
Sack'16 [11]	×	✓	×	×	×	✓	Self-built	No end-to-end latency measurement, control network-induced delay
Beye'15 [4]	✓	×	×	×	×	✓	Commercial	High-speed (240fps) camera recording
Jars'13 [10]	×	✓	×	×	×	×	Self-built	No end-to-end latency measurement, control network-induced delay
Clin'13 [5]	×	✓	×	×	×	×	Self-built	No end-to-end latency measurement, control network-induced delay
Iqba'21 [8]	✓	✓	×	✓	×	✓	Commercial	Training the model to determine response frames
Graf'21 [7]	×	✓	×	×	×	×	Commercial	No end-to-end latency measurement, control network-induced delay
Lind'20 [6]	×	✓	×	×	×	×	Commercial	Use cloud game platform external interface, frame tracing
Jars'11 [9]	×	✓	×	×	×	×	Self-built	No end-to-end latency measurement, control network-induced delay
Ours	✓	✓	✓	✓	✓	✓	Self-built	System-level general solution to accurately measure end-to-end latency

As illustrated in Table 1, recent years have witnessed a plethora of studies dedicated to examining the Quality of Service (QoS) in cloud gaming. A common focal point in most of these studies has been the additional latency introduced by cloud-based systems. This demonstrates the critical significance of latency as a metric profoundly influencing the Quality of Service in cloud gaming. A significant portion of research in this domain refrains from measuring the end-to-end latency of cloud gaming platforms, primarily due to the absence of a robust testing methodology. Instead, these studies often substitute the measurement of end-to-end latency with the introduction of additional network delays, thereby circumventing the challenge [5, 7, 9–11]. Among those that do endeavor to measure the end-to-end latency of cloud gaming platforms, the methodologies employed differ based on the platforms' degrees of freedom and the ease of access to testing tools. These methodologies can broadly be categorized into the following groups.

Screen recording. In the past, a conventional approach for measuring the end-to-end latency of streaming systems involved using high-frequency cameras to capture both the sender's screen and the receiver's screen [12]. This entailed simultaneously displaying the end-to-end video stream on both ends of the screens, showing the system's time of sending at the source, and employing high-speed cameras at both ends to capture images. The goal was to observe the time displayed at the receiving end and calculate the real-time difference from the sending end to determine the screen delay. For instance, in the work by [4], a high-speed camera operating at 240 frames per second (fps) was utilized to identify response frames and subsequently compute the end-to-end latency. However, this method was labor-intensive and time-consuming, mainly due to the complexities involved in software engineering.

Using ad-hoc hardware. An alternative approach for measuring end-to-end latency is to utilize specialized hardware, such as the NVIDIA Reflex monitor [2], equipped with a built-in latency analyzer. For instance, the NVIDIA Reflex monitor features a distinctive port that connects to your input device and can identify graphical changes in specific areas on the screen. It determines end-to-end latency by measuring the time elapsed between an input event and a change on the screen. This method offers enhanced accuracy and reliability compared to using high-speed cameras. However, it comes with some limitations. It necessitates a compatible monitor and input device and permits users to collect only one sample at a time. Users must manually record the readings, and there is no automated mechanism for transferring all the collected data to storage.

Deep learning-based response detection. To measure the overall manipulation delay, including factors beyond image rendering delay, DECAF [8] employed an AI bot trained for each specific game used in the tests. This bot played the game autonomously and identified the response frame corresponding to its actions. Figure 2(a) illustrates the latency measurement workflow in DECAF. The bot initiates an operation at time C1 and uploads it to the server. The server then completes game rendering and transmits the game image back to the client. Upon receiving the image, the client timestamps each accepted video

frame. The bot subsequently identifies the image associated with its operation in the transmitted video stream, records the timestamp at time C2, and calculates the end-to-end latency as C2-C1. However, deep learning model-based methods such as DECAF encounter several limitations: 1) To train a deep learning model, DECAF necessitates manual construction of the training dataset, which demands significant human effort. Additionally, the model's prediction accuracy may introduce errors. 2) For different games, DECAF requires retraining a bot to adapt to each new game, which limits its adaptability. 3) DECAF refines delay measurement in stages, addressing components like coding delay and jitter delay. However, real cloud gaming platforms involve various potential error-introducing links, and more dimensions could be explored in the discussion of delay.

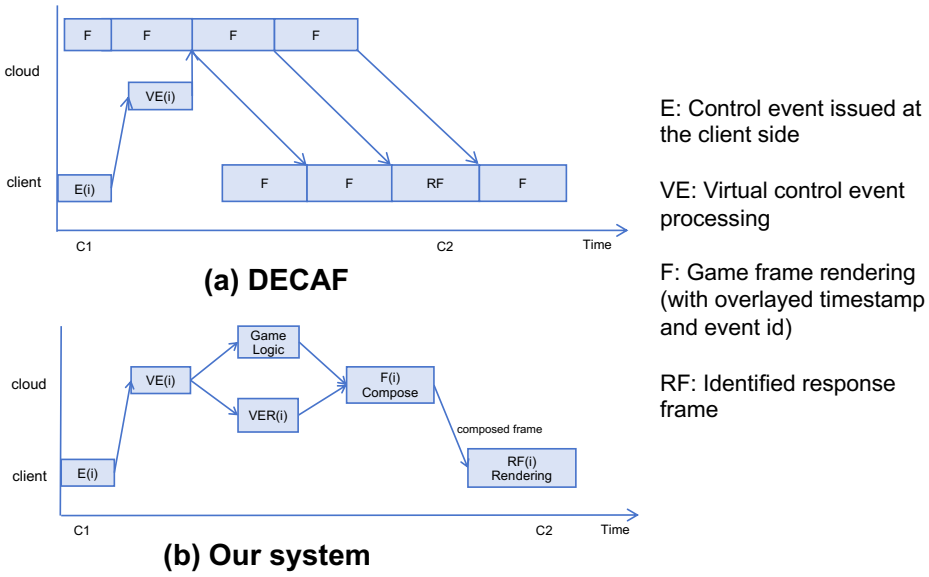

Fig. 2. Latency measurement workflow of DECAF [8] and our system.

User subjective evaluation. Numerous studies also place significant emphasis on evaluating the player's subjective experience. Players often provide ratings of their experiences as "good," "acceptable," or "unacceptable" concerning delays [6, 10]. However, in these works, subjective evaluation typically serves as the experimental dependent variable used to examine its correlation with objective delay.

The streaming measurement system proposed in our work primarily focuses on end-to-end latency. Simultaneously, it incorporates the collection of data on dimensions crucial to the Cloud gaming link, such as packet loss rate and bit rate. Furthermore, it delves into the sub-categorization of control delay, thereby facilitating the collection of segmented delay data across various dimensions.

Unlike DECAF, which solely offers black-box testing for commercial cloud gaming systems, our system boasts the advantage of easy integration into any cloud gaming streaming system. This flexibility makes it highly convenient for other researchers to self-assess their streaming systems, ultimately resulting in more precise measurement outcomes.

3 Design

3.1 Summary

In this section, we present the design of our streaming latency measurement system. This system is constructed upon the message transmission framework that operates between the client (e.g., mobile devices) and the cloud server. It also establishes a set of event identification protocols, ensuring that all operation events dispatched to the cloud server are meticulously analyzed and processed. These data are subsequently consolidated for offline analysis, ultimately yielding statistical results for metric collection. Compared to prior approaches such as [2, 4, 12], our method offers a more generalized solution. It eliminates the need for additional hardware, such as cameras for real-time latency capture, and does not necessitate any alterations to either the client-side or server-side components. As a novel messaging protocol, it seamlessly integrates into any existing end-to-end video streaming system.

Furthermore, building upon the end-to-end latency, our system establishes a comprehensive set of metrics within video streaming systems. These metrics encompass, but are not limited to, end-to-end latency jitter, operation freeze rate, and more. Such metrics serve as valuable references for both fundamental and intricate data analysis, thereby enhancing existing video streaming infrastructures. In the following, we will begin by elucidating the latency measurement method and subsequently provide an overview of the metrics incorporated in our system.

3.2 Latency Measurement

System Workflow. Different from DECAF, our system employs a system-level approach to correlate user operation events with response frames. This approach not only reduces the need for human labeling and the training of deep learning models for various game and video genres but also enhances the system's robustness and generality.

The specific workflow of this system is shown in Figure 2(b). Upon a user's initiation of an operation, the client-side system assigns a self-increasing event id to the operation event. Subsequently, it transmits the event, along with the event id, to the server while simultaneously storing the locally generated timestamp. For instance, in the case of a click operation, which comprises four distinct events (one action-down, two action-moves, and one action-up), the respective event ids are denoted as 141 to 144. The client records the transmission times

Fig. 3. Response frame of Honkai: Star Rail [1] with frame id and event id.

of all four events. Upon receiving the event with an event id of 144, the server displays the corresponding id on the screen and forwards the event for game processing. Additionally, the server appends the current timestamp to the screen. For instance, if the millisecond timestamp reads 1684414045941 when the event id is 144, this timestamp is utilized as the identifier for the frame. As depicted in Figure 3, since this is the event id 144 on the initial screen, it is recognized as the response frame for the event with id 144. The rendered image at this moment encompasses three components: 1) the game screen; 2) the timestamp, equivalent to frame id 1684414045941; 3) event ids from a certain time period, with the most recent id being 144. Subsequent to the server's transmission of the composite frame to the client-side, the client-side executes the final rendering and preserves the rendered frame along with the corresponding rendering timestamp, enabling further processing. In the case of operation events with id 144, it is simply a matter of iterating through the saved frames and identifying the first frame where the event id "144" appears, thus concluding that this frame corresponds to the response for operation event 144. This event id identification process can be easily automated through the use of well-established OCR (Optical Character Recognition) technology, which is currently known for its high accuracy. It significantly surpasses the precision achieved by game robots trained using DECAF. Once the control event and its corresponding response frame are identified, the end-to-end control delay *C2-C1* can be calculated by subtracting the control event time *C1* from the display time *C2* of the response frame.

Given that both the generation of the control event and the reception of the user's response screen occur on the client side, where the stream is pulled, there is no need to contend with NTP timestamp discrepancies. In a 60-frame scene, where the expected frame interval is 16ms, utilizing the millisecond NTP timestamp as the frame id ensures the uniqueness of each frame's identifier. Furthermore, as the event id employs an auto-increment ID, it is assured that each subsequently sent event ID differs from the previous one.

3.3 Metrics Definition

In prior research on cloud gaming Quality of Service (QoS), the majority of metrics were video stream-centric, including parameters like latency, frame rate, and picture quality. Additionally, there were metrics rooted in data streams, encompassing factors like bitrate and packet loss rate. Building upon operation events, our system introduces a comprehensive set of metrics for assessing the operational experience within current cloud gaming systems. These metrics serve as a complement to quality evaluations performed at the video stream and data stream levels. The specific metrics are detailed in Table 2.

For every pair consisting of an operation event and its corresponding response frame, our system can furnish the end-to-end latency. This latency can be further divided into sub-components, including the latency pertaining to the response frame at the server, network, and client. This subdivision aids in identifying the source of elevated operation latency. Additionally, the system generates statistical metrics for all operation events throughout the entire stream. Notably, while previous research on cloud gaming may have afforded limited attention to jitter, our system recognizes the significance of both video frame rate jitter and end-to-end latency jitter as crucial metrics for assessing response service stability. As a result, we have incorporated these metrics into the QoS evaluation criteria.

All the aforementioned metrics assist developers and streaming platform operators in gaining a more comprehensive understanding and facilitating the swift and straightforward diagnosis of issues.

4 Evaluation

To validate this system, we established a cloud gaming platform utilizing the open-source real-time streaming technology, WebRTC [3], and integrated the measurement system to analyze the actual gameplay process. As depicted in Figure 4, the server component of the cloud gaming system is powered by the Snapdragon 865 SoC chip, running the Android host system while initiating the docker service. Subsequently, the Android system's image is launched within the docker environment, and the game "Honkai: Star Rail" and the streaming component are executed in this container system. On the client side, players utilize a Huawei P40 Pro smartphone, which is equipped with the cloud gaming client for operation, streaming, and display.

In the first experiment, we opted for the deployment of servers in two provincial capital cities: one located 276 kilometers from the player, and the other positioned at a distance of 1741 kilometers. This choice was made to highlight the measurement system's sensitivity to network delays. In the second experiment, we selected servers at two different resource levels: one with a single container system running on a physical SoC, and the other with three container systems. This choice was made to assess the system's sensitivity to server-level system resources.

We conducted six sets of experiments in each scenario, with each experiment spanning 10 minutes. The reported values represent the averages. In addition

Table 2. Metrics defined in our system.

Type	Metrics	Description
Delay	end-to-end latency	Delay from user physical operation to response frame displayed on the client
	Picture transport delay	Delay from response frame captured on the server to response frame displayed on the client
	Network round trip time	Delay of response frame Transmittion in network
	Encode delay	Delay of response frame encoding in the server
	Pacer and send delay	Delay of response frame waiting in the pacer queue in the server
	Jitter buffer delay	Delay of response frame waiting in the jitter buffer in the client
	Decode delay	Delay of response frame decoding in the client
	Render delay	Delay of response frame renderinging in the client
Operation	Operation freeze rate	In Chromium, a freeze is counted when the inter-frame delay satisfies: *interFrameDelay >= max (3 * avgInterFrameDelay, avgInterFrameDelay + 150(ms))*. Similarly, we can assume that operation freeze is occured when the end-to-end latency satisfies: *E2ELatency >= max(3 * avgE2ELatency, avgE2ELatency + 220(ms))*
	Operation lost	The number of operation events lost in the system
	Standard deviation of end-to-end latency	The fluctuation of end-to-end latency, which indicates system stability

to end-to-end latency, we also provide data on custom operation freeze rates, operation loss rates, standard deviations of end-to-end latency, and various segmentation delays in Table 3.

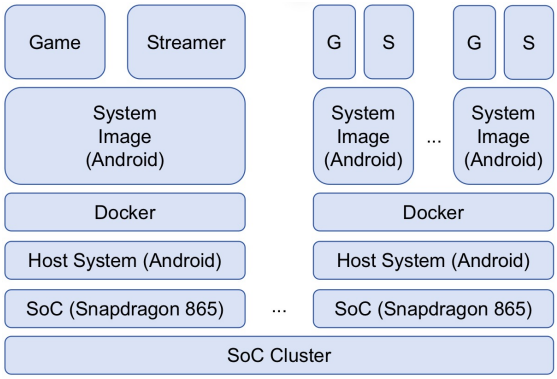

Fig. 4. Server end experiment architecture

Table 3. Result of experiments. 'd' in the header represents delay, 'r' represents rendering, 'e2e' represents end-to-end, 'c' represents client, 's' represents server, and 'n' represents network.

server detail	d_r	d_e2e	d_s	d_c	d_n	freeze	lost	σ(d_e2e)
276km, single container server	101.33	155.21	36.23	59.70	18.05	1.67%	0.4%	17.29
1741km, single container server	123.96	187.88	37.07	71.93	40.23	3.16%	1.3%	17.61
276km, three container server	102.87	155.21	38.96	58.44	18.17	1.90%	0.5%	18.33

Upon analyzing the results from the three sets of experiments, it becomes evident that the end-to-end latency is approximately 50% greater than the rendering delay. This additional delay primarily arises from client-side input processing time, message up-stream, and server-side event processing. These components also play a crucial role in shaping the user's actual experience, underscoring the importance of end-to-end latency measurement. It also indicates that the assessment of this aspect cannot be adequately addressed by relying solely on traditional rendering delay metrics.

The experimental results for the first experiment are presented in the first and second rows of the table. It is evident that both rendering delay and end-to-end

latency increase as the communication distance lengthens. Further examination reveals that the rise in end-to-end latency is primarily attributed to the increases in d_n (network delay) and d_c (client delay). Clearly, as the communication distance extends, the network-induced delay also grows. The escalation in client delay is a consequence of WebRTC's jitter buffer policy. WebRTC relies on UDP transmission and must independently address the issue of out-of-order packet delivery. In situations of week network conditions, it extends the waiting time within the jitter buffer prior to packet decoding. This waiting period allows for the reception of preceding data and reduces the failure rate of decoding. As a result, client-side delay increases, aligning with the experimental findings.

The experimental results for experiment B are presented in the first and third rows of the table. It is evident that whether it is a "single-open" or "multi-open" scenario, the impact on end-to-end latency is not significantly substantial. However, in the case of a "triple-open" server, due to a lack of available resources, if all three virtual systems reach peak utilization simultaneously, resource contention may occur. This extreme scenario results in a slight increase in server delay, stuttering rate, and event loss, leading to reduced end-to-end latency stability. This demonstrates that our testing system can effectively pinpoint bottlenecks and evaluate the server's multi-open capacity.

5 Discussion

The paper has certain limitations. Firstly, in the system design, it overlooks the latency inherent in the game itself and focuses more on the additional delay introduced by the interactive streaming system. This approach benefits streaming system developers by allowing them to concentrate on their systems, irrespective of specific user programs. However, the limitation lies in the fact that the actual delay experienced by users might slightly exceed the system's measured value. The specific additional delay represents the inherent latency within the user program itself.

Secondly, the system primarily focuses on measuring of end-to-end latency and offers some degree of analysis on the causes of such freezes and jitters. Yet, it does not propose solutions to address these QoS issues. Experienced developers could rectify problems directly based on the system's issue identification. However, non-professional engineers might require additional learning resources to tackle such issues effectively.

These limitations indicate potential directions for further research:

Further investigation into game-specific latency: Consider incorporating the inherent latency within the game itself into the study. Delve deeper into its impact on user experience and propose methods to comprehensively consider both game-specific latency and the delay introduced by the interactive streaming system.

Providing solutions: Explore potential solutions to address freezes and jitters and propose suggestions or methods in the paper to assist less experienced developers in resolving such problems.

6 Conclusion

The system solves the following three technical challenges at the same time: 1) through the design of the software, an automated method is used to test end-to-end latency, avoiding the use of traditional cameras to take pictures that consume a lot of manpower, and can easily obtain a large amount of data; 2) the end-to-end latency counts the time it takes for the user to initiate the manipulation instruction to the first video frame that receives the response, and uses the manipulation event as the granularity rather than the picture frame for statistics, which solves the problem that the traditional measurement method can only measure the picture transmission delay does not include the deviation caused by the transmission delay of the manipulation event; 3) an interactive quality evaluation system is defined by this system, which gives meaning to the original data source and directly outputs valid conclusions to users and developers.

References

1. Honkai:starrail. https://hsr.hoyoverse.com/en-us/, 2023.
2. Nvidia reflex. https://www.nvidia.com/en-us/geforce/technologies/reflex/, 2023.
3. Webrtc. https://webrtc.org/, 2023.
4. Justus Beyer, Richard Varbelow, Jan-Niklas Antons, and Sebastian Möller. Using electroencephalography and subjective self-assessment to measure the influence of quality variations in cloud gaming. In *2015 Seventh International Workshop on Quality of Multimedia Experience (QoMEX)*, pages 1–6, 2015.
5. Victor Clincy and Brandon Wilgor. Subjective evaluation of latency and packet loss in a cloud-based game. In *2013 10th International Conference on Information Technology: New Generations*, pages 473–476, 2013.
6. Sebastian Flinck Lindström, Markus Wetterberg, and Niklas Carlsson. Cloud gaming: A qoe study of fast-paced single-player and multiplayer gaming. In *2020 IEEE/ACM 13th International Conference on Utility and Cloud Computing (UCC)*, pages 34–45, 2020.
7. Philippe Graff, Xavier Marchal, Thibault Cholez, Stéphane Tuffin, Bertrand Mathieu, and Olivier Festor. An analysis of cloud gaming platforms behavior under different network constraints. In *2021 17th International Conference on Network and Service Management (CNSM)*, pages 551–557, 2021.
8. Hassan Iqbal, Ayesha Khalid, and Muhammad Shahzad. Dissecting cloud gaming performance with decaf. *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, 5(3):1–27, 2021.
9. Michael Jarschel, Daniel Schlosser, Sven Scheuring, and Tobias Hoßfeld. An evaluation of qoe in cloud gaming based on subjective tests. In *2011 Fifth International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing*, pages 330–335, 2011.
10. Michael Jarschel, Daniel Schlosser, Sven Scheuring, and Tobias Hoßfeld. Gaming in the clouds: Qoe and the users' perspective. *Mathematical and Computer Modelling*, 57(11):2883–2894, 2013. Information System Security and Performance Modeling and Simulation for Future Mobile Networks.

11. Andreas Sackl, Raimund Schatz, Tobias Hossfeld, Florian Metzger, David Lister, and Ralf Irmer. Qoe management made uneasy: The case of cloud gaming. In *2016 IEEE International Conference on Communications Workshops (ICC)*, pages 492–497, 2016.
12. Mengwei Xu, Zhe Fu, Xiao Ma, Li Zhang, Yanan Li, Feng Qian, Shangguang Wang, Ke Li, Jingyu Yang, and Xuanzhe Liu. From cloud to edge: a first look at public edge platforms. In *Proceedings of the 21st ACM Internet Measurement Conference*, pages 37–53, 2021.