Tooth: Toward Optimal Balance of Video QoE and Redundancy Cost by Fine-Grained FEC in Cloud Gaming Streaming

Congkai An$^+$, Huanhuan Zhang$^+$, Shibo Wang$^+$, Jingyang Kang$^+$, Anfu Zhou$^+$, Liang Liu$^+$, Huadong Ma$^+$, Zili Meng$^\diamond$, Delei Ma$^\circ$, Yusheng Dong$^\circ$, Xiaogang Lei$^\circ$

$^+$*Beijing University of Posts and Telecommunications*,
$^\diamond$*Hong Kong University of Science and Technology*, $^\circ$*Well-Link Times Inc.*

Abstract

Despite the rapid rise of cloud gaming, real-world evaluations of its quality of experience (QoE) remain scarce. To fill this gap, we conduct a large-scale measurement campaign, analyzing over 60,000 sessions on an operational cloud gaming platform. We find that current cloud gaming streaming suffers from substantial bandwidth wastage and severe interaction stalls simultaneously. In-depth investigation reveals the underlying reason, *i.e.*, existing streaming adopts coarse-grained Forward Error Correction (FEC) encoding, without considering the adverse impact of frame length variation, which results in over-protection of large frames (*i.e.*, bandwidth waste) and under-protection of smaller ones (*i.e.*, interaction stalls). To remedy the problem, we propose Tooth, a per-frame adaptive FEC that aims to achieve the optimal balance between satisfactory QoE and efficient bandwidth usage. To build Tooth, we design a dual-module FEC encoding strategy, which takes full consideration of both frame length variation and network dynamics, and hence determines an appropriate FEC redundancy rate for each frame. Moreover, we also circumvent the formidable per-frame FEC computational overhead by designing a lightweight Tooth, so as to meet the rigid latency bound of real-time cloud gaming. We implement, deploy, and evaluate Tooth in the operational cloud gaming system. Extensive field tests demonstrate that Tooth significantly outperforms existing state-of-the-art FEC methods, reducing stall rates by 40.2% to 85.2%, enhancing video bitrates by 11.4% to 29.2%, and lowering bandwidth costs by 54.9% to 75.0%.

1 Introduction

Cloud gaming, as an increasingly embraced application, involves high-quality video streaming from remote servers to diverse devices, thereby obviating the need for advanced local hardware and enabling pervasive gameplay across all connected devices, particularly mobile devices with limited computation and storage capacity. In recent years, several gaming platforms have launched acclaimed cloud-based versions of games, such as Cloud Genshin Impact [12] and Tower of Fantasy [13]. Market analyses reveal that the global cloud gaming market, which stood at US\$ 1,905.6 million in 2023, is projected to reach US\$ 48,552.2 million by 2032, demonstrating a compound annual growth rate of 42% [5].

To ensure a satisfactory interaction QoE, cloud gaming requires consistently ultra-low latency from player request to game response, typically less than 100 milliseconds [29]. To better understand the real-world QoE of cloud gaming, we conduct a comprehensive field measurement involving 66,128 players across 22 cities worldwide, as detailed in Table 1. Our measurements reveal that the downlink delivery of game video content (*i.e.*, from the remote server to the end devices) is the primary bottleneck impacting interactive QoE. To mitigate this, besides usual retransmission mechanisms [21,36], cloud gaming commonly adopts Forward Error Correction (FEC) from the current video streaming works [18,20,31,33,42] to recover lost data packets. However, our analysis indicates that existing FEC algorithms significantly increase bandwidth usage, adding an additional 0.26× to 1.0× redundancy than the original data streaming. We further conduct an in-depth exploration of existing FEC schemes and identified their two primary limitations.

- *Frame length affects the redundancy rate for recovering a frame.* Specifically, FEC can recover the lost packets within a frame when the redundancy rate can cover the Loss Rate In Frame (LRIF). However, we observe that the video frame length (number of packets) in cloud gaming varies largely, *i.e.*, spanning from 2 packets to over 80 packets, and frame LRIF is a random variable that follows a binomial-like distribution with a function of the frame length. Especially, the LRIF of a small frame is as high as 30+% and that of a large frame is as low as 2%, differing by over 10× and requiring completely different redundancy rates (§2.3).
- *Existing FEC algorithms are coarse-grained, ignoring frame LRIF differences but applying a uniform redundancy rate across almost all video frames.* As a result, they tend to over-protect large frames by adding significantly more redundancy than necessary, with median values ranging from 5.2× to 9.2×. Worse still, they under-protect small frames, failing to recover their lost packets and resulting in frequent interaction stalls, occurring more than once per minute. Furthermore, our findings indicate that coarse-grained FECs exacerbate stall frequency in operational cloud gaming systems due to excessive redundancy, which leads to increased network congestion bursts. This, in turn, elevates the LRIF for video frames, particularly for smaller frames, making them more difficult to recover.

In this paper, we aim to address a key problem: *How to follow frame LRIF to ensure successful recovery in small frames meanwhile avoiding wasting redundant bandwidth in* *large frames, so as to propel cost-effective cloud gaming in the real world?* To accomplish this, we propose Tooth, a novel fine-grained FEC tailored for cloud gaming at the frame level. It precisely determines the delivery probability of each frame through the frame length distribution, and adaptively applies the per-frame redundancy rates for different frames to align the delivery probability. Although conceptually simple, building Tooth in practice introduces two significant challenges.

Firstly, the realistic LRIF pattern is more complicated than the binomial distribution, and identifying the critical factors that influence the binomial distribution of per-frame LRIF is essential for enabling precise FEC. Our field measurements reveal that both the application-layer frame length and the transport-layer network loss patterns significantly impact LRIF. In particular, the frame length can be readily acquired from the video encoder, while inferring the network loss pattern is a non-trivial task. Traditionally, loss patterns are simply characterized by the loss rate [18,20,27,31,38]. However, our evaluations show that this metric does not accurately depict the distribution of network loss events on a per-video-frame basis. To address this deficiency, we introduce a novel metric–network loss aggregation, which quantifies whether network packet loss events are more dispersed across multiple video frames or concentrated within a few frames, thereby aligning more effectively with fine-grained FEC encoding strategies.

Secondly, effectively mapping LRIF-related factors to per-frame redundancy rate, while ensuring Tooth lightweight enough for practical deployment. In particular, the relationship between per-frame redundancy rate and LRIF-related factors is non-linear. Straightforward mapping methods like linear function or neural network models, cannot guarantee the video QoE and running cost simultaneously (as will be validated in §5.3). Furthermore, the key factors we investigated, such as network loss rate and loss aggregation, are highly volatile due to network dynamics, *i.e.*, only historical network loss pattern cannot reliably reflect future conditions. Therefore, we design a dual-module FEC encoding for Tooth, which consists of *(i)* a *slow-module*, which learns network volatility to estimate the future network loss pattern (loss rate lr and loss aggregation la) based on historical loss patterns and packet-by-packet reception. It utilizes a compressed neural network model to improve prediction accuracy and operates with lower frequency, *i.e.*, once for each network RTCP feedback cycle. *(ii)* a *fast-module*, which formulates the non-linear and discontinuous mapping between LRIF-related factors and redundancy of each frame as a regression problem, and then uses a lightweight random forest method to model the relationship. The *fast-module* runs every time a video frame is encoded, thus ensuring that redundancy is immediately determined and applied to each newly generated video frame. As a result, the above decoupled dual modules avoid implementing frame-level FEC through heavy neural network models and also reduce the execution frequency, which allows Tooth agile enough to operate in real-world scenarios.

We have implemented Tooth in a commercial cloud gaming system, one of the popular cloud gaming platforms (anonymized as Company W), and deployed it in its production environments, involving about 2,300 sessions and lasting 6 weeks in total. Compared to state-of-the-art FEC solutions such as RL-AFEC [18] and RTC-FEC [9, 28], Tooth achieves an ideal balance between optimized video QoE and efficient redundancy cost. For instance, it significantly reduces stall frequency by 40.2%-85.2%, increases perceptual video bitrate by 11.4%-29.2%, and reduces bandwidth costs by 54.9%-75.0%. Moreover, we also evaluate Tooth's effectiveness in different *(i)* network types like WiFi, 4G, and 5G; *(ii)* game types like 2D games and 3D games; *(iii)* Internet Service Providers (ISP); *(iv)* distances of game server to end devices, *i.e.*, cross-city sessions and intra-city sessions. The above extensive experiments (§5.2) validate that Tooth can achieve excellent QoE consistently in real-world cloud gaming scenarios. We also conduct a series of ablation studies to verify the key design modules of Tooth.

To summarize, our main contributions are:

- We conduct a large-scale measurement on an operational cloud gaming system and uncover underlying reasons for the inefficiencies of existing loss recovery solutions (§2).
- We propose Tooth, a lightweight dual-module approach that can map the application-layer frame length and transmission-layer network loss pattern to per-frame redundancy in real-time, thus achieving a fine-grained FEC for enhancing cloud gaming streaming (§3).
- We implement and deploy Tooth in the operational cloud gaming system and conduct real-world evaluation, which outperforms state-of-the-art solutions significantly and consistently across environment variation (§5).

2 Background and Motivation

2.1 Background

2.1.1 Packet Loss Recovery

In conventional real-time applications such as video conferencing, playback delays of 200 to 500 milliseconds are typically tolerable [28, 43, 47]. However, in cloud gaming scenarios, stronger interactivity is critical, with significantly stricter latency constraints, *i.e.*, the application-layer interaction latency must consistently remain below 100 milliseconds to ensure optimal user QoE [22–24]. Here we clarify that the application-layer interaction latency refers to the duration between a user's operation input (*e.g.*, run and hit) in the game client and the moment the user receives the corresponding screen feedback. It contains the system processing delay and data round-trip network delay, the measurement method is given in Appendix A. Given the stringent latency requirements of cloud gaming, it significantly influences the design considerations for packet loss recovery. In particular, there are typically two loss recovery mechanisms, which are usually adopted jointly in cloud gaming:

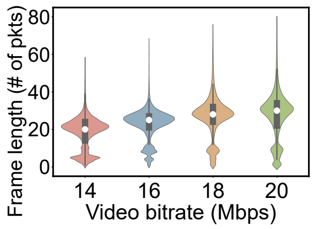


Figure 1: Frame length details at different video bitrates.

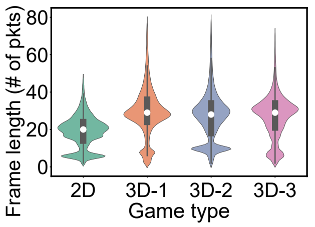


Figure 2: Frame length details in 2D and 3D games.

Feature	Hairpin [36]	Tambur [42]	Xu *et al.* [51]	Ours
User sessions	3,308	9,700	2,430	69,426
Playtime (hours)	600+	-	121.5	31,316
Players	3,308	9,700	1	66,168
Player's City	-	-	1	22
Edge server's City	1	-	1	17
ISP	3	-	-	4
User device models	-	-	1	83
Cloud game products	2	2	3	4

Table 1: Datasets statistics of our large-scale measurements on an operational cloud gaming system, compared with other existing measurement efforts.

- *Forward Error Correction (FEC)* [18, 20, 31, 42], which proactively supplements original data with additional encoded data as redundancy before each transmission. In this case, the lost original data packets will very likely be recovered by the receiver. FEC reduces the delay by not requiring retransmission but increases the bandwidth cost since future packet losses are uncertain.
- *Retransmission* [21, 36]. There are also recent proposals that call for the revitalization of retransmission due to the reduction of RTT. When the packet is lost and detected, the sender will transmit the lost data packets again. This will introduce additional RTT delays to the data delivery, but will only consume the bandwidth of the lost packets.

2.1.2 Variation in Frame Length

A key observation in this paper is the variation of frame length in cloud gaming, for the following two reasons:

- *Rate adaptation.* The available network bandwidth is highly dynamic, sometimes dropping by a factor of 50× in wireless real-time communications [35], so the encoding bitrate will also change accordingly. This is determined by the rate adaptation algorithms such as GCC [17]. Such an adaptive bitrate will result in the variation in frame length (number of packets). We emphasize that the frame length variation here is different from the segment length variation reported in [45], which has a far loose latency requirement.
- *Variable bitrate (VBR).* Meanwhile, due to the internal design of video codecs, even if the bitrate is fixed, the frame length can also vary due to the VBR encoding. For example, under the same video bitrate, the frame length of a static scene will be shorter than that of a dynamic scene.

We conduct a measurement campaign over our operational cloud gaming system (later described in §2.2), and the statistics are shown in Table 1. The video codec we use is NVENC [1, 8] following H.265 [7] and it adopt infinite GOP encoding manner to meet the low-latency transmission characteristics of cloud gaming. Unlike the traditional finite GOP manner sets a fixed GOP length, such as 30 in RL-AFEC [18], infinite GOP disables the server to send I frames periodically, to reduce latency surges, more explanations in Appendix B. The congestion control algorithm is GCC [17]. We present the distribution of frame lengths when the encoding bitrates are fixed to different levels in Fig. 1, and this length difference exists in both 2D and 3D cloud games, as shown in Fig. 2.

2.2 Inefficacy of Existing Solutions

Unfortunately, when deploying FEC and retransmission into production-scale systems, we find that existing solutions suffer from a fundamental trade-off between bandwidth cost and interaction latency.

Measurement Setup. We conduct a large-scale measurement campaign over our operational cloud gaming system. The measurement covers 22 cities across China and 4 ISPs, with other statistics shown in Table 1. It is worth noting that the dataset we collect is the largest dataset in the field of cloud gaming to the best of our knowledge.

FEC algorithms often over-protect data packets with drastically high bandwidth costs. An interesting observation is that many existing FEC solutions often add much more redundant packets than necessary. For example, when there are 5 packet losses out of 25 data packets, ideally only 5 additional redundant packets will protect the data packets. However, existing solutions [18, 36, 42] often add much more than 5 packets. This reason is quite intuitive – more redundancy is added to get prepared for unforeseen packet losses, however, which leads to drastic bandwidth waste. Especially considering that bandwidth cost is the major contributor to the operational expenses for streaming applications such as cloud gaming, this will directly lead to a revenue loss [2, 3, 42].

We validate the observation by deploying two representative FECs in our cloud gaming platform:

- *RTC-FEC* [28], the de-facto rule-based solution in WebRTC [9] based on the current video bitrate and network loss rate.
- *RL-AFEC* [18], a reinforcement learning driven FEC solution. It defines the first I frame and the following K-1 P frame of one GOP (length is N) as the K "critical" frames[1], and the remaining N-K P frames as "non-critical" frames. RL-AFEC finds that non-recoverable critical frames have a more serious negative impact on video quality, so it sets different redundancy rates[2] for the critical frames of each

[1]In *RL-AFEC* [18], there are 6 candidate values for K, *i.e.*, 5, 10, 15, 20, 25, and 30. Here we use the recommended optimal value, *i.e.*, K = 15.

[2]Here we set 10 candidate redundancy rates for RL-AFEC, *i.e.*, 10%, 20%,...100%, completely consistent with [18].

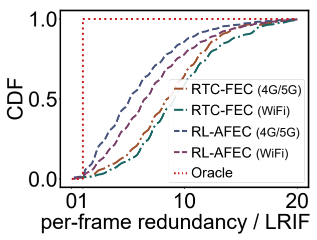


Figure 3: CDF of the ratio of per-frame redundancy rate to LRIF for coarse-grained FEC.

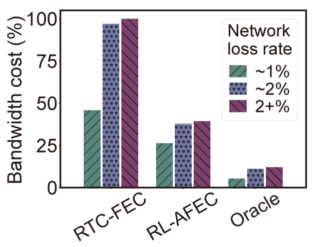


Figure 4: Bandwidth costs of the representative FECs and the ideal Oracle solution.

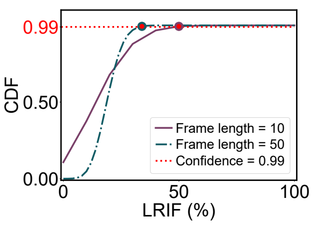


Figure 5: The CDF of video frame LRIF binomial distributions with different numbers of trials.

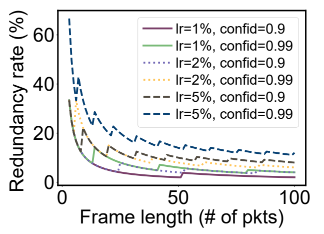


Figure 6: Lines: when the loss rate (lr) is 1%, 2%, and 5%. Y-axis: the redundancy rate. X-axis: the frame length.

GOP. As for the remaining non-critical frames, it still uses the same uniform redundancy rate.

As shown in Fig. 3, both algorithms add much more redundant packets to frames than necessary, with a median number of 5.2×-8.6× in 4G/5G networks and 6.6×-9.2× in WiFi networks, resulting in serious bandwidth waste. In Fig. 4, we show the overall redundancy bandwidth costs of two representative FECs, both are very high, *i.e.*, 45.9%-100% and 26.3%-39.4% respectively. To show the potential optimization space, we compare the existing FECs with an "Oracle" approach, *i.e.*, we count the actual data loss rates of all video frames as the ideal bandwidth cost. As shown in Fig. 4, only 5.4%-12.1% redundancy is required.

Retransmission-oriented solutions still suffer from severe stalls. Including the recent effort, Hairpin [36], there has been an understanding that since the RTT is low enough, merely relying on retransmission might be sufficient to recover the lost packets. However, our experiments show the contrary:

- The network RTT measured in our cloud gaming system is 23 ms for WiFi users and 35 ms for 4G/5G users. This is because as the scale of the product expands, the users with worse network conditions are naturally included.
- The detection of packet loss also takes time in the commonly used negative acknowledgement (NACK) methods. For example, in WebRTC, it will take the receiver up to 20 ms before sending NACK through RTCP protocol.
- Affected by the congestion control and pacing rate of the sender, the transmission duration of video frame data packets often exceeds 10 ms.

Therefore, considering the stringent deadline requirement of around 100 ms [22–24, 36, 37, 48], it is still insufficient to rely on retransmission to recover the lost packets, and the stall frequency even reaches 41.9 times per minute. We will provide the detailed experimental results in §5.2.

2.3 Observation – Frame length matters

In Fig. 3, we have shown that existing FEC algorithms often over-protect many video frames while under-protecting some others (the ratio < 1), leading to both bandwidth waste and interaction stalls. In this subsection, we will show that the root cause of this inefficiency is the neglect of the frame length in the FEC algorithms. For the sake of clarity, we first denote the loss rate in each frame as LRIF, which is defined as $\frac{\#\ of\ lost\ packets\ in\ frame}{\#\ of\ all\ packets\ in\ frame}$. We later have the following findings:

Finding 1: The distribution of video LRIFs vary significantly with the frame length. As we all know, network loss events are random and bursty [33, 36, 42]. Now let's begin with a simple probability question: assume that the loss rate of each packet is 10% and they are independent from each other, if we want to protect the delivery of the frame with the confidence of 99%, how many redundant packets do we need to add when (a) the frame has 10 data packets; and (b) the frame has 50 data packets? In this case, we need to calculate the per-frame LRIF as it is the necessary redundancy rate to be set. The LRIF in this example follows the Binomial Distribution [11], with the probability of each trial being 0.9, and we want the minimum number of trials such as $P(X \geq framelength) \geq 0.99$. The answer is: when the frame length is 10 packets, we need to add 4 redundant packets; when the frame length is 50 packets, we need to add 12 redundant packets. The redundancy rate for smaller frames (40%) is much higher than that for the larger frames (24%)[3].

The reason behind this phenomenon is a basic law in probability theory – the law of large numbers. As shown in Fig. 5, when the number of trials (in our case, frame length) increases, it requires a lower redundancy rate to successfully recover 99% of all frames. Therefore, the redundancy rate needed to protect the frame will decrease as the frame length increases. Here we clarify that a lower redundancy rate does not directly represent adding fewer redundant packets. For a video frame, the ideal redundancy rate is its LRIF and the ideal number of the redundant packets is $framelength \times LRIF$. Next, we further simulate the necessary redundancy rates with different confidence levels and frame lengths, and the results are shown in Fig. 6. This phenomenon is general – the redundancy rate for small frames is much higher than that for large frames.

Finding 2: Existing FEC algorithms ignore the LRIF variation caused by frame length, either over-protect large frames or under-protect small frames. We delve anew into existing FEC algorithms and find the underlying

[3]One can validate the result at https://stattrek.com/online-calculator/binomial.aspx. The parameters we use are (0.9, 14, 10) and (0.9, 62, 50).

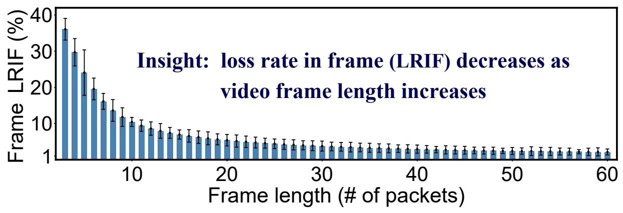
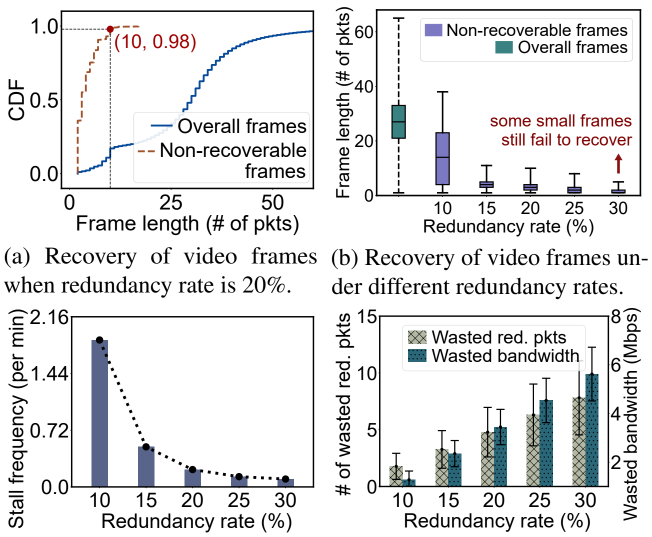


Figure 7: Relationship between video frame length and LRIF. The statistical results come from video frames that suffered packet loss in our commercial cloud gaming platform. Only these frames can reflect the optimal redundancy rate to protect each video frame, more explanations in Appendix D.



(a) Recovery of video frames when redundancy rate is 20%.

(b) Recovery of video frames under different redundancy rates.

(c) Stall frequency under different redundancy rates.

(d) Wasted redundancy (original streams are average 19.3Mbps).

Figure 8: Video frame recovery and wasted bandwidth under coarse-grained redundancy rates. The results are obtained from the controlled analysis to FEC. We replay real-world stream traces in the simulator and don't consider the damage to network condition caused by excessive data transmission.

reason: most of them are coarse-grained, ignoring the impact of frame length, and set a uniform redundancy rate *r* (defined as $\frac{redundant\ data}{original\ data}$) for all frames based on network conditions. While RL-AFEC [18] proposes setting varying redundancy rates for each of the first K critical frames in one GOP, it overlooks the remaining N-K non-critical frames. This is especially unsuitable in the context of cloud gaming, where N-K non-critical frames can comprise up to 99.9% of all frames in one GOP (details in Appendix B). Thus the vast majority of frames will still be set the uniform redundancy rate, even though these frames behave with significant LRIF variations. Therefore, all existing FEC methods tend to set a higher redundancy rate, as we presented above. Unfortunately, even if the redundancy rate is high enough for most frames, it might still be insufficient for small frames. For example, in the line of (lr=10%, conf=99%) in Fig. 5, when setting the redundancy rate to 40% (4× the average loss rate), frames

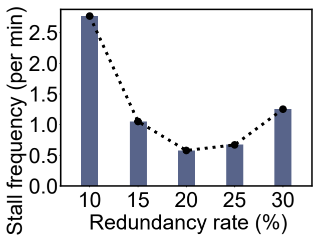


Figure 9: As redundant rate increases, the over-sent redundancy even exacerbates interaction stalls.

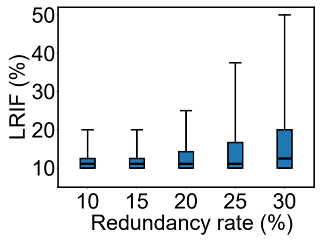


Figure 10: As redundant rate increases, the over-sent redundancy exacerbates small frames' LRIF.

smaller than 10 packets are still likely to fail to recover even though for frames larger than 10 packets the redundancy will be significantly wasted. We further validate this observation by our online measurements as well. As shown in Fig. 7, as the frame length increases, the LRIF mean of the frames that suffered packet loss decreases significantly. This indicates that large frames will easily be over-protected while small frames will be under-protected by the coarse-grained FECs.

The adverse impact of coarse-grained FECs on operational cloud gaming systems. Here we demonstrate how existing coarse-grained FECs impact the performance of cloud gaming in practice, for which we use all traces as listed in Table 1. Firstly, we apply a uniform redundancy rate of 20% to each gaming stream and observe the frame recovery. As shown in Fig. 8a, the non-recoverable frames are relatively small, *i.e.*, 98% of them are small frames (length ≤10). Furthermore, we also apply other 4 redundancy rates to each video stream in turn, *i.e.*, 10%, 15%, 25%, 30%. Fig. 8b plots the recovery results under all five redundancy rates. We can observe that: increasing the redundancy rate can indeed recover more video frames, but some small frames still fail to be recovered, which requires a higher redundancy rate, exceeding 30%. To examine the impact on QoE, we present the interaction stall frequency in Fig. 8c and the wasted redundancy in Fig. 8d. Obviously, the stall frequency decreases along with the increase of redundancy rate, but the marginal effect diminishes very fast, *e.g.*, we barely have extra gain when the redundancy rate rises from 25% to 30% but the wasted bandwidth keeps increasing substantially, from 4.5 Mbps to 5.6 Mbps.

More importantly, in the real world, *our measurement results show the counter-intuitive fact: more redundancy instead exacerbates cloud gaming's interaction stall*. Specifically, we measure the interaction gains of FEC under different redundancy rates of 10%, 15%, 20%, 25%, 30%. As shown in Fig. 9, when the redundancy rate is relatively low, *i.e.*, 10%, 15%, 20%, FEC can continuously reduce the interaction stall frequency. However, as redundancy rate keeps increasing, FEC fails to further optimize and even worsens the stall performance. We investigate the traces and find the root reason: adding too much redundancy will lead to more bursting network congestion, thus the LRIF of a video frame especially a small frame will be even higher, as shown in Fig. 10.

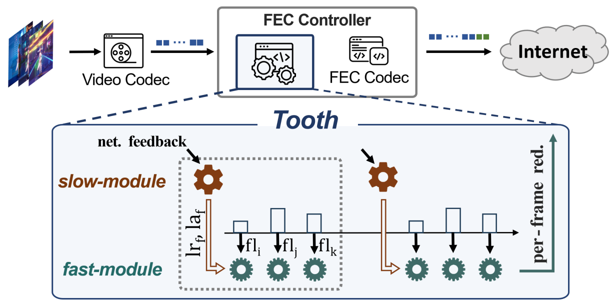


Figure 11: Design overview of Tooth.

To sum up, existing FEC algorithms are commonly coarse-grained and ignore the LRIF differences across video frames. Hence they are trapped in the dilemma of improving frame recovery and avoiding bandwidth waste. Nowadays, cloud gaming applications are becoming more bandwidth-demanding (with higher image quality), it is more imperative to break the dilemma with a novel FEC design.

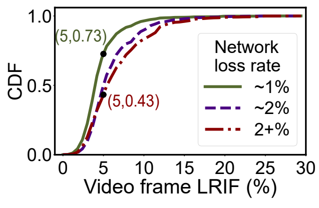
2.4 Design Challenges

Although we have revealed the significant impact of frame length on per-frame redundancy rate, adapting it to achieve fine-grained FEC is non-trivial, which faces 3 key challenges: *(i)* Besides frame length, LRIF also intertwines with the transport-layer network loss pattern. Identifying the critical factors that influence the per-frame redundancy rate is essential for enabling precise FEC at the fine-grained level. *(ii)* The relationship between per-frame redundancy rate and the related factors is non-linear and discontinuous, making it difficult to represent using a straightforward mapping function. Meanwhile, network loss pattern fluctuates, meaning the historical observations cannot reliably reflect the future network conditions. *(iii)* Fine-grained FEC necessitates immediate determining and executing redundancy rate upon generating a new video frame (*e.g.*, 16.7 ms at FPS=60). The high frequency and stringent time constraints make existing learning-driven approaches impractical.

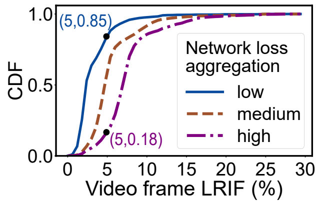
3 Design

3.1 Design Overview of Tooth

We first perform the field measurements to identify critical factors influencing the per-frame redundancy rate, including application-layer frame length and transmission-layer network loss pattern. Notably, in addition to the loss rate introduced in the conventional approaches [16], we based on the FEC operation to supplement loss pattern with loss aggregation, which further describes the temporal distribution of network packet loss events (§3.2). Then, we design Tooth as a decoupled dual-module architecture (Fig. 11), which can map the LRIF-related factors to per-frame redundancy rate and meanwhile minimize computational and inference time overheads. It consists of a *slow-module* and a *fast-module*, which circumvents the need for bulky large neural networks but



(a) Network loss rate lr.



(b) Network loss aggregation la.

Figure 12: Other related factors affecting video frame LRIF. Fig. 12b is obtained by analyzing the data with a network loss rate of ~1% in Fig. 12a. The network loss rate is analyzed from the RTCP feedback (every 100ms in our system).

also lowers the execution frequency, effectively tackling the computational complexities and inference time constraints in real-world environments. Specifically,

- *Slow-module* utilizes a compressed neural network model to discern network fluctuations and estimate future loss pattern. It does not need to execute frame by frame but is activated only upon receiving new RTCP network feedback to avoid unnecessary computational overhead (§3.3).
- *Fast-module* incorporates an exceedingly lightweight machine learning model to determine per-frame redundancy based on the frame length and future network loss patterns obtained from the *slow-module* (§3.4).

3.2 Investigating the Impact Factors of LRIF and Evaluating Strawman Solutions

We make an in-depth analysis of the large-scale measurement dataset in §2.2, and find that *besides the frame length, transport-layer network loss rate (lr) and loss aggregation (la) also significantly influence over video frame LRIF.* Next, we give more analysis: *(i)* in Fig. 12a, we plot the CDF of the video frame LRIFs related to network loss rate lr. The result clearly demonstrates a trend where frames experience increasing LRIF as the network loss rate escalates. For instance, when lr is 1%, 73% of the frames have LRIFs below 5%. However, when lr increases to 2% or higher, only 43% of the frames maintain LRIFs below 5%, marking a significant difference of 30%. *(ii)* Previous study [42] has noted that continuous packet loss severely impacts video frame recovery. However, we argue that for advanced FEC methods like Reed-Solomon codes [41], there is no distinction between losing n packets continuously or discretely within a single frame. Therefore, we introduce a novel metric, *i.e.*, network loss aggregation la, to reflect whether packet loss events will be more dispersed in many frames or concentrated in a few frames. la can be represented as follows,

$$la = \begin{cases} 0 & n \leq 1 \\ \dfrac{n}{\sum^{n} |t_n - centroid| + \alpha} & n > 1 \end{cases} \tag{1}$$

Here n represents the number of packets lost within a network feedback period, and t_n denotes the ideal arrival time of the n^{th} lost packet, measured in milliseconds. We define the timing

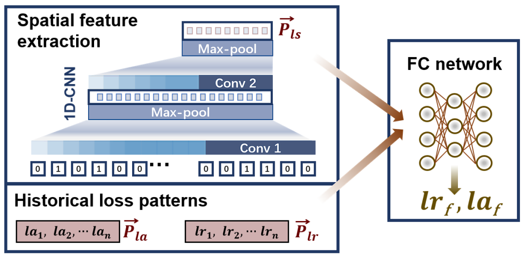


Figure 13: *Slow-module*'s model design.

centroid of all lost packets as $\frac{\sum^n t_n}{n}$, and α is set to 0.5 to avoid $\sum^n |t_n - centroid|$ being 0, which means packets are lost within 1ms. To intuitively illustrate the impact of la on LRIF, we briefly divide all la values into 3 levels according to the numerical distribution (*i.e.*, low: 0-1, medium: 1-5, high: 5-max) and categorize the video LRIFs. As shown in Fig. 12b, elevated la will increase the video frame LRIF. For instance, when la is low, 15% of frame LRIFs exceed 5%, but when la is high, this jumps to 82% of frame LRIFs exceed 5%.

Strawman solutions for fine-grained FEC. To assess the impact of video frame length (fl), network loss rate (lr), and loss aggregation (la) on per-frame redundancy rate, one preliminary approach is to manually establish the mapping relationship. However, this task proves to be non-trivial due to several challenges: *(i)* The relationships among the three factors and per-frame redundancy are not straightforwardly linear. Instead, they are implicit and complex, making manual modeling difficult. *(ii)* Both lr and la are subject to continual changes due to network fluctuations. Notably, there is also a dependent relationship where an increase in lr can sometimes lead to a higher la, though not invariably. Thus, it is imperative for Tooth to accurately determine per-frame redundancy amid varying network conditions.

Another strawman idea is to leverage powerful neural network models [18–20, 31]. Existing approaches generally operate at a coarser granularity, typically making decisions every 100 ms or longer [32, 42], which is inadequate for Tooth's fine-grained frame-by-frame optimization. On one hand, Tooth necessitates frequent redundancy execution within the tight frame encoding period of 16.7 ms at a frame rate of 60 FPS. On the other hand, Tooth must maintain exceptionally low overheads in both computation and inference time to enable feasible deployment in production environments. However, even neural network models with optimized architectures and parameters, such as pruning, often fail to satisfy these stringent overhead requirements, which will be validated in §5.3.

3.3 *Slow-module*: Estimating Future Network Loss Pattern

Slow-module is tasked with learning network fluctuations to estimate future network loss rate lr_f and loss aggregation la_f, and update them to the *fast-module*. Obviously, estimating la is very challenging since it is potentially affected by lr in addition to its own intrinsic characteristics. Existing simple methods, like arithmetic average of the heuristic observations, will bring serious deviations to the judgment of per-frame redundancy, as verified in §5.3. Thus, we introduce a NN model to build *slow-module*, and we provide more details next.

Input and output. As illustrated in Fig. 13, *slow-module*'s input is the historical network loss pattern vectors: the network loss rate sequence in the past n RTCP feedback, *i.e.*, $\vec{P_{lr}} = (lr_1, lr_2, ..., lr_n)$, the loss aggregation sequence ($\vec{P_{la}} = la_1, la_2, ..., la_n$), and the spatial distribution features $\vec{P_{ls}}$ of historical packet loss events. $\vec{P_{ls}}$ is extracted from the packet loss events in the past n network feedback periods. In particular, we encode the arrivals of all N packets in the past n RTCP feedback periods into a 0, 1 vector $\vec{ls}$, *i.e.*:

$$\vec{ls} = (x_1, x_2, ..., x_N), \quad x_i = \begin{cases} 1 & \text{packet lost} \\ 0 & \text{packet received,} \end{cases} \tag{2}$$

Here we build a 1D-CNN block to process $\vec{ls}$, which consists of two cascaded one-dimensional convolution layers with kernel sizes of 100 and 4, respectively. Each convolution layer in the 1D-CNN module uses *ReLU* activation and a pooling layer with a kernel size of 2 to extract the spatial features $\vec{P_{ls}}$.

The output of Tooth's *slow-module* is the future network loss rate (lr_f) and loss aggregation (la_f).

Neural network model. We pass the concatenated sequences of $\vec{P_{lr}}$ (10×1), $\vec{P_{la}}$ (10×1), and $\vec{P_{ls}}$ (12×1) to the *slow-module*'s cascaded FC networks, with 64, 32 and 16 units respectively. For the loss function, we emphasize that the overestimated lr_f and la_f will guide Tooth add excessive redundancy, increasing bandwidth cost. But the underestimated future lr_f and la_f may lead to insufficient redundancy, thus incurring stalls, more unacceptable than wasting bandwidth. Therefore, we set a larger loss for the underestimated case and the final loss function $\mathcal{L}_S$ is defined as follows:

$$\mathcal{L}_S = \frac{(lr_f - \hat{lr})^2 + (la_f - \hat{la})^2}{2} \times \left(\frac{e^{(\hat{lr} - lr_f) + (\hat{la} - la_f)}}{|\hat{lr} - lr_f + \hat{la} - la_f| + \beta}\right), \tag{3}$$

where $\hat{lr}$ and $\hat{la}$ are the true values of network loss rate and loss aggregation. The first part calculates the difference between the estimated lr_f, la_f and the true $\hat{lr}$, $\hat{la}$. For the second part, if lr_f, la_f are larger than $\hat{lr}$, $\hat{la}$, the loss can be appropriately reduced and when lr_f and la_f are smaller than $\hat{lr}$, $\hat{la}$, the loss will be enlarged. β is a constant and is set to 1 to avoid $|\hat{lr} - lr_f + \hat{la} - la_f|$ being 0.

3.4 *Fast-module*: Determining Per-frame Redundancy

Fast-module takes the future network loss rate lr_f, loss aggregation la_f, frame (to be encoded) length fl as inputs, and outputs the per-frame redundancy decision. In crafting *fast-module*, we gather lr, la, fl, and *LRIF* from historical frames as label to build sample dataset. Next, we incorporate a machine learning approach different from complex NN models to reduce the computation complexity.

Incorporating Random Forest model. We clarify *fast-*

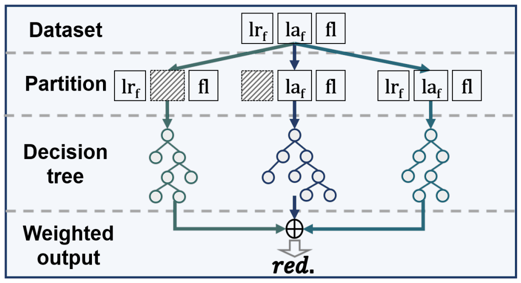


Figure 14: *Fast-module*'s model design.

module's inputs (lr, la, fl) exhibit the following characteristics: *(i)* High cardinality, especially lr and la boast hundreds of non-continuous potential values, while fl may also have dozens, thus rendering the feature space inherently variable and intricate. *(ii)* These features exert distinct and non-linear impacts on per-frame redundancy. For instance, a marginal 1% increase in lr may result in the loss of several more packets within a frame, whereas a similar increase in fl may not yield a substantial damage. In light of these issues, we leverage Random Forest (RF) model to construct *fast-module* as multiple LRIF decision trees. On one hand, RF model can effectively handle high-cardinality samples. Each redundancy decision tree of RF model will be built by randomly selecting training samples, ensuring strong generalization ability. On the other hand, each redundancy decision tree will split its nodes based on different input features during the establishment process, enabling to learn the degree to which each feature affects the per-frame redundancy. As shown in Fig. 14, we first partition the dataset based on lr,la, and fl, while ensuring that the elements in each dataset are evenly distributed to avoid situations where a large amount of data is concentrated near a certain value. Subsequently, we adjust the features carried by each subset, and the explicit features carried by each subset are divided into (lr,fl), (la,fl), and (lr,la,fl). This strategy prevents the decision trees from overly relying on a certain feature. Moreover, we choose MSE (Mean Squared Error) as the *criteria* of RF model to train the *fast-module*.

The final redundancy decision of the RF model is taken as the average of all decision trees' outputs. We emphasize that *(i)* the combination of multiple decision trees can help *fast-module* learn the non-linear relationship between lr, la, fl, and per-frame redundancy. *(ii)* Each tree examines distinct features and autonomously assesses the significance of each input feature, culminating in the integration of multiple decision trees' outputs. This can significantly enhance *fast-module*'s ability to generalize across varied network conditions and gaming environments, as validated in §5.2.

4 Implementation

Tooth's implementation workflow. As the workflow illustrated in Fig. 15, Tooth serves as a pivotal control module of the game server, positioned between Video Codec and FEC Codec. For each gaming session, Tooth acquires the receipt

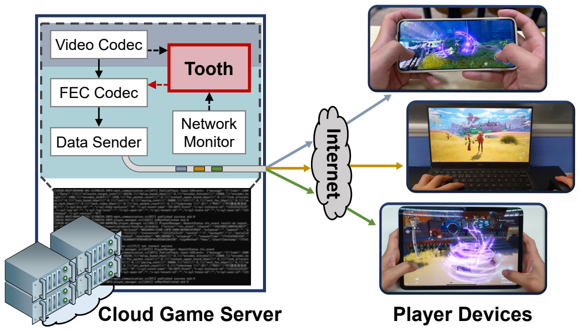


Figure 15: The implementation of Tooth.

information of past data packets from Network Monitor and the frame length from Video Codec, subsequently controlling FEC Codec to add redundancy for each newly generated frame. In terms of data flow, the server generates the real-time game content, which is encoded by Video Codec and FEC Codec. Then, Data Sender transmits the data packets to players' devices (*e.g.*, smartphone, laptop, iPad) via the Internet.

Tooth's dual-module training methodology. In §2, we have collected extensive cloud gaming traces from the production environment, which records *(i)* transmission and reception details of transport-layer RTP packets. From which, we count the network loss rate and aggregation at a granularity of 100ms to construct the network loss pattern dataset; *(ii)* Each packet's sequence number and the associated video frame ID, locations of all lost packets. For each frame, we count its length, number of lost packets, and integrate its current network loss pattern to construct the video frame LRIF dataset. Based on these two datasets, we complete the training of Tooth.

System implementation details. *(i) Tooth's integration.* Our commercial cloud gaming system has been running online for nearly two years and has more than 100,000 players. It follows the RTP protocol stack [44] and is implemented based on WebRTC [9, 10]. On game server, there is a real-time streaming service (RTSS) module, responsible for handling the game video content and streaming it to the client. Tooth is integrated as an RTSS sub-module, embedded into RTSS's media and network stack to *1)* obtain the frame information and network information *2)* return to RTSS main process the redundancy rate decision. *(ii) Redundancy encoding method.* We leverage the Reed-Solomon (RS) codes [41] to build FEC Codec. Through our engineering refinement, it empowers an FEC block to handle up to 256 packets and the encoding time consistently remains below merely 1 ms. Also, we use WebRTC's FlexFEC codec to support existing FEC solutions. For protection of uplink client-to-server data flow, see Appendix C. *(iii) Network Monitor* discerns the RTP packets' arrival via the client's RTCP feedback (per 100 ms) then provides it for Tooth as the basis for FEC decision-making. Moreover, it calculates the frame LRIFs and archives comprehensive network metrics (*e.g.*, RTT, throughput) into the trace database based on the MQTT protocol [4]. These massive network traces facilitate continuously optimizing Tooth in the offline environment.

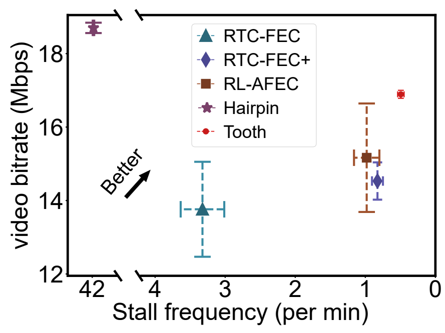


Figure 16: Interaction QoEs of all baseline methods. Tooth exhibits the optimized performance in stall frequency and video bitrate.

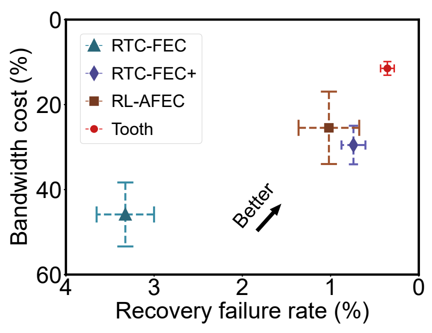


Figure 17: Redundancy rate and recovery failure rate of different FEC methods. Tooth recovers more frames with less redundancy.

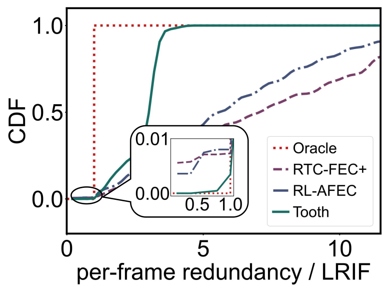


Figure 18: CDF of the ratio of per-frame redundancy rate to LRIF. A ratio of 1 means FEC adds the most appropriate redundancy to a frame.

5 Evaluation

5.1 Experimental Methodology

Comparison baselines. Here we compare Tooth with four state-of-the-art loss recovery baselines:

- *RL-AFEC* [18], which we have already introduced in detail in §2.2. Here we maintain its K as the recommended optimal value of 15 in [18] and provide the further ablation experimental results for other K values in the Appendix E.
- *RTC-FEC* [28], as the default FEC solution of WebRTC framework [9], is widely used in the industry. We integrate the m119 release from Chromium [10] in 2023 into our cloud gaming system.
- *RTC-FEC+*, which replaces *RTC-FEC*'s FlexFEC codec with more powerful RS codec [41]. We also tune *RTC-FEC+*'s redundancy table to suit the RS codec.
- *Hairpin* [36], which is the latest effort in interactive video streaming. It noticed that the loss of retransmission packets will bring multiple RTT overheads and ultimately incur stalls. So it adds redundancy to retransmission packets.

Evaluation metrics. For each cloud gaming session, we log the following metrics to conduct a comprehensive evaluation.

- Our direct optimization goals, *i.e.*, FEC performance, including *1)* bandwidth cost from FEC redundancy; *2)* recovery failure rate, which is the percentage of video frames that fail to recover when suffering packet loss. We clarify that adding redundancy will lower the video bitrate because production systems often have a bitrate cap (20 Mbps in our case) to control overall bandwidth costs, which is a common industry practice. Moreover, when FEC fails to recover a video frame, retransmission is used to ensure the data integrity of non-recoverable frames (whether I frame or P frame). Otherwise, the subsequent video frames won't play normally, leading to interaction stalls.
- Player interaction QoEs, including *3)* stall frequency, *i.e.*, how many times a player experiences stall per minute. We consider an interaction latency of more than 100ms as a stall, for which players will feel that the game is laggy; *4)* video bitrate, which does not include redundancy, represents better picture quality when it is higher; *5)* video PSNR and VMAF, which are close to the user-perceived feedback and have been widely adopted in the community and industry [18, 19, 31, 48].

To avoid the impact of differences in gaming session duration, we calculate these metrics at a one-minute granularity.

Field deployments. We deploy Tooth and baseline solutions to a production server, and conduct rigorous A/B tests in the real world. To ensure fair comparisons, we randomly apply an FEC solution to a gaming session instead of relying on manual allocation, and all solutions operate within the consistent system environment. The evaluation spans a duration of 6 weeks, and the players are in various activity scenarios (*e.g.*, library, office, cafe, home, park) with various wireless network conditions (*i.e.*, WiFi, 4G, 5G). We remove the first minute of each gaming session, during which the game is in the initial animation phase with no user interaction, and the system only monitors the user's network loss conditions.

5.2 System-level Evaluation

Tooth significantly enhances interaction QoE. Here we first focus on the two key application-layer QoE metrics, *i.e.*, stall frequency and video bitrate, to explore Tooth's QoE gains. As shown in Fig. 16, Tooth achieves the lowest stall frequency, *i.e.*, only 0.49 times per minute, reducing that of baseline FECs by 40.2%-85.2%. Furthermore, Tooth achieves the highest video bitrate, *i.e.*, 17.8 Mbps, which is 11.4%-29.2% higher than the baseline FECs. As a result, Tooth can enhance the state-of-the-art FECs' QoE in terms of both stall frequency and video bitrate. Next, we delve into all FECs' bandwidth costs and frame recovery results to further elucidate. As depicted in Fig. 17, Tooth can surpass existing FECs in reducing their bandwidth costs by 54.9%-75.0% while reducing the recovery failure rate by 51.9%-89.3%. This is attributed to Tooth sets higher redundancy rates for small frames to prevent recovery failures and lower redundancy rates for large frames to reduce unnecessary bandwidth costs. Since there are fewer small frames, the redundancy saved by large frames far exceeds the redundancy increased by small frames.

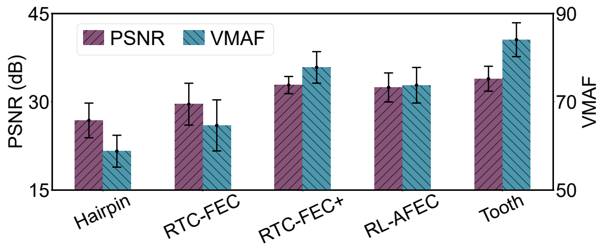


Figure 19: Video PSNR and VMAF results of different loss recovery methods.

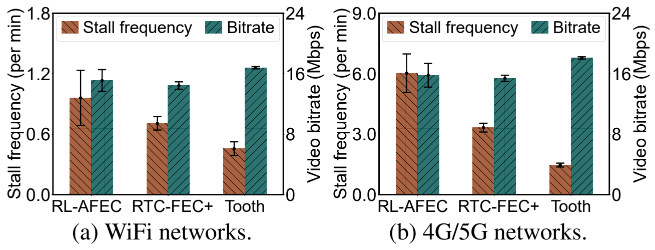


Figure 20: Interaction QoE under different network types. Tooth can decrease stall frequency more in 4G/5G networks.

To further clarify this, we calculate the ratio of per-frame redundancy rate and LRIF, where a ratio of 1 means that FEC adds the most appropriate redundancy to a frame. As shown in Fig. 18, Tooth is closest to the Oracle solution.

Regarding other baselines: *(i)* RTC-FEC, constrained by FlexFEC codec (mentioned in §5.1), behaves with a 45.9% bandwidth cost and 3.3% frame recovery failure rate. Consequently, RTC-FEC demonstrates a QoE of 3.3× stall frequency and a video bitrate of only 13.8 Mbps, inferior to Tooth. *(ii)* RTC-FEC+, optimizes RTC-FEC's bandwidth cost and frame recovery failure rate by 35.6% and 77.8%, respectively, thus improving QoE. *(iii)* RL-AFEC achieves a 13.7% lower bandwidth cost than RTC-FEC+ via reinforcement learning, but with a 35.2% degradation in recovery failure rate. These two competent coarse-grained FECs, RTC-FEC+ and RL-AFEC, can only fall into the inherent trade-off between saving bandwidth cost and ensuring recovery. *(iv)* For the retransmission-based Hairpin, its stall frequency is as high as 41.9 times per minute due to the additional RTT delays, despite improving video bitrate. Although Hairpin aims to recover all lost packets with only one round of retransmission, even this brings an additional RTT delay, which in our system will cause the interaction latency to be noticed. In the following sections, we will not delve further into the codec-limited RTC-FEC and retransmission-based Hairpin because they fail to meet the bitrate or stall frequency demands.

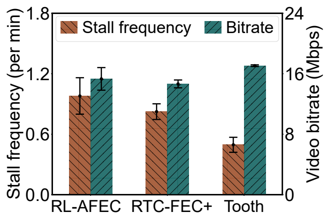
Tooth exhibits better video PSNR and VMAF performance. Note that video PSNR and VMAF require frame-by-frame comparison between the server-side source video and the client-side received video, which is not yet feasible in our current commercial platform. Therefore, we recruit 20 volunteers across campus to conduct a small-scale evaluation [4]. Specifically, we first build a local testbed using our commercial cloud gaming system, and randomly select 20 network traces (one per volunteer) from our network dataset to simulate the real network environments. Then, every volunteer is asked to initiate 5 cloud gaming sessions, with each session lasting 10 minutes and using different packet loss recovery methods. Finally, we collect 100 groups of gaming session videos and use the FFmpeg tool [6] to calculate their PSNR and VMAF results. As shown in Fig. 19, Tooth outperforms the baseline methods, with a 3.3% higher video PSNR and an 11.9% higher video VMAF than the next best RTC-FEC+.

Tooth is more effective in 4G/5G networks. Players on our game platform come from two network types, *i.e.*, WiFi and cellular[5], which represent different network conditions. WiFi networks exhibit lower RTT than 4G/5G networks, *i.e.*, 22.9ms vs. 35.1ms in average, which means the retransmission packets are more likely to timeout to incur stalls if FEC fails to recover the frames. Moreover, WiFi's network loss rate is 1.15% and 4G/5G's is 0.76%, *i.e.*, fewer frames will suffer packet loss in 4G/5G networks. In Fig. 20, we give the QoE results under these two networks. We conclude that: *(i)* From WiFi to 4G/5G networks, all three FECs' stall frequencies increase obviously due to higher RTT. For example, Rl-AFEC's stall frequency increases from 1.02 to stunningly 6.03. But Tooth can achieve more QoE gains, from reducing the baseline FECs' stall frequency by 35.5%-52.4% on WiFi to 55.9%-75.6% on 4G/5G. *(ii)* Due to the lower loss rate, RL-AFEC and RTC-FEC+ tend to decrease the redundancy rate, with their video bitrates increasing by 4.6% and 6.7%. Tooth can achieve a more significant bitrate increase of 7.7%. As proved in Fig. 7, the LRIF of large frames gradually approaches the network loss rate. So when the network loss rate greatly reduces, there are more bandwidth can be saved in large frames for Tooth, but which is something that uniform redundancy rate of coarse-grained FECs cannot achieve.

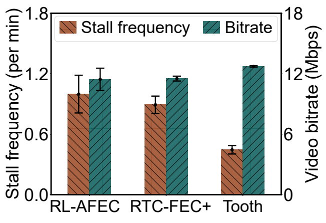
Tooth can still achieve significant QoE gains in 2D games whose frame length differences are smaller. As mentioned in Fig. 2, the frame length difference of 2D games is smaller overall than that of 3D games, with the maximum and minimum values being 50 and 80 respectively. An intuitive worry is: does Tooth still have QoE gains in 2D games? We break down the interaction QoE results of 3D games and 2D games to clarify the issue, as shown in Fig. 21. For 3D games, Tooth can achieve 39.9%, 49.5% lower stall frequency and 11.4%, 16.4% higher video bitrate than RTC-FEC and RL-AFEC, respectively (Fig. 21a). More importantly, for 2D games, Tooth still shows significant QoE gains over the baselines (Fig. 21b). In details, RL-AFEC and RTC-FEC+ both show slightly worsened stall frequency due to more small frames in 2D games,

[4]For campus volunteers, we only collect their operation instructions and video content from the cloud game client. This data collection process doesn't involve any user privacy information and has received approval from the ethical review committee.

[5]Our game APP cannot obtain detailed 4G and 5G access information from player devices, so they are unified into cellular players. In addition, the number ratio of WiFi players to cellular players is almost 8.5:1.



(a) 3D games.



(b) 2D games.

Figure 21: Tooth can not only benefit 3D game but also 2D games whose frame length differences are smaller.

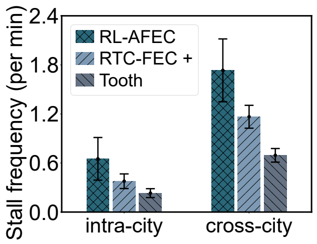


Figure 23: Stall frequency under different server distances.

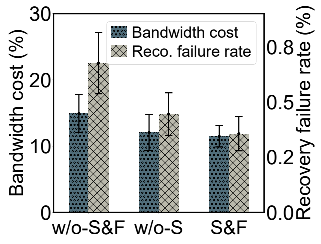


Figure 24: Ablation study of Tooth's two modules.

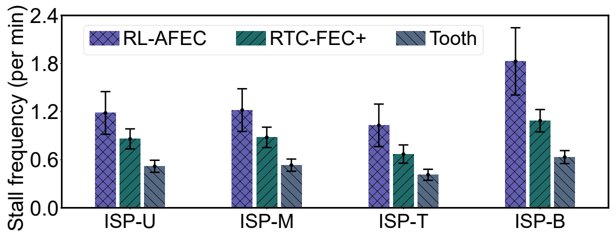


Figure 22: Tooth can achieve consistently lower stall frequency than baseline FECs under different ISP networks.

from 0.82 to 0.89 and from 0.97 to 0.99, respectively, while Tooth can further decrease the stall frequency from 0.49 to 0.44. In terms of video bitrate, Tooth still achieves considerable improvement, 10.4% and 11.1% higher than the baseline FECs. It can reduce the stall frequency by 76.2%, 77.8% and increases the video bitrate by 7.8%, 12.5% (Fig. 21b). Because in both 3D games and 2D games, frequent operations and dynamic game content will lead to inevitable variations in frame length. Thus, Tooth can exploit this difference to adaptively add appropriate redundancy to each frame.

Tooth exhibits exceptional stall performance in various ISP networks. Our cloud gaming platform encompasses players from all four major ISPs in China, *i.e.*, ISP-U (China Unicom), ISP-M (China Mobile), ISP-T (China Telecom), and ISP-B (China Broadnet), with player ratios of 1.1:3.1:1.8:0.01 respectively. Despite ISP variations, video bitrate gain remains largely unaffected, thanks to consistent CDN egress bandwidth set for each game stream (20Mbps in our system). Remarkably, there's a notable contrast in stall frequency. Illustrated in Fig. 22, players from ISP-T exhibit the lowest stall frequency, trailed by ISP-U and ISP-M, those from ISP-B encounter the highest stall frequency. Divergent stall performances stem from disparities in fundamental network performance and link load among ISPs: *(i)* ISP-B, in its nascent stage, is gradually enhancing its CDN links and base station infrastructure, showing relatively poor network RTT, average 28.0ms. *(ii)* The RTT conditions among ISP-M and ISP-U are comparable, 24.3ms and 23.8ms respectively, both are worse than ISP-T's 21.8ms. Fortunately, irrespective of the ISP networks, Tooth can achieve exceptional stall performance, reducing the stall frequency of the next best RTC-FEC+ by 1.63× to 1.72×.

Tooth can achieve consistent QoE gains in both intra-city and cross-city sessions. We divide gaming sessions into two categories: intra-city sessions and cross-city sessions, contingent upon whether the player and game server are in the same city, accounting for 17.2% and 82.8% of sessions respectively. As illustrated in Fig. 23, Tooth can decrease the stall frequency of the baselines by 40.5% and 59.9% in cross-city sessions. More importantly, in cross-city sessions, Tooth's stall performance can even match that of RL-AFEC in intra-city sessions, and it demonstrates greater robustness (with smaller error bar). Additionally, for intra-city sessions boasting superior network conditions, Tooth shows noteworthy stall frequency gains, reducing by 38.6% and 64.5% compared to the baselines.

5.3 Micro-benchmark Comparisons

Ablation study of Tooth's two modules. Here we examine the separate utility of Tooth's two modules. For *slow-module*, we build its heuristic replica with the arithmetic average method, estimating network loss rate lr and loss aggregation la as the averages over the last three RTCP feedback periods. For *fast-module*'s heuristic replica, we look for a past RTCP feedback period where the network loss pattern is closest to lr and la, then use fl as the observation window to observe the maximum number N of lost packets in this period. The replica will add N redundant packets to the frame. Next, we develop three variants of Tooth: w/o-S&F, w/o-S, S&F, leveraging identical cloud gaming session traces in the simulator for a fair comparison. The results in Fig. 24 show that: *(i)* w/o-S&F can yield 34.3% lower recovery failure rate than w/o-S, and meanwhile reduce the bandwidth cost by 19.2%. The S&F, employing a Random Forest model, demonstrates significantly superior accuracy in determining per-frame redundancy compared to simple historical observations. *(ii)* Compared to w/o-S, *slow-module* provides a more precise estimation of network loss patterns through the NN model. This enhancement allows *slow-module* to further diminish the recovery failure rate from 0.41% in w/o-S to 0.32% in S&F, marking a noteworthy 22.0% improvement. Additionally, we also evaluate the inference accuracy of *slow-module*, more details in Appendix F.

Importance of LRIF-related factors. In §3.2, we expand the LRIF-related factors beyond just frame length (fl) to include network loss rate (lr) and loss aggregation (la). To justify this expansion, we evaluate various combinations of these factors as inputs to Tooth's *fast-module*, specifically A (fl, la), B (fl, lr), and C (fl, lr, la). We train the corresponding *fast-module* variants with these inputs, naming them *fast*-A,

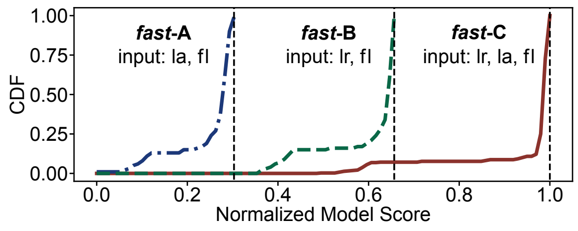


Figure 25: Importance of LRIF-related factors, *i.e.*, network loss rate (lr), loss aggregation (la), and frame length (fl).

Structure	GPU usage (%)	CPU usage (%)	Inference time (ms)	FEC encode time (ms)
Single-stage	7.9	7.1	3.6	0.5
Dual-module	1.4	1.8	0.7	0.5

Table 2: Computational and inference time overhead.

fast-B, and *fast*-C. To eliminate experimental contingency, each *fast-module* variant is trained with 400 different sets of model parameters, obtaining 1200 (3×400) Random Forest models. These models are then tested on the same dataset, with higher scores indicating more accurate LRIF judgment. Fig. 25 illustrates the CDF of model scores for each variant. It is evident that the maximum normalized scores of *fast*-A, *fast*-B, and *fast*-C are 0.30, 0.65, and 1.0 respectively. This highlights the importance of introducing lr and la as LRIF-related factors, especially la, which further emphasizes the subtle differences in network loss pattern that have not been considered by existing works [18, 20, 31, 49, 50].

Overhead comparison of Tooth's model structures. As discussed in §3.2, although it is feasible to directly use powerful neural networks to build Tooth as a single-stage structure, the high execution frequency (every 16.7ms) will inevitably lead to unacceptable computational and inference time overhead. Instead, the dual-module structure we designed decouples Tooth into two modules to reduce overhead. We test both structures on a game server (CPU: Xeon 8369HC; GPU: RTX 3060) and then present their detailed differences in Table 2. Obviously, by decoupling modules and reducing execution frequency, Tooth's dual-module structure can reduce GPU usage, CPU usage, and inference time by 5.6×, 3.9×, and 5.1× respectively. Both structures use RSFEC codec (mentioned in §4) and take 0.5ms for encoding the redundancy, which is a little lower than the decision-making time of Tooth's dual module, *i.e.*, 0.7ms. However, the single-stage decision-making time is 7.2× longer than the FEC encoding time, which obviously reduces its timeliness in commercial cloud gaming.

6 Related work

Network transmission optimization in cloud gaming. Traditional video streaming applications, such as video conferencing and live streaming [32, 34, 39, 46, 53], typically tolerate end-to-end network delays of several hundred milliseconds. In contrast, cloud gaming requires significantly lower latency, necessitating that the time from the player's action to the screen response consistently remains below 100 milliseconds [28, 43, 47]. Cloud gaming currently depends on congestion control algorithms [14, 15, 25, 26, 30, 52–54] to dynamically adjust the data transmission rate, aiming to minimize queuing delays of network packets. Our system employs a custom-optimized variant of GCC [17] for enhanced performance (see Appendix G). Despite these advanced measures, packet loss remains an inevitable challenge, consistently resulting in incomplete data delivery. To address this, cloud gaming systems have to implement effective loss recovery mechanisms, such as retransmission, represented by ART [33], Hairpin [36], and PTO [21]. However, retransmission mechanism commonly introduces additional delays—at least one RTT, and can lead to catastrophic interaction stalls (§5.2).

FEC-based loss recovery. FEC is initially employed to recover lost packets in audio transmissions [38, 40], and has recently been expanded to various video-related applications. For instance, CLOSET [50] and PATON [49] utilize FEC to reduce stalls during TCP-based video playback. DeepRS [20] and RL-AFEC [18] integrate FEC in real-time video communications to minimize end-to-end delays while optimizing FEC bandwidth costs. Additionally, R-FEC [31] and ABRF [19] combine FEC with congestion control to enhance the overall real-time video QoE. A recent advancement in videoconferencing, Tambur [42], introduces stream coding [27] to use multiple sequential frames to recover lost packets from earlier frames, but brings delay that spans multiple frame intervals. However, the above solutions all utilize coarse-grained redundancy mechanisms that do not satisfy the stringent interaction QoE requirements in cloud gaming scenarios (§5.2).

7 Conclusion

In this paper, we propose Tooth, an innovative fine-grained FEC method specifically designed for commercial cloud gaming. We highlight two main novelties: the first-of-its-kind large-scale measurements conducted in real-world environments to investigate the bottlenecks of existing loss recovery methods, and the design of fine-grained FEC encoding mechanisms to achieve an optimal balance between video QoE and redundancy bandwidth cost. Our online evaluations significantly demonstrate the advantages of Tooth. We believe that Tooth has the potential to benefit a wide range of latency-sensitive applications in the future.

Acknowledgements

We sincerely thank the anonymous reviewers and our shepherd, Quentin De Coninck for their insightful comments. Huanhuan Zhang, Anfu Zhou and Liang Liu are the corresponding authors. This work was supported in part by the NSFC Project (62225204, 62302058), the A3 Foresight Program of NSFC (62061146002), the Innovation Research Group Project of NSFC (61921003), the Beijing Major Science and Technology Project (Z231100007423014), and the Youth Top Talent Support Program.

References

[1] Nvidia video encoder programming guide v4.0. https://developer.download.nvidia.cn/compute/nvenc/v4.0/NVENC_VideoEncoder_API_ProgGuide.pdf, 2014.

[2] Alibaba cloud cdn pricing. http://docs-aliyun.cn-hangzhou.oss.aliyun-inc.com/pdf/cdn-billing-intl-en-2016-07-15.pdf, 2016.

[3] How much does it cost to maintain twitch infrastructure? https://trembit.com/blog/how-much-does-it-cost-to-maintain-twitch-infrastructure/, 2019.

[4] Mqtt protocol. [Online]. Available: https://mqtt.org, 2022.

[5] Cloud gaming market report. [Online]. Available: https://www.imarcgroup.com/cloud-gaming-market, 2023.

[6] Ffmpeg. https://www.ffmpeg.org/, 2023.

[7] H.265: High efficiency video coding. https://www.itu.int/rec/T-REC-H.265-202309-S/en, 2023.

[8] Nvidia video encoder programming guide v12.1. https://docs.nvidia.com/video-technologies/video-codec-sdk/12.1/nvenc-video-encoder-api-prog-guide/index.html#recommended-nvenc-settings, 2023.

[9] Webrtc homepage. [Online]. Available: https://webrtc.org/, 2023.

[10] Webrtc in chromium. [Online]. Available: https://chromium.googlesource.com/chromium/src/+/e8f60fd53d98c1b1f67397209066e1453f6957ab/, 2023.

[11] Binomial distribution. [Online]. Available: https://en.wikipedia.org/wiki/Binomial_distribution, 2024.

[12] Genshin impact·cloud. [Online]. Available: https://cloudgenshin.hoyoverse.com/en-us, 2024.

[13] Tower of fantasy. [Online]. Available: https://cloudbase.gg/g/tower-of-fantasy/, 2024.

[14] Soheil Abbasloo, Chen-Yu Yen, and H. Jonathan Chao. Classic meets modern: a pragmatic learning-based congestion control for the internet. In *SIGCOMM*, pages 632–647. ACM, 2020.

[15] Venkat Arun and Hari Balakrishnan. Copa: Practical delay-based congestion control for the internet. In *NSDI*, pages 329–342. USENIX Association, 2018.

[16] Simone Basso, Michela Meo, Antonio Servetti, and Juan Carlos De Martin. Estimating packet loss rate in the access through application-level measurements. In *W-MUST@SIGCOMM*, pages 7–12. ACM, 2012.

[17] Gaetano Carlucci, Luca De Cicco, Stefan Holmer, and Saverio Mascolo. Analysis and design of the google congestion control for web real-time communication (webrtc). In *MMSys*, pages 13:1–13:12. ACM, 2016.

[18] Ke Chen, Han Wang, Shuwen Fang, Xiaotian Li, Minghao Ye, and H. Jonathan Chao. RL-AFEC: adaptive forward error correction for real-time video communication based on reinforcement learning. In *MMSys*, pages 96–108. ACM, 2022.

[19] Sheng Cheng, Han Hu, and Xinggong Zhang. ABRF: adaptive bitrate-fec joint control for real-time video streaming. *IEEE Trans. Circuits Syst. Video Technol.*, 33(9):5212–5226, 2023.

[20] Sheng Cheng, Han Hu, Xinggong Zhang, and Zongming Guo. Deeprs: Deep-learning based network-adaptive FEC for real-time video communications. In *ISCAS*, pages 1–5. IEEE, 2020.

[21] Sheng Cheng, Han Hu, Xinggong Zhang, and Zongming Guo. Rebuffering but not suffering: Exploring continuous-time quantitative qoe by user's exiting behaviors. In *IEEE INFOCOM 2023 - IEEE Conference on Computer Communications*, pages 1–10, 2023.

[22] Mark Claypool and Kajal T. Claypool. Latency and player actions in online games. *Commun. ACM*, 49(11):40–45, 2006.

[23] Mark Claypool and Kajal T. Claypool. Latency can kill: precision and deadline in online games. In *MMSys*, pages 215–222. ACM, 2010.

[24] Matthias Dick, Oliver Wellnitz, and Lars C. Wolf. Analysis of factors affecting players' performance and perception in multiplayer games. In *NETGAMES*, pages 1–7. ACM, 2005.

[25] Mo Dong, Qingxi Li, Doron Zarchy, Philip Brighten Godfrey, and Michael Schapira. PCC: re-architecting congestion control for consistent high performance. In *NSDI*, pages 395–408. USENIX Association, 2015.

[26] Mo Dong, Tong Meng, Doron Zarchy, Engin Arslan, Yossi Gilad, Brighten Godfrey, and Michael Schapira. PCC vivace: Online-learning congestion control. In *NSDI*, pages 343–356. USENIX Association, 2018.

[27] Silas L. Fong, Ashish Khisti, Baochun Li, Wai-Tian Tan, Xiaoqing Zhu, and John G. Apostolopoulos. Optimal streaming codes for channels with burst and arbitrary

erasures. *IEEE Trans. Inf. Theory*, 65(7):4274–4292, 2019.

[28] Stefan Holmer, Mikhal Shemer, and Marco Paniconi. Handling packet loss in webrtc. In *ICIP*, pages 1860–1864. IEEE, 2013.

[29] Teemu Kämäräinen, Matti Siekkinen, Antti Ylä-Jääski, Wenxiao Zhang, and Pan Hui. A measurement study on achieving imperceptible latency in mobile cloud gaming. In *MMSys*, pages 88–99. ACM, 2017.

[30] Jaehong Kim, Youngmok Jung, Hyunho Yeo, Juncheol Ye, and Dongsu Han. Neural-enhanced live streaming: Improving live video ingest via online learning. In *SIGCOMM*, pages 107–125. ACM, 2020.

[31] Insoo Lee, Seyeon Kim, Sandesh Dhawaskar Sathyanarayana, Kyungmin Bin, Song Chong, Kyunghan Lee, Dirk Grunwald, and Sangtae Ha. R-FEC: rl-based FEC adjustment for better qoe in webrtc. In *ACM Multimedia*, pages 2948–2956. ACM, 2022.

[32] Jinyang Li, Zhenyu Li, Ri Lu, Kai Xiao, Songlin Li, Jufeng Chen, Jingyu Yang, Chunli Zong, Aiyun Chen, Qinghua Wu, Chen Sun, Gareth Tyson, and Hongqiang Harry Liu. Livenet: a low-latency video transport network for large-scale live streaming. In *SIGCOMM*, pages 812–825. ACM, 2022.

[33] Tong Li, Wei Liu, Xinyu Ma, Shuaipeng Zhu, Jingkun Cao, Senzhen Liu, Taotao Zhang, Yinfeng Zhu, Bo Wu, and Ke Xu. ART: adaptive retransmission for wide-area loss recovery in the wild. In *ICNP*, pages 1–11. IEEE, 2023.

[34] Xianshang Lin, Yunfei Ma, Junshao Zhang, Yao Cui, Jing Li, Shi Bai, Ziyue Zhang, Dennis Cai, Hongqiang Harry Liu, and Ming Zhang. Gso-simulcast: global stream orchestration in simulcast video conferencing systems. In *SIGCOMM*, pages 826–839. ACM, 2022.

[35] Zili Meng, Yaning Guo, Chen Sun, Bo Wang, Justine Sherry, Hongqiang Harry Liu, and Mingwei Xu. Achieving consistent low latency for wireless real-time communications with the shortest control loop. In *SIGCOMM*, pages 193–206. ACM, 2022.

[36] Zili Meng, Xiao Kong, Jing Chen, Bo Wang, Mingwei Xu, Rui Han, Honghao Liu, Venkat Arun, Hongxin Hu, and Xue Wei. Hairpin: Rethinking packet loss recovery in edge-based interactive video streaming. In *USENIX NSDI*, 2024.

[37] Zili Meng, Tingfeng Wang, Yixin Shen, Bo Wang, Mingwei Xu, Rui Han, Honghao Liu, Venkat Arun, Hongxin Hu, and Xue Wei. Enabling high quality real-time communications with adaptive frame-rate. In *NSDI*, pages 1429–1450. USENIX Association, 2023.

[38] Marcin Nagy, Varun Singh, Jörg Ott, and Lars Eggert. Congestion control using FEC for conversational multimedia communication. In *MMSys*, pages 191–202. ACM, 2014.

[39] Zipeng Pan, Yuan Zhang, Tao Lin, and Jinyao Yan. Liveae: Attention-based and edge-assisted viewport prediction for live 360° video streaming. In *EMS@SIGCOMM*, pages 28–33. ACM, 2023.

[40] Colin S. Perkins, Orion Hodson, and Vicky Hardman. A survey of packet loss recovery techniques for streaming audio. *IEEE Netw.*, 12(5):40–48, 1998.

[41] James S Plank. A tutorial on reed–solomon coding for fault-tolerance in raid-like systems. *Software: Practice and Experience*, 27(9):995–1012, 1997.

[42] Michael Rudow, Francis Y. Yan, Abhishek Kumar, Ganesh Ananthanarayanan, Martin Ellis, and K. V. Rashmi. Tambur: Efficient loss recovery for videoconferencing via streaming codes. In *NSDI*, pages 953–971. USENIX Association, 2023.

[43] Sandesh Dhawaskar Sathyanarayana, Kyunghan Lee, Dirk Grunwald, and Sangtae Ha. Converge: Qoe-driven multipath video conferencing over webrtc. In *SIGCOMM*, pages 637–653. ACM, 2023.

[44] Henning Schulzrinne, Stephen L. Casner, Ron Frederick, and Van Jacobson. RTP: A transport protocol for real-time applications. *RFC*, 3550:1–104, 2003.

[45] Susanna Schwarzmann, Nick Hainke, Thomas Zinner, Christian Sieber, Werner Robitza, and Alexander Raake. Comparing fixed and variable segment durations for adaptive video streaming: a holistic analysis. In *MMSys*, pages 38–53. ACM, 2020.

[46] Ziyi Wang, Yong Cui, Xiaoyu Hu, Xin Wang, Wei Tsang Ooi, and Yi Li. Multilive: Adaptive bitrate control for low-delay multi-party interactive live streaming. In *INFOCOM*, pages 1093–1102. IEEE, 2020.

[47] Bingyang Wu, Kun Qian, Bo Li, Yunfei Ma, Qi Zhang, Zhigang Jiang, Jiayu Zhao, Dennis Cai, Ennan Zhai, Xuanzhe Liu, and Xin Jin. XRON: A hybrid elastic cloud overlay network for video conferencing at planetary scale. In *SIGCOMM*, pages 696–709. ACM, 2023.

[48] Jiangkai Wu, Yu Guan, Qi Mao, Yong Cui, Zongming Guo, and Xinggong Zhang. Zgaming: Zero-latency 3d cloud gaming by image prediction. In *SIGCOMM*, pages 710–723. ACM, 2023.

[49] Jiyan Wu, Bo Cheng, Ming Wang, and Junliang Chen. Priority-aware FEC coding for high-definition mobile video delivery using TCP. *IEEE Trans. Mob. Comput.*, 16(4):1090–1106, 2017.

[50] Jiyan Wu, Chau Yuen, and Junliang Chen. Leveraging the delay-friendliness of TCP with FEC coding in real-time video communication. *IEEE Trans. Commun.*, 63(10):3584–3599, 2015.

[51] Xiaokun Xu and Mark Claypool. Measurement of cloud-based game streaming system response to competing TCP cubic or TCP BBR flows. In *IMC*, pages 305–316. ACM, 2022.

[52] Hyunho Yeo, Chan Ju Chong, Youngmok Jung, Juncheol Ye, and Dongsu Han. Nemo: enabling neural-enhanced video streaming on commodity mobile devices. In *MobiCom*, pages 1–14, 2020.

[53] Huanhuan Zhang, Anfu Zhou, Jiamin Lu, Ruoxuan Ma, Yuhan Hu, Cong Li, Xinyu Zhang, Huadong Ma, and Xiaojiang Chen. Onrl: improving mobile video telephony via online reinforcement learning. In *MobiCom*, pages 29:1–29:14. ACM, 2020.

[54] Anfu Zhou, Huanhuan Zhang, Guangyuan Su, Leilei Wu, Ruoxuan Ma, Zhen Meng, Xinyu Zhang, Xiufeng Xie, Huadong Ma, and Xiaojiang Chen. Learning to coordinate video codec with transport protocol for mobile video telephony. In *MobiCom*, pages 1–16, 2019.

Appendices

A Measurement Method for Application-layer Interaction Latency

We track user interaction delays frame by frame to identify whether a video frame is delayed and by how much. The interaction latency refers to the time it takes from when a user inputs an operation (*e.g.*, run and hit) into the game client to when the user receives the screen feedback. To measure this latency actually, we break down the cloud gaming application-layer latency into two parts:

(i) System processing delay, which involves tasks like capturing user input, video encoding, decoding, and playback on the game client and server. These contribute to a total average delay of 32ms, which is relatively stable.

(ii) Data round-trip network delay, which involves transmitting user operations to game server (uplink) and receiving all RTP packets of the responded video frame from the game server to the client (downlink) via the Internet. These delays depend on network transmission conditions, we record frame ID and the timestamps of all uplink and downlink packets to calculate them.

If the sum of these two parts of delays exceeds 100ms, we determine that the user interaction has timed out, in the same way, we can know how late the complete frame arrives.

B Applying Infinite GOP Video Encoding in Cloud Gaming

Traditional real-time video communication usually uses the finite GOP encoding manner and sets a fixed GOP length for the video codec, such as 30 set in RL-AFEC [18]. For cloud gaming, this ultra-low latency interactive video streaming application, a common practice in the industry is to adopt the infinite GOP video encoding [1, 8]. Infinite GOP video encoding disables the server to send I frames periodically, and the client is allowed to request a new I frame only for necessary error recovery, *e.g.*, the client frame buffer pool overflowed after multiple video frames suffering packet loss and failing to be decoded. As a result, our cloud gaming system sends an I frame every 4.7 minutes on average, which means our average GOP length is 16,920 and 564× higher than that of conventional video streaming in [18].

We explain that the P frame only needs to encode the difference between the current video frame and the previous one, while the I frame must encode the entire video frame. When encoding the next video frame, the size of a P frame will not exceed that of an I frame, even if the cloud gaming scene changes. Therefore, the infinite GOP encoding can avoid frequently sending high-load I frames and the accompanying latency surge, and is recommended by the popular NVENC video codec [1, 8] for low-latency cases, especially game-streaming, video conferencing, *etc*. We emphasize that multiple top gaming platforms, including our partner company W, adopt this encoding method to optimize the cloud gaming interaction latency. This is a common practice in industry although less frequently introduced in academic papers.

C Protection of uplink client-to-server data

The client-to-server data flow in cloud gaming, mainly discussing player operation data here, typically ranges from a few to several tens of Kbps, which is much smaller than the available network bandwidth. For such a small operation data flow, there are no multiple concurrent packets like server-to-client video flow. Therefore, the industry usually doesn't add redundancy using FEC rather than adopting some repeated delivery strategies (*i.e.*, sending data replicas) to ensure reliable transmission. Take our system as an instance, a single player operation amounts to no more than 10 bytes, and the total number of concurrent operations remains far below the maximum capacity of an RTP packet (MTU=1500). Thus, an operation RTP packet from the client will not only contain the latest player operations but also the previous *n* operations. Additionally, each operation RTP packet will be sent twice, with a 2ms interval, to further improve reliability.

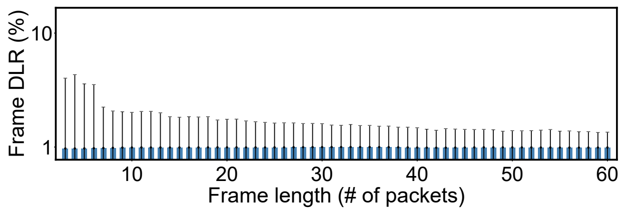


Figure 26: LRIF results of all video frames (including those experiencing no losses) observed in our commercial cloud gaming platform.

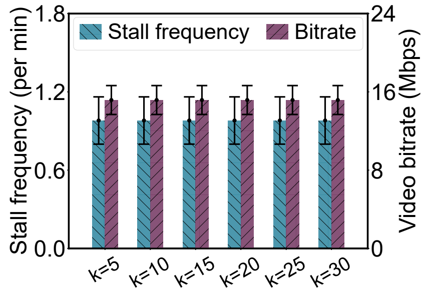


Figure 27: Ablation experiment results of the RL-AFEC model with different K values.

D A Deeper Dive into LRIF Results

We make the following explanations to clarify why in Fig. 7 we need analyze the LRIFs of frames that suffered packet loss rather than all GOP frames. *(i)* FEC in video streaming is more interested in higher percentiles of LRIF because only they can reflect how many redundant packets that FEC should add to protect all video frames as much as possible from network packet loss events; *(ii)* As shown in Fig.26, the average and standard deviation of all GOP frame LRIFs are actually consistent with the overall network loss rate (about 1%), but obviously, FEC should not adjust the redundancy rate based on this insight. So, in Fig. 7, we plot the LRIF average and std of video frames that suffered packet loss to reveal the relationship between non-zero LRIF and video frame length, which is more meaningful for guiding us in designing the fine-grained FEC in cloud gaming.

E Ablation Study of RL-AFEC Using Different K Values

For RL-AFEC [18], we further present ablation study experiments with the variation of K, *i.e.* setting K as 5, 10, 15, 20, 25, 30, consistent with [15]. As shown in Fig. 27, the six candidate values of K have little impact on the cloud gaming interaction QoE, which can be almost ignored. That's because the GOP length of cloud gaming streaming is very large, it is far from enough to optimize only a few K critical frames. We should set a fine-grained redundancy rate for each video frame.

More importantly, we emphasize that directly defining K to be GOP size N with RL-AFEC will not yield a similar

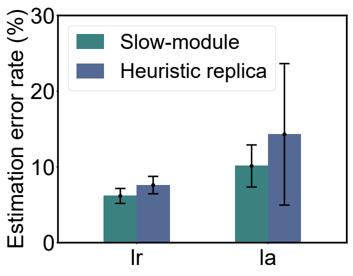


Figure 28: Estimation errors of network loss rate *lr* and loss aggregation *la*.

outcome as Tooth. We elaborate on the key differences in the design principles of Tooth and RL-AFEC: *(i)* RL-AFEC is designed for common video call scenarios, where the length of the remaining P frames remains relatively uniform, apart from the initial I frame in each GOP. In other words, RL-AFEC does not account for significant frame length variations and the resulting huge LRIF differences in cloud gaming streaming. As a result, RL-AFEC may fall short of fully meeting the redundancy adaptation design required for specific cloud gaming scenarios. *(ii)* In RL-AFEC, the GOP length N is 30 and the authors conclude that the optimal K value is 15, the reason is that increasing K will cause its reinforcement learning model to explore huge action space, resulting in an obvious decrease in its model effectiveness. Especially considering that the GOP size N will be hundreds even thousands of times larger than K in cloud gaming scenarios, defining K to be N is even less feasible. Instead, Tooth, with its unique dual-module design, makes larger K or even infinite K feasible, and ultimately is fully capable of setting fine-grained redundancy rate for each frame.

F Micro-benchmark Experiments on Tooth's Slow-module

To further evaluate Tooth's *slow-module*, we analyze Tooth online dataset to supplement the micro-benchmark experiment results, including the estimation errors (mean and std) of network loss rate *le* and loss aggregation *la*. Then, we calculate the mean and std of all estimation error rates. Moreover, we also replay these online traces in the simulator and evaluate *slow-module*'s heuristic replica (implementation details in §5.3) to compare their performance differences. As shown in Fig. 28, Tooth's *slow-module* behaves good inference accuracy, with an average error of 6.2% for *lr* and 10.1% for *la*, which are 18.8% and 29.2% lower than the heuristic replica, respectively.

G Fine-tuning of GCC's hyper-parameters in our system

The default GCC congestion control algorithm in WebRTC [10] isn't fully adapted for the streaming characteristics of cloud gaming. We follow the algorithm logic of GCC but customize its hyper-parameters based on practical engineering experiments: (i) We increase GCC's upper bitrate limit to 20Mbps to meet the production streaming requirement of cloud gaming; (ii) We shorten the observation window length in GCC's trend-line estimator from 20 to 15, making the system more aware of delay increases; (iii) We slightly enlarge the rate increase factor in AIMD-based bitrate control from 1.08 to 1.11 for fast bitrate recovery after the delay stabilizes; (iv) We disable GCC's probe action during bitrate reduction since cloud gaming is more delay-sensitive than bandwidth-sensitive. Probing at this time can easily harm delay performance. (v) When the observed network loss rate is below 0.3%, we block GCC's loss-driven decision on reducing bitrate and rely solely on the delay-driven bitrate adjustments.