From Source to Solution: Tackling Packet Losses in Large-scale Cloud Gaming Systematically and Precisely

Jing Wang*, Xiao Kong* , Yunzhe Ni, Nian Wen, Jiaxing Zhang, Congcong Miao, Honghao Liu†
Tencent Inc.

Abstract

Cloud gaming requires all video frames to be delivered before a stringent delay deadline to ensure seamless gaming experience. However, meeting this requirement is challenging due to packet losses, which greatly magnifies the frame delay. Various FEC-based loss recovery schemes were recently proposed to address the packet loss issue. However, the source of such packet losses remains unrevealed. Our production measurement results from *Tencent START* cloud gaming platform have shown that 66.5% of packet losses are caused by network infrastructure's preferences against UDP and network congestion. Moreover, off-the-shelf video streaming systems like WebRTC could not detect retransmission loss efficiently. These issues completely nullified the performance gain of loss recovery schemes. To address this, we design and implement LADR, which combines loss avoidance, detection, and recovery to tackle packet losses. LADR incorporates the loss-based and delay-based congestion control algorithms and adopts RACK-TLP for loss avoidance and detection. Furthermore, LADR adopts an opportunistic FEC scheme to perform loss recovery. LADR has been rolled out at *Tencent START* cloud gaming platform, a large-scale cloud gaming provider, for one year. Production measurement results show that LADR only suffers from 0.049% packet loss rate (-59.8% vs. existing solutions) and delivers 99.87% of video frames within 100 milliseconds.

1 Introduction

Cloud gaming, which allows users to play games remotely by streaming gameplay video and user input between local and remote machines, has been proven as a promising approach of bringing resource-hungry video games to low-end devices without expensive graphics cards or large external storage [1–5]. Different from traditional livestreaming applications , cloud gaming further requires the frame delay [1] to be *consistently* low to support seamless interactions between the user and the game application. As shown in Fig.1, the frame delay could be coarsely broken into three parts: (1) propagation delay is the minimum time needed for packets to pass through the network; (2) queuing delay is the time packets spent in network queues, and (3) recovery delay is the additional time spent on retransmitting. The propagation delay could be reduced to 10-20 ms [6] by pushing the cloud gaming servers toward the network edge, while the queuing delay has been reduced to less than 17 ms [7] by applying proper congestion control (CC) algorithm. However, retransmissions, whenever required, can introduce a large delay penalty proportional to the sum of propagation delay, queuing delay, and an additional loss detection delay. Our production measurement results show that recover delay is the direct reason why 66.2% of *>100 ms frames* (Tab.1) missed their 100 ms delivery deadline — In practice, frames with delay over 100 ms would be considered as stalls [7] that severely hurt the user experience.

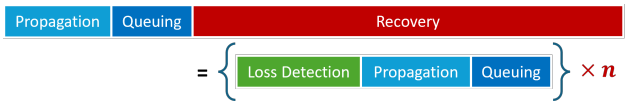


Figure 1: Frame delay breakdown.

On the popular livestreaming platform WebRTC [8], which is widely adopted by various cloud gaming applications including Xbox Cloud Gaming [3], Amazon Luna [4] and Pioneer [5], packet losses are handled by both retransmissions and forward-error correction (FEC). By proactively adding encoded packets, FEC could avoid retransmissions when encountering packet losses and significantly reduce recovery delay. Due to the fact that FEC takes up a lot of network bandwidth and computing resources, WebRTC only enables it after packet losses happened [9], resulting in long recover delay. Our measurement results of WebRTC have shown that the average recover delay of >100 ms lossy frames is 54.0 ms, which contributed 28.4% of the total frame delay. To reduce recovery delay, various solutions [6, 10–18] were proposed to optimize the performance of the FEC-based loss recovery scheme. However, our large-scale production measurement results from *Tencent START* cloud gaming platform reveal that the potential of reducing recovery delay is not limited to the loss recovery process itself, but exists at every stage from the source of the packet loss to its eventual resolution. Specifically, we have carried out a thorough measurement covering ∼200,000 hours on *Tencent START* cloud gaming, examining various components involved in the video delivery process, including the transport protocol, congestion control algorithm, loss detection scheme, and loss recovery scheme.

*These authors contributed equally to this work.
†Corresponding authors.
[1] The total end-to-end latency experienced by a data frame from the moment the frame is created at the sender to the moment the sender successfully receives a corresponding acknowledgment from the client.

Our key observations are summarized below:

• **Loss rate of different transport protocols varies (§2.3).** By default, WebRTC transfers video traffic over UDP sockets. However, even with the same CC algorithm and loss recovery scheme, the loss rate of TCP packets is 44.8% lower than that of UDP[2] ones, confirming that the Internet infrastructure is not UDP-friendly and that using UDP will result in unnecessary packet losses. Thus, *using a transport protocol that is not ill-treated by the network would greatly reduce packet losses.*

• **Congestion losses are common (§2.4).** Lowering the video bitrate of our cloud gaming sessions to a fixed 2 Mbps would reduce the overall packet loss rate by 39.4% indicates that congestion is still an important cause of packet losses. Also, congestion is typically related to shallow in-network buffers whose size is only several milliseconds worth of traffic, *i.e.* would easily be overfilled. Such congestion often causes a loss rate as high as 50% and possibly cause repeated losses. Therefore, *avoiding congestion would also significantly reduce packet losses, and more importantly, reduce heavy losses that require plenty of redundant traffic to perform recovery*.

• **Loss of retransmitted packets is not efficiently detected (§2.5).** The NACK-based loss detection mechanism in realtime video transport systems like WebRTC [8] and LiveNet [19] could not determine whether the sender retransmitted a packet or not. Therefore, to trigger retransmission more than once, they wait for a RTT-related timeout to send redundant NACKs. Therefore, *a proper loss detection scheme is needed for the loss recovery mechanism to know about packet loss events in a timely manner*.

In fact, 1) 66.5%[3] of packet losses, including heavy losses that could only be recovered with high bandwidth cost, could be avoided instead, and 2) the most important input of loss recovery schemes, loss events, are not correctly sensed. Therefore, to address the packet loss issue, we advocate for a *systematic* approach that not only recovers from packet losses efficiently, but also avoids losses skillfully and detects losses correctly in this paper. Based on this idea, we propose Loss Avoidance-Detection-Recovery (LADR), a novel design framework and an all-in-one solution that tackles packet losses from their source to the corresponding countermeasures (§3.1). It contains four co-operated components:

• **Static loss avoidance.** Co-existing TCP and UDP flows for data transmission to leverage the low loss rate of TCP packets, while preserving the possibility to use FEC (§4);

• **Runtime loss avoidance.** Delay-based CC algorithm [7] to avoid congestion in normal cases and CUBIC [20]-like loss-based CC algorithm to reduce congestion loss when shallow in-network buffers are detected (§3.2);

• **Loss detection.** SACK-based loss detection integrating RACK-TLP [21] for quickly detecting tail packet losses and loss of retransmitted packets (§2.5);

• **Loss recovery.** Sending rate limiting scheme and opportunistic FEC scheme (§3.3) to avoid repeated losses and perform loss recovery upon loss events (§3.4).

We implement LADR on *Tencent START* cloud gaming platform and LADR is in deployment since March, 2024. The deployment data of LADR shows that LADR only suffers from 0.049% loss rate and 0.115% >100 ms frames . Compared to WebRTC, the state-of-the-art livestreaming system and Hairpin, the state-of-the-art FEC-based loss recovery scheme, our result shows that the portion of frames with loss and the portion of >100 ms frames of LADR are 71% and 61% lower than the baseline solutions, respectively. (§5).

The contributions of this paper are summarized as follows:

• We reveal various issues that greatly affect the efficiency of loss recovery schemes through our measurement campaign, and motivate the new idea of addressing packet loss issues through carefully co-design of loss avoidance, loss detection and loss recovery schemes.

• We design and implement LADR, the solution that tackles packet losses in cloud gaming, and generalize its design principle into a design framework for handling packet losses.

• We evaluate and deploy LADR on *Tencent START* cloud gaming platform, demonstrating its effectiveness through extensive experiments.

Ethical claim. All user data collected in this work are obtained with explicit permission from the users and are anonymized to protect their privacy. This work does not raise any ethical concerns and conforms to the IRB policies of the authors' institutions.

2 Cloud Gaming Performance Measurement

To track down the source of packet losses in cloud gaming and derive possible solutions of the packet loss issue, we measure the performance and analyze the behavior of cloud gaming system on *Tencent START* cloud gaming platform.

2.1 Measurement Methodology

Video transport system implementation. On *Tencent START* cloud gaming platform, we carefully implemented our video transport system in a modularized manner that allows us to replace individual components without affecting the behavior of the whole system. Our video transport system contains four major replacable components:

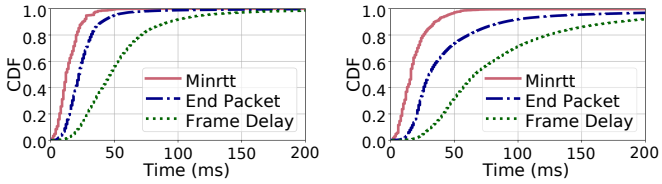
• *Transport protocol* provides unified interface for transferring video contents. TCP and RUDP-a Reliable version of UDP-are available for our system. We implement RUDP by mimicking the behavior of kernel TCP.

• *Congestion control* controls the maximum sending rate. Pudica [7], Copa [22], Salsify [23], and SQP [24] are available.

• *Loss detection* monitors the state of the transport protocol and reports loss events accordingly. NACK flavored with WebRTC [25] and SACK-based loss detection is available. The SACK-based loss detection is based on TCP SACK [26],

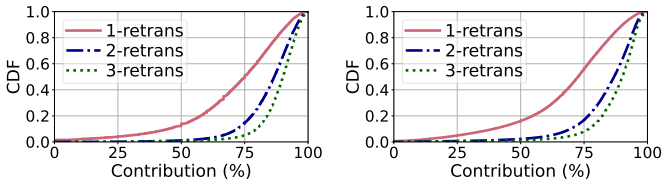
[2]Extended by us to support TCP-style reliable transport.

[3]Equals to 1 - (1 - 44.8%) × (1 - 39.4%).



(a) Wired. (b) Wi-Fi.

Figure 2: Dist. of delay composition of lossy frames.



(a) Wired. (b) Wi-Fi.

Figure 3: Contribution of recovery in remaining frame delay.

integrated with RACK-TLP [21] to support detection of retransmitted packets and tail packets.

• *Loss Recovery* listens for loss events and performs recovery for corresponding packets. Plain retransmission, WebRTC-flavored FEC [9] and Hairpin [6] are available. Here we note that FEC-based recovery schemes could only work with RUDP, and we use Reed-Solomon code [27] instead of the default XOR code in WebRTC to improve performance.

In the rest of this section, we study the impact of each component on the performance of cloud gaming system by alternating the implementation of each component. **When studying the impact of one component, we use the default implementation for the remaining ones unless otherwise stated.** Our default transport protocol, congestion control, loss detection and loss recovery schemes are TCP, Pudica, SACK-based loss detection and plain retransmission, respectively.

Types of users. Different users access our cloud gaming service with different types of Internet connections. Regarding the different characteristics of different networks, we divide our users into two groups according to the network types: *Wired* and *Wi-Fi*. Measurement data of different groups are presented separately. We note that *Tencent START* cloud gaming is capable of delivering video via multipath transport, to avoid bias caused by dividing video traffic onto multiple network paths, we only consider single-path users. Subsequently, we do not consider cellular networks in this paper, for we measure the average raw delay of cellular networks to be 27.2 ms, which is too large to meet the QoE requirements of cloud gaming without multipath transport functionality.

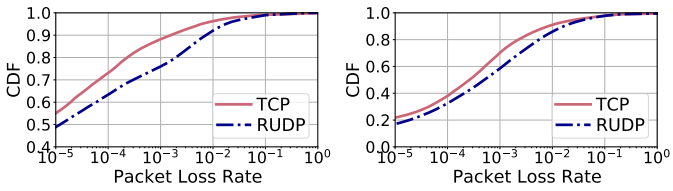
Definition of terminologies. We use various terminologies in our analysis on frames and packet losses throughout this paper. We list the terminologies and their definitions in Tab.1.

2.2 Packet Losses vs. Frame Delay

Before presenting our analysis of the packet loss issue, we first summarize the data collected using the default implementation of our video transport system to assess the impact of

(a) Wired. (b) Wi-Fi.

Figure 4: Delay contribution to frames > 100ms.



(a) Wired. (b) Wi-Fi.

Figure 5: Dist. of packet loss rate.

packet loss on cloud gaming performance. For wired (Wi-Fi) users, 0.77% (0.71%) of frames are lossy frames, and 1.07% (1.24%) are loss-affected frames. To quantify this impact, we decompose frame delay into propagation delay, queuing delay, and recovery delay. We estimate propagation delay with the minimum observed RTT and calculate delays from non-loss factors based on the time from the start of frame transmission to the receipt of the last non-lost packet. We plot the minimum RTT as the estimated propagation delay and the RTT of the end packet (i.e. last non-lost packet in a lossy frame). As shown in Fig.2, for lossy frames, propagation and queuing delay only constitute a small portion of the frame delay. Meanwhile, Fig.3 shows that the recovery delay constitutes 71.6% (58.3%) of the remaining frame delay after excluding the propagation delay for ethernet and Wi-Fi, respectively. For 3-retrans frames, this ratio increases to 85.9% (70.0%), causing 46.0% (71.4%) of these frames to experience delays exceeding 100 ms.

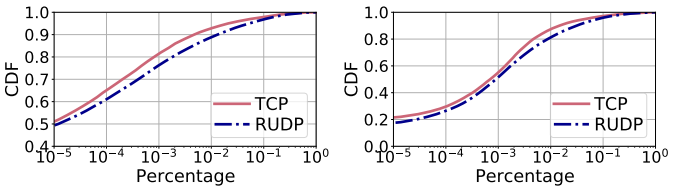
Additionally, we calculate the portion of long-tail delay frames related to packet loss. As shown in Fig.4, more than 66.2% (25.2%) of >100 ms frames are loss-affected frames, of which 53.2% (20.7%) are multi-retrans frames.As a conclusion, packet losses, especially repeated ones, are strongly related to high frame latency.

2.3 Impact of Transport Protocols

For analyzing the source of packet losses, we first present the impact of underlying transport protocol on loss rate and cloud gaming performance by comparing UDP and TCP. Traditionally, livestreaming systems typically prefer UDP over TCP [8, 19] for its flexibility. UDP enables customized loss recovery mechanism like FEC, supports out-of-order delivery and allows dropping lost packets, while all those enhancements on TCP would make it non-compliant to the standards, raising the need of modifying the client-side in-kernel TCP implementation and making various middleboxes to drop packets [28]. However, given that UDP traffic was reported to be rate-limited [29] by the network, would UDP traffic also suffer from higher loss rate compared to TCP?

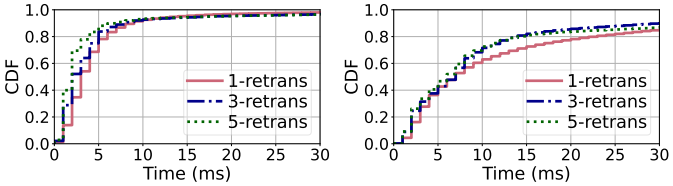
Terminology	Definition
Lossy frame	A frame that contains at least one lost packet (w/o FEC) or A frame not successfully decoded in the first transmission (w/ FEC).
Loss-affected frame	A frame that is either a lossy frame or is fully delivered while previous losses remain unrecovered.
>t ms frame	A frame that was not delivered within a t-milliseconds deadline.
n-retrans frame	A frame that was retransmitted for at least n times.
Loss event	A contiguous period during which every transmitted frame is a lossy, and the frames sent immediately before/after this period are not.
FEC-enabled frame	A frame that was allocated non-zero redundancy rate by the FEC scheme.

Table 1: Frequently-used terminologies in this paper.



(a) Wired. (b) Wi-Fi.

Figure 6: Dist. of lossy frame rate.



(a) Wired. (b) Wi-Fi.

Figure 7: Dist. of end packet queuing time.

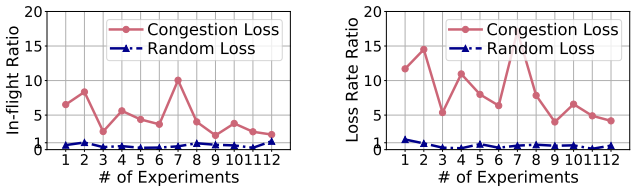


Figure 8: Comparison of loss rate ratio and in-flight ratio under random packet loss and congestion packet loss.

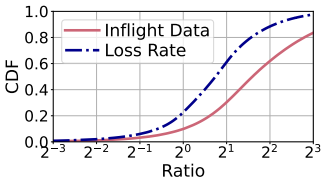


Figure 9: Loss rate and inflight pattern.

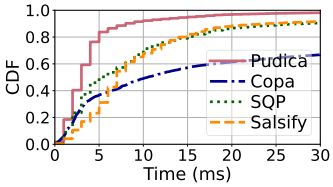


Figure 10: Dist. of end packet queuing time of CC variants.

To examine this, we compare the performance of RUDP against TCP in different types of networks. To achieve fair comparison, we use our default congestion control algorithm, Pudica for both RUDP and TCP, as stated in §2.1. Also, we ensure that both RUDP and TCP generate packets with the same size by establishing one TCP connection per client upon startup to reuse the Maximum Segment Size (MSS) negotiation functionality of TCP, and using the RSS value to limit the size of both TCP and RUDP packets. For users accessing cloud gaming with wired (Wi-Fi) network, the average IP packet loss rate of RUDP cloud gaming sessions is measured to be 0.24% (0.32%), while that of TCP session drops to 0.13% (0.18%).

The distribution of packet loss rate is shown in Fig.5. Combining the two types of users, the loss rate of RUDP is 81.2% higher than TCP. Then, we present the portion of lossy frames in Fig.6. In wired (Wi-Fi) networks, the average portion of lossy frames in RUDP cloud gaming sessions are 0.99% (1.57%), and with TCP it drop to 0.59% (0.68%). In short, UDP and TCP are indeed not treated equally by ISPs. The choice of transport protocol is a major reason of packet loss for both wired and Wi-Fi cloud gaming users.

2.4 Impact of Network Congestion

Secondly, we consider the packet losses caused by overflow of in-network buffers, *i.e.* network congestion. Our measurement data are based on our delay-based congestion control algorithm, Pudica. Although congestion has been known as an important source of packet losses, the frequency of congestion-related losses was unexpected for us, given that we have deployed Pudica, the congestion control algorithm that is very sensitive to increase in packet RTT and tries to avoid any network queuing by lowering its sending rate. In fact, Pudica has achieved the lowest loss rate among four SOTA delay-based CC algorithms suitable for cloud gaming scenarios (Tab.3 in appendix). Therefore, the problem is: would there still be severe congestion even if we already avoided most queuing in the network?

We first compare packet loss rates under two maximum encoding bitrate constraints: 2 Mbps and 50 Mbps. As shown in Fig. 35, the packet loss rate and the portion of lossy frames differ markedly between these two settings. This finding demonstrates that reducing the encoding bitrate can substantially decrease packet loss, suggesting that a significant portion of packet losses are congestion losses. Remarkably, even with the deployment of Pudica, congestion losses remain prevalent. Further investigation shows that most packet losses occur at low queuing delay, as shown in Fig. 7, with the end packet's queuing delay rarely exceeding the frame interval (P80 "retrans $\geq$ 3"[4] of wired < 4ms and P80 "3-retrans" of Wi-Fi < 15ms). This makes congestion detection particularly challenging for delay-based CC algorithms in shallow-buffer environments, exposing a fundamental limitation: their inability to promptly react to congestion in shallow buffers.

To further quantify the prevalence of congestion losses, we performed a detailed statistical analysis of frames with packet loss. Notably, examining the characteristics of isolated frames with packet loss is insufficient to distinguish congestion losses from random losses. Instead, we analyze the root causes of

[4]Frames with "3-retrans" contribute significantly to tail latency.

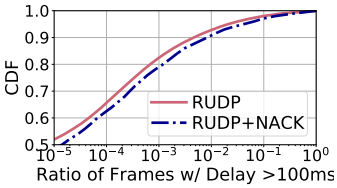


Figure 11: Dist. of >100ms frames, with ACK or NACK.

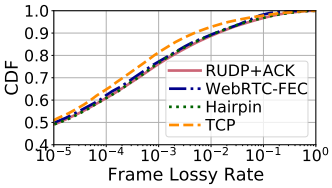


Figure 12: Dist. of frame lossy rate.

each loss event by considering the set of video frames preceding a rate reduction as a whole. Specifically, we investigate the temporal characteristics of these frames under both random and congestion loss scenarios. We hypothesize that, compared to random loss, congestion loss leads to a higher packet loss rate for subsequent frames, as well as a more pronounced accumulation of in-flight retransmitted packets over time. This phenomenon is closely tied to the retransmission patterns and the nature of congestion losses, as detailed in Appendix A.

To empirically examine how loss rate and in-flight retransmissions change, we check the following two ratio metics:

• **Loss Rate Ratio:** We divide the first 4[5] frames in each loss event into two halves. Loss rate ratio is defined as the ratio of the average packet loss rate of the two halves. A value greater than 1 indicates non-random loss.

• **In-Flight Ratio:** Denote F_i as the i-th frame in the loss event and n_i as the number of unrecovered packets in F_i when all non-lost packets in the 4-th frame are SACKed, in-flight ratio $r_{inflight}$ is defined as $(n_1+n_2+n_3+n_4)/s_{avg} * p_{loss}$, where s_{avg} is the average size (number of packets) of the 4 frames and p_{loss} is the loss rate. Given that the RTT is less than the frame interval, if packet losses are random, lost packets would be retransmitted once per frame interval and their number would got reduced exponentially. Consequently, we should have $E(n_i) \leq s_{avg} * p_{loss}^{5-i}$ since we should have performed $4-i$ retransmissions for F_i. Therefore, we have

$$E(r_{inflight}) = \frac{E(n_1+n_2+n_3+n_4)}{s_{avg} * p_{loss}} \leq \sum_{i=1}^{4} p_{loss}^{i-1}$$

In most cases where $p_{loss} \leq 50\%$, we have $E(r_{inflight}) \leq 1.875$ and thus frequent occurrence of larger $r_{inflight}$ value indicates non-random loss.

We conducted experiments in an emulated environment, introducing random packet loss (5% random loss rate) and congestion loss (by limiting the Linux TC buffer size and adding random background traffic). For each scenario, we recorded the two ratio metrics across 12 loss events. As shown in Fig. 8, both ratios are significantly greater than 1 in the case of congestion loss, clearly distinguishing it from random loss. Our large-scale results, summarized in Fig. 9, show that 80% of cases exhibit a loss rate ratio above 1, and 75% of cases exhibit an in-flight ratio above 1.875. These findings provide compelling evidence that the majority of observed packet losses are congestion losses, even when in-network queuing delay has been reduced to near zero by Pudica. This confirms that the bottleneck buffer is as *shallow* as several milliseconds worth of traffic sometimes and congestion in such buffers is still causing heavy packet losses.

Since most packet losses in Pudica are already caused by shallow bottleneck buffers and all other CC algorithms performed worse (Fig. 10) under such circumstances, congestion would be an even more dominant cause of packet losses when using these CC algorithms. However, by correctly avoiding congestion, most packet losses would not occur at all, making loss recovery significantly easier and cheaper. In shallow bottleneck buffers, this goal may be achieved with a loss-based CC algorithm, since they would not incur significant queueing delay and react towards packet losses quickly.

2.5 Impact of Loss Detection Schemes

After studying the source of packet losses, we now check the performance of loss detection schemes, which is often implicitly assumed to be perfect in loss recovery works. Various video transport systems including WebRTC and LiveNet follow a receiver-centric approach to detect packet losses: the receiver detects packet losses according to sequence number gaps or timeout events and send NACK packets to the sender to trigger loss recovery. However, the client-centric loss detection scheme has three major drawbacks: 1) Without unique sequence numbers in retransmitted packets, the client would not be able to detect repeated losses with sequence number gaps, which is quicker than timeout-based loss detection; 2) The client could not determine the send time of the tail packet of each frame before packets from the next frame arrive, causing delay penalty in detecting tail packet losses; 3) NACK losses could only be resolved by sending NACKs more frequently, but this requires the sender to determine whether to perform recovery or not upon receiving a NACK.

Although 1) could be resolved by assigning every packet a unique sequence number (*e.g.* QUIC [30, 31]) and 2) could also be resolved with padding packets, The presence of 3) indicates impossibility to perform proper loss detection without sender-side involvement, which introduces unnecessary system complexity. Therefore, we argue that loss detection should follow a *sender-only* approach. We adopt a TCP SACK [26]-based loss detection scheme. It is able to detect both repeated losses and tail packet losses with RACK-TLP [21]. As shown in Fig.11, simply replacing the loss detection scheme would reduce the portion of >100ms frames by 34.7%. Therefore, the efficiency of loss detection schemes is also essential for performing effective loss recovery.

2.6 Performance of Loss Recovery Schemes

Previously in this section, we have revealed two major sources of packet losses: the choice of transport protocol and shallow bottleneck buffers. Given that FEC-based loss recovery

[5]Pudica reduces its sending rate after observing bandwidth utilization ratio feedback exceeds one, resulting in the first 4 frames in loss events to be encoded at similar bitrate in shallow buffer scenarios.

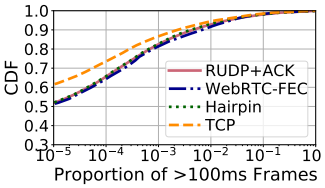


Figure 13: Dist. of > 100ms frames.

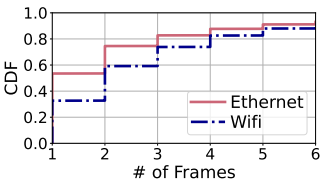


Figure 14: Length of loss events.

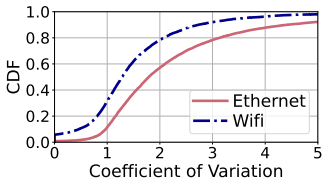


Figure 15: CoV of loss interval.

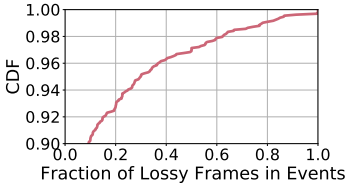


Figure 16: Portion of clustered lossy frames (per session)

schemes could only be used with UDP and are not aware of congestion, it is suspicious whether the performance gain brought by the loss recovery schemes could nullify the loss rate penalty or not. In Fig.12 and Fig.13, we compare the performance of *WebRTC-FEC* and *Hairpin* with *UDP + plain retransmission* and *TCP + plain retransmission* (our default implementation). The result shows that, under 0.187% packet loss rate (measured with UDP + plain retransmission), the loss recovery schemes significantly lowered retransmission rate by 7-41% and reduced the portion of >100 ms frames by 9%-32% comparing with *UDP + plain retransmission*. However, their performance are still behind *TCP + plain retransmission* by at least 32% and 15% for retransmission rate and >100ms frames respectively, confirming that the performance gain of FEC-based loss recovery schemes was completely consumed by the unfriendly network condition.

Furthermore, even if the FEC schemes could be used with TCP and benefit from its low loss rate, we found that designing a proper FEC-based loss recovery scheme is still quite challenging due to the difficulty in predicting packet losses: Firstly, as shown in Fig.35, the average loss rate is lower than 0.1% for 80% of sessions, and the average frame lossy rate is lower than 1% for 90% of sessions. In such sessions, the pattern of packet losses is hard to capture due to the shortage of training data; Secondly, from Fig.14 we can see that loss events are typically short. 82.3% of the loss events lasted less than 3 frame intervals (about 50 ms), indicating that loss prediction schemes must respond to changes in the loss rate very quickly; Meanwhile, the loss events also hardly show periodic pattern. We confirm this by plotting the coefficient of variance of the intervals of loss events in each session in Fig.15: in 80.3% of sessions, the coefficient of variance exceeds 1 (*i.e.* the standard derivation is larger than the average value), indicating that packet losses hardly occur in periodic patterns; Moreover, as discussed in 2.4, packet losses are not tightly coupled with increase in RTT, in fact, 83.1% of packet losses happened when $RTT - RTT_{min}$ is no more than 6 ms.

Therefore, we conclude that an inappropriate choice of transport protocol and CC algorithm could pose a large penalty on the overall performance of the cloud gaming system, and such penalty is hard to eliminate with loss recovery schemes alone.

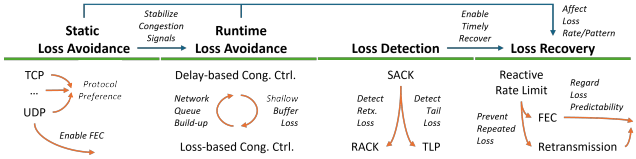


Figure 17: LADR Architecture

2.7 Summary of Key Findings

We summarize the key findings of our measurements and their implications on performance optimization below.

• Packet loss is a major cause of high frame delay, the video streaming system should properly handle the packet loss issue to improve performance (§2.2).

• The network infrastructure has preference on transport protocols. Avoid using ill-treated transport protocols could reduce overall loss rate significantly (§2.3).

• Network congestion is a major source of packet losses even if there is no severe in-network queuing. Loss-based CC algorithms is suitable for avoiding congestion in this case (§2.4).

• Inefficient loss detection schemes pose difficulties in detecting all losses timely. To address this issue, loss of retransmitted packets and tail packets must be properly handled (§2.5).

• The performance gain of loss recovery schemes is limited by packet losses that could be avoided otherwise (§2.6).

3 Design of LADR

Based on the idea that packet losses could be handled better with a systematic approach that not only recover from losses efficiently, but also detects losses timely and avoids most losses in advance, we now present LADR, our solution to the packet loss issue.

3.1 Overall Architecture

We illustrate the architecture of LADR in Fig.17. LADR runs on the sender side and is tightly coupled with the video transport system. Instead of a complete system that runs as a stand-alone executable or a functional module, LADR is a high-level design framework that acts as a suite of requirements for designing a loss-resistant video streaming system. As shown, LADR consists of the following components:

• **Static Loss Avoidance** avoids losses by adapting the video transport system design to the network environment, it mainly affects the data path used for video transport. Regarding the protocol preference of the network infrastructure and the need of using FEC, it requires one customizable data path to support FEC and one data path with preferred protocol. In our implementation, we use co-existed TCP and RUDP connections to meet such requirements.

• **Runtime Loss Avoidance** avoids losses by monitoring the realtime status of the video transport system and adjusting sending rate accordingly. Given that congestion in shallow and deep bottleneck buffers generate different congestion signals, it requires co-existence of a delay-based CC that lowers queuing delay in deep buffers and a loss-based CC that slows down quickly upon losses in shallow buffers. In our implementation, we use the combination of Pudica and a CUBIC-based algorithm designed by us.

• **Loss Detection** ensures that the loss recovery scheme knows about packet losses as early as possible. It requires timely detection of the loss of both retransmitted packets and tail packets. In our implementation, we use the SACK-based loss detection scheme as introduced in §2.1 and §2.5

• **Loss Recovery** takes action after loss events to quickly deliver lost packets and avoid repeated losses. It requires an additional sending rate limit scheme to avoid repeated loss of lost packets, and a proper loss recovery scheme. In our implementation, we design our own reactive rate limit algorithm and perform loss recovery with opportunistically invoked FEC-based loss recovery scheme.

By optimizing towards packet losses in each stage of video transport, LADR is able to handle certain types of packet losses, which could not be easily handled with loss recovery schemes only, and achieve better performance. We also note here that LADR requires the design of the four components to be tightly coupled for the following reasons: 1) Different static loss avoidance schemes result in different loss rate. Reducing losses helps simplifying loss-based CC algoritms and makes congestion signals more reliable; 2) Loss avoidance schemes also affects loss patterns. By filtering out heavy losses, loss avoidance components could lower the cost of loss recovery schemes; 3) Ensuring timely loss detection provides more time for loss recovery, making retransmission more effective. In short, the components relies on each other to ensure the quality of their input and resolving issues that they could not handle. Such a *systematic* design enables each component to focus on specific types of packet losses and addresses them *precisely*. Here we note that LADR is not built on conceptually-new network optimizations approaches. Instead, LADR is a data-driven approach that carefully chooses existing methods to fit the real condition of the WAN, which is commonly considered as a black box. In fact, what makes LADR unique is *how* it combines its building blocks, instead of *what* it uses to form itself.

In the rest of this section, we will discuss the design of the CUBIC-based CC for shallow bottleneck buffers (§3.2), the reactive rate limit scheme for accelerating loss recovery (§3.3) and the opportunistic FEC-based loss recovery scheme (§3.4). The implementation of the video transport system, along with the usage of co-existed TCP/RUDP connections, would be detailed in §4 instead.

3.2 Loss-Based Bitrate Control

As discussed in §2.4, in networks with shallow bottleneck buffers, delay-based CC algorithms are prone to congestion-caused packet losses. Therefore, based on CUBIC [20] the default CC algorithm in Linux kernel, we design a loss-based CC algorithm as a complement of Pudica, our delay-based CC algorithm, to avoid congestion losses when the bottleneck buffer is shallow.

Congestion signal detection. We adopt the basic idea of loss-based CC algorithms that the sending rate should be decreased multiplicatively when packet losses are observed, but instead of slowing down upon each single packet loss, we only decrease the sending rate when all the following conditions are met:

• The average ACK rate of the n frames is less than their average bitrate. The ACK rate is calculated on the time period between when the first and last non-lost packet in the n frames are ACKed. This condition further ensures that congestion happened with the fact that the network is not ACKing packets as quick as we send them.

• In the n consecutive lossy frames, the estimated queuing delay of the first lossy frame (as defined in §2.4) is less than $RTT_{min} + \delta$. We use this condition to differentiate congestion in shallow bottleneck buffers from the other ones, which should otherwise handled by the delay-based CC algorithm.

Optimally, we would like to pick the minimum possible value of n and maximum possible value of δ. In our experiments, however, we found that n need to be at least 3 to smooth out the temporal fluctuations of the ACK rate, and δ need to be less than a frame interval to filter out the congestion that could be detected by Pudica. Here we note that the optimal value of δ depends on the delay-based CC in use and changing it requires δ to be tuned again. We also note that in our second condition, the corner case where all packets in the n frames are lost, we set the ACK rate to 0, which ensures that this condition is met.

Congestion Avoidance. We adopt the core idea of CUBIC that: 1) congestion losses happened because we reached an over-aggressive sending rate; 2) if we send slower than the ACK rate when congestion happens, we will be safe since the network is capable of delivering such amount of traffic; 3) as we start from a safe sending rate and approach the over-aggressive rate again, it will be increasingly likely that congestion losses would happen, so we should slow down as we approach the over-aggressive rate.

Therefore, similar to CUBIC, when we detect a congestion signal, we record the sending rate when we sent the packets that caused congestion as the over-aggressive rate, denoted as B_{agg}. Meanwhile, we record $B_{safe} := (1-\beta) \times R_{ack}$ as the safe rate, where R_{ack} is the ACK rate, and β is a CUBIC parameter that defaults to 0.8.

Then, we adjust our target rate B_l according to the following cubic function:

$$B_l(t) = C(t-K)^3 + B_{max}$$

Where t is the time elapsed since we detected the congestion signal, C is a CUBIC parameter that affects the time needed (K) for B_l to reach B_{max}. We empirically set it to 1. K is computed with the formula from original CUBIC:

$$K = \sqrt[3]{\frac{B_{max} - (1-\beta) \times B_{ack}}{C}}$$

Co-existence of the loss-based CC and Pudica. At runtime, we keep both Pudica and our loss-based CC running. Assuming B_d is the bitrate of the delay-based CC and B_l is the bitrate of the loss-based CC, the final target encoding rate is $\min(B_l, B_d)$. Our rationale here is: 1) when loss-based congestion signals are detected, the loss-based CC would be properly initialized so that the sending rate would be limited by the loss-based CC; 2) when the shallow buffer is no longer the network bottleneck, we will no longer observe loss-based congestion signals before causing severe queuing. In this case B_l would quickly exceed B_d, allowing Pudica to take the control.

3.3 Reactive Rate Limit

Different from the traditional idea that loss recovery is focused on preventing lost packets from delaying delivery of user traffic, LADR further introduces an additional sending rate limit scheme as a part of the loss recovery scheme. The rationale here is, congestion avoidance schemes are not designed to avoid *all* congestion since congestion could not be detected at 100% precision. They must properly balance the risk of wasting available bandwidth and allowing congestion to happen. This is to say, the CC algorithm *must* be sometimes inevitably over-aggressive. Therefore, with packet losses already happened, we have the potentiality to reconsider this trade-off by showing our preference towards avoiding further congestion, and this is when the reactive rate limit scheme takes action. For the case of cloud gaming, we design our reactive rate limit scheme as follows:

Applicable conditions. For all cloud gaming sessions, we first determine if reactive rate limit is applicable to the session by checking if any retransmitted packets were lost. If no retransmitted packets were lost, we choose to disable reactive rate limit and trust the ability of the congestion avoidance scheme instead. For sessions with lost retransmitted packets, we continuously record n_{rec}, the number of packets to be recovered, and enable reactive rate limit when n_{rec} reaches n_{high}. At the same time, we mark all unsent data in the TCP/RUDP buffer as governed by reactive rate limit. When n_{rec} becomes less than n_{low} and all data governed by reactive rate limit was sent, we disable reactive rate limit. In practice, to ensure that reactive rate limit only responses to severe packet loss events that require urgent recovery, we empirically set n_{high} and n_{low} to 4 and 2, respectively.

Pacing rate limit. When reactive rate limit is enabled, we limit the pacing rate to the ACK rate. Normally, CC algorithms rely on pacing at higher rates to probe the bandwidth. Reactive pacing limit temporarily disables such ability to avoid further congestion. Moreover, our pacing rate limit is applied to both new packets and retransmitted packets. The ACK rate is calculated as follows: 1) if any frames sent $RTT_{min} + t_{intv}$ ago are still unACKed, we judge the network as congested. Here t_{intv} is the frame interval (*e.g.* 16.67ms for 60 fps). In this case, we backtrack to the first frame F_c whose delay is at least $RTT_{min} + t_{intv}$ and estimate the ACK rate during current congestion with the ACK rate since $t_{F_c} + RTT_{min}$, where t_{F_c} is the time that F_c was sent; 2) otherwise, we use the average ACK rate of 6 recently-ACKed frames.

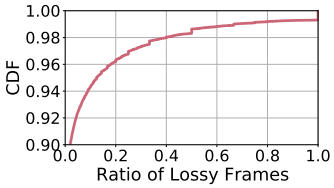


Figure 18: Dist. of frame lossy rate for frames w/ RTT inflation (per session).

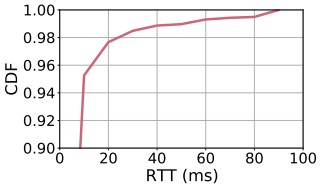


Figure 19: Dist. of the >80% clustering center of end queueing delay of lossy frames (per session).

Pending data limit. When performing loss recovery, the pacing rate would be shared between new packets and retransmitted packets. To ensure quick delivery of retransmitted packets, we prioritize them whenever retransmission is required, resulting in local queuing delay of pending frames. To avoid such delay, we also limit the amount of unsent traffic. Every t_{intv}, we compute t_{pend}, the time needed to send all pending packets with current pacing rate. if $t_{pend} > 2t_{intv}$, we avoid generating new traffic by requiring the video encoder to skip the next frame (detailed in §6.1).

3.4 Opportunistic FEC-based Loss Recovery

After avoiding most packet losses and detecting the remaining ones in time, we now present the design of our loss recovery scheme. As discussed in §2.6, predicting packet losses is challenging due to the lack of training data, fluctuations of loss rate and unclear loss patterns. In our measurements, WebRTC-flavored FEC scheme is able to generate enough redundant packets for recovering 99.1% of FEC-enabled lossy frames. But, unsurprisingly, only 48.4% of lossy frames were FEC-enabled, resulting in a 48% *overall* recover rate.

Although loss prediction in general is difficult, by looking into per-session results, we found that there exist two types of loss patterns that are quite fit for FEC-based loss recovery: 1) As shown in Fig.16, in ~3% of sessions, packet losses are clustered in long loss events that lasted for at least 5 frame intervals. In such sessions, we could achieve fairly high recovery rate by simply enabling FEC for subsequent frames after detecting losses; 2) As demonstrated in Fig.4, high queuing delays contribute to high-delay frames. Even with most congestion avoided, we notice that in about 2% of

sessions, packet losses are typically detected when queuing delay reaches a certain threshold. Fig.18 illustrates that, in these sessions, >40% of high-delay frames are lossy. Furthermore, when packet loss occurs in these frames, the RTT exhibits extremely strong clustering characteristics (see Fig.19). This is possibly caused by aggressive background senders that filled the bottleneck buffer and causing our packet to drop even if we are already sending at the lowest possible video bitrate. In such sessions, due to severe queuing, retransmissions have no chance of recovering packets within the delay deadline. Therefore, applying FEC as queue builds up would help deliver packets that would otherwise definitely miss the delay deadline. It is worth noting that over all frames, only 0.2% of them suffer from high queuing delays, which means the bandwidth overhead for applying FEC to them is always low.

Therefore, we design our opportunistic FEC-based loss recovery scheme as follows:

Applicable Conditions. We run the following two session-monitoring procedures to determine if FEC should be enabled:
• *Procedure #1:* We maintain two counters, sum_{long} and sum_{short}, where sum_{long} is the accumulated number of packets lost in *long* loss events with length ≥ 5 frame intervals and sum_{short} is the accumulated number of other lost packets. After each loss event, we add the number of lost packets n_{lost} to the corresponding counter. Anytime we detect a packet loss, we enable FEC instantly if $sum_{long} > k \times sum_{short}$, where k is a scaling factor. Then, we check for packet losses and disable FEC if packet losses did not happen in recent t_{hold} ms.
• *Procedure #2:* We monitor if the frames with $>t_q$ ms queuing delay are lossy frames. If the proportion of lossy frames in frames with $>t_q$ ms queuing delay is at least 5%, we enable FEC whenever queuing delay reaches t_q ms. Then, we keep monitoring the queuing delay and disable FEC if the queuing delay stayed under t_q ms for t_{hold} ms.

We empirically set k, t_{hold} and t_q to 1, 400 and 50, respectively. The values are chosen to balance the amount of lossy frames that the FEC scheme may recover and the non-lossy frames with FEC enabled. In §C we present a separated study about the value of the parameters. Here we note that optimally, in procedure #2 we should detect the delay threshold for losses to happen for each session at runtime. However, given that heavy queuing are rare even in those sessions, learning the threshold at runtime would cost too much time. Therefore, we empirically set t_q to a "safe" threshold that works for most sessions.

FEC Scheme. Given that WebRTC-flavored FEC scheme recovers FEC-enabled lossy frames quite effectively, when FEC is enabled, we simply apply it to all frames.

4 Video Transport System Implementation

In this section, we present the implementation of LADR, which integrates multi-protocol, multi-connection strategies, adaptive rate control, and intelligent frame distribution. The following sections detail the core components of the system.

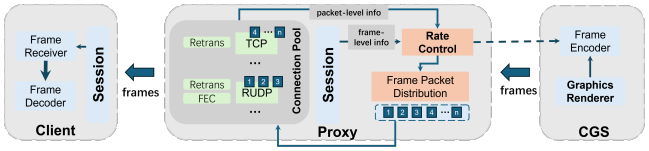


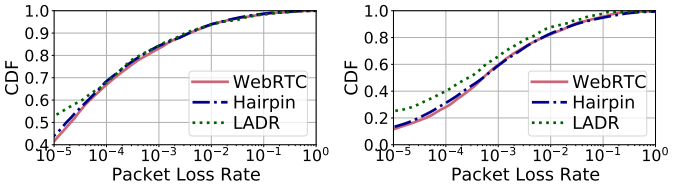
Figure 20: Video transport system implementation

Multi-connection architecture: The system employs a multi-protocol, multi-connection architecture between the client and the proxy, enabling each user session to establish multiple TCP and RUDP connections concurrently. This design differentiates the system from conventional multi-protocol, single-connection approaches [32]. All connections are centrally managed within a unified connection pool, allowing the system to select the most appropriate connection for data transmission based on real-time network and frame characteristics. Importantly, only RUDP connections support the activation of FEC mechanisms. Due to the apparent loss rate difference between TCP and UDP, TCP is prioritized as the default.. The decision to enable or disable FEC is made dynamically through continuous monitoring of packet loss patterns, thereby enhancing transmission reliability under varying network conditions.

The reason to establish multiple TCP connections is that the reliable nature of TCP connections forbids us from abandoning unACKed traffic. However, in cloud gaming, severe network disruptions would cause multiple in-flight video frames to be outdated. Normally, to reduce the video bitrate, most of our video frames are P frames that could only be decoded sequentially and thus we must deliver all old frames before decoding new frames. However, with a large amount of packets to be retransmitted, sending a new I frame at low bitrate and abandoning all undelivered frames would be a better option for the cloud gaming session to quickly recover from network failures. By maintaining multiple TCP connections, we could easily abandon one TCP connection and switch to another to support such functionality.

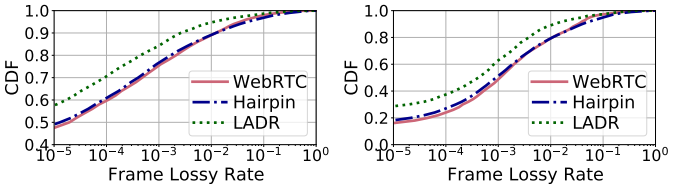
Rate control: To reduce control loop latency, rate control is implemented at the proxy, which functions as the central module for adaptive bitrate management. It aggregates both frame-level statistics and packet-level metrics to accurately assess network capacity and dynamically adjust video encoding bitrate. Bitrate adjustment decisions are sent in real time to the cloud gaming server (CGS), ensuring smooth and efficient end-to-end cloud gaming delivery.

Frame distribution among multiple connections: The frame distribution module utilizes an intelligent scheduling algorithm that incorporates both the network information provided by the rate controller and real-time packet-level frame data. It continuously monitors the latency and packet loss patterns of each connection, dynamically selecting the optimal combination of connections for transmitting each video frame.



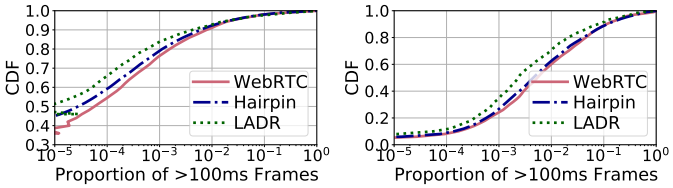
(a) Wired. (b) Wi-Fi.

Figure 21: Dist. of packet loss rate.



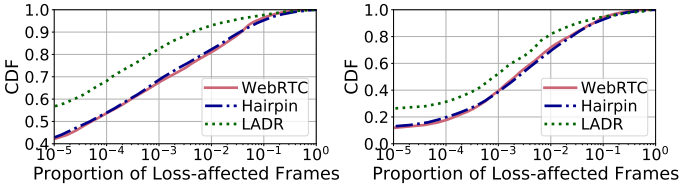
(a) Wired. (b) Wi-Fi.

Figure 22: Dist. of frame lossy rate.



(a) Wired. (b) Wi-Fi.

Figure 23: Dist. of >100ms frames.



(a) Wired. (b) Wi-Fi.

Figure 24: Dist. of percentage of frames affected by losses.

This strategy not only enables parallel transmission across multiple connections but also facilitates proactive switching to healthier connections, thereby maintaining robust end-to-end performance and reliability.

5 Deployment Performance

Our measurements were conducted on *Tencent START* cloud gaming platform utilizing edge computing services, where the base RTT for 50% and 90% of gaming sessions is below 10 ms and 20 ms, respectively, for both wired and Wi-Fi networks. To meet the user demands of consistently low latency and high bitrate, we also deployed Pudica, a congestion control algorithm designed to achieve near-zero queuing delay for cloud gaming. Our deployment accumulated over 400,000 total gameplay hours. After filtering out incomplete logs and test traffic, the final evaluation dataset spanning March 2024 to April 2025 comprises ∼324,000 sessions, each assigned to one of the proposed schemes, with a total of ∼200,000 valid gameplay hours. Notably, cloud gaming evaluation prioritizes tail cases, which are the primary sources of user complaints, over overall metric distributions. Since network performance is near-ideal in most scenarios, our analysis focuses exclusively on the tail distribution of evaluated metrics.

5.1 Overall Performance of LADR

We first present the performance of LADR by comparing it against the baseline video transport systems. As shown in Fig.23, only 0.115%/0.031% of the frames missed the 100 ms/200 ms deadline, respectively. Compared to WebRTC and Hairpin, LADR reduced the portion of high-latency frames (>100ms/>200ms) by 60%/28% and 55%/36%, respectively. Such a significant improvement is largely related with the reduction in packet loss rate, and subsequently, the need of performing loss recovery. We plot the CDF of per-session packet loss rate in Fig.21. As shown, LADR suffers from only 0.049% (0.320%) average (90% percentile) packet loss rate, which is 20% (18%) lower than the UDP-based baselines. Under such a low loss rate, LADR only need to perform loss recovery for 0.151% of frames, while WebRTC and Hairpin performed loss recovery for 0.518% and 1.021% of frames, respectively. The distribution of per-session portion of lossy frames is shown in Fig.22.

To demonstrate the cost of loss recovery, we plot the CDF of *frame recovery time* of all frames in Fig.29. The recovery time of frame F is defined as the time spent on recovering lost packets in F, and is computed as $t_F - RTT_{P_F}$, where t_F is the frame delay, RTT_{P_F} is the RTT of the last non-lost packet P_F in F as defined in §2.4.

Compared to WebRTC and Hairpin, LADR reduced the portion of frames with high recovery time (>50ms/>100ms) by 74.1%/63.2% and 81.7%/75.5%, respectively. Moreover, considering the fact that frames are transmitted with reliable transport protocols in cloud gaming, packet losses that severely delayed a frame would also delay subsequent frames due to head-of-line blocking. We denote a frame as a *loss-affected frame* if: 1) this frame is a lossy frame or 2) by the time that all packets in this frame were SACKed, at least one packet sent before this frame was not delivered yet. Fig.24 shows the distribution of per-session portion of loss-affected frames. Compared to WebRTC (Hairpin), LADR reduced the portion of loss-affected frames by 73% (87%), respectively.

5.2 Performance Gain Breakdown

To illustrate the gain of each individual optimization, we start from the default implementation (see §2.1) of our video transport system, and enable our optimizations one by one. The impact of each individual optimization on the five key performance metrics, namely, packet loss rate, the proportion of lossy frames, the proportion of loss-affected frames, the proportion of >100 ms frames and recovery delay, is plotted in Fig.25, Fig.26, Fig.27, Fig.28, Fig.30, respectively. We also summarize the improvement of each optimization on the performance metrics in Tab.2.

Loss-based bitrate control. By switching between loss-based CC algorithm and delay-based CC algorithm to avoid

Scheme	Packet loss (%)	Lossy frame (%)	Loss-affected frame (%)	>100 ms frame (%)	Avg frame delay (ms)
Default	0.122/0.178	0.519/0.477	0.823/0.724	0.192/0.510	20.61/37.24
LBBC	0.062/0.142	0.485/0.372	0.658/0.641	0.185/0.470	20.21/37.23
LBBC+RRL	0.039/0.120	0.327/0.316	0.452/0.495	0.143/0.452	20.08/35.81
LADR	0.049/0.320	0.151/0.260	0.233/0.412	0.115/0.426	19.97/34.61

Table 2: LADR Performance Improvement Breakdown (Average value / 90% percentile).
Default: default implentation; LBBC: Loss-based bitrate control; RRL: reactive rate limit.

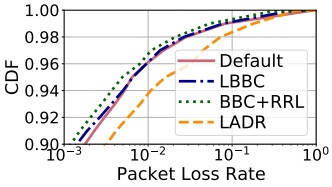


Figure 25: Dist. of packet loss rate (breakdown).

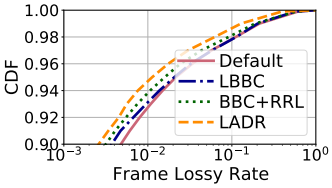


Figure 26: Dist. of frame lossy rate (breakdown).

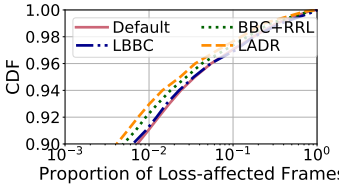


Figure 27: Dist. of loss-affected frames (breakdown).

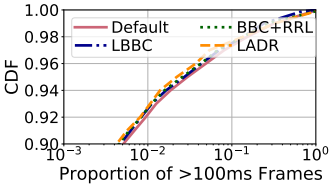


Figure 28: Dist. of percentage of > 100ms (breakdown).

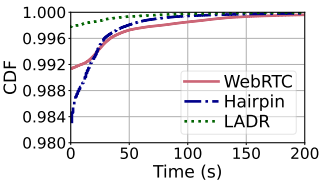


Figure 29: Dist. of loss recovery time.

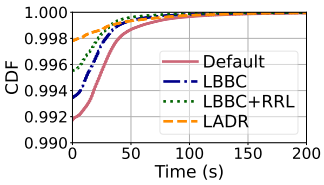


Figure 30: Dist. of loss recovery time (breakdown).

congestion in both shallow and deep bottleneck buffer, loss-based bitrate control is specialized in reducing congestion losses. Although the proportion of lossy frames is reduced by only 6.6%, LBBC significantly reduced loss rate by 49.2%, indicating that LBBC is especially useful in avoiding congestion that may cause heavy packet losses.

Reactive rate limit. Being enabled only upon severe packet losses, reactive rate limit enhances cloud gaming performance mainly by shortening loss events and avoiding queuing and loss of retransmitted packets. As a result, reactive rate limit significantly reduced the proportion of lossy frames by 32.6%, and achieved 22.7% >100 ms frame reduction, which is the best among the three design points.

Opportunistic FEC-based loss recovery. By enabling UDP-based FEC scheme under certain circumstances, opportunistic FEC-based loss recovery trades minor overall loss rate for better chance of delivering lost packets in fewer retransmissions. As shown, by opportunistically using UDP, opportunistic FEC-based loss recovery raises packet loss rate by 25%. However, due to its capability of resolving losses in highly lossy sessions, both lossy frames and loss-affected frames are reduced by over 50%, resulting in 19% less >100 ms frames. For most sessions whose average loss rate is below 0.1%, FEC will cause the loss rate to increase slightly. But for sessions at the tail, adopting FEC significantly reduces retransmission traffic (e.g. reduce loss rate from 77.23% to 55.9% at 99.9% percentile), hence the average gets lower. Here we note that in §3.4 we measure the recover rate of the WebRTC-flavored FEC scheme to be only 48%. However, despite the fact that: 1) LADR suffers from higher loss rate compared to LBBC+RRL, which is a TCP-only solution; 2) LADR does not enable FEC for all frames; 3) LADR also uses the WebRTC-flavored FEC scheme, LADR achieved a better recover rate. Such result indicates that loss avoidance not only reduces loss rate, but also produces more predictable loss patterns, facilitating the design process of FEC-based loss recovery schemes. The reduced loss rate further contributes to lower overhead. Specifically, LADR achieves a FEC overhead of only 0.6% (2.4%) for wired and Wi-Fi scenarios, respectively, which is substantially lower than WebRTC-FEC (21.7% and 33.5%) and Hairpin (3.2% and 5.3%) under the same conditions.

6 Discussion

Machine learning-based loss prediction. Although we presented that packet losses could not be accurately predicted with simple methods because of the shortage of training data, the fluctuating loss rate and random loss pattern. It is possible that sophisticated machine learning-based methods would capture the high-level feature of the network condition and perform accurate loss prediction. However, to reduce deployment cost, currently our video proxies at network edge are not equipped with necessary hardwares to run the resource-hungry machine learning methods. We leave the design of machine learning-based loss prediction as our future work.

Integrating advanced loss recovery schemes. In this paper we have covered loss recovery via retransmission and applying FEC to packets sent for the first time. However, by adopting advanced loss recovery schemes such as applying FEC to retransmitted packets and picking redundancy rate accordingly [6, 33], we may further improve the performance of loss recovery. We note that LADR framework allows easy integration of other loss recovery schemes and we leave this as future work.

6.1 Lessons learned

To enable scalable deployment of *Tencent START* cloud gaming platform, several critical challenges must be addressed,

including limited cross-layer network visibility, rigid retransmission mechanisms, frame synchronization delays due to sender-side buffer accumulation, and performance degradation stemming from nested congestion control. To overcome these limitations, the system incorporates the following key optimizations.

Cross-layer information interaction. To enhance the application layer's network awareness and adaptability, the system establishes a cross-layer communication interface between the protocol stack and the application layer. The application layer obtains real-time metrics such as per-packet RTT and packet loss rate, enabling fine-grained flow control and retransmission strategies at the application layer.

Hierarchical retransmission. To enable flexible retransmission, the system implements a hierarchical strategy. At the connection level, transport protocols (e.g., TCP, reliable UDP) provide packet-level reliability. At the session level, the application layer maintains frame and packet status, enabling selective retransmission of lost fragments to other connections. When a particular connection experiences degradation, lost packets can be retransmitted via higher-quality connections.

Adaptive frame synchronization. To address the issue of frame timeout caused by buffer accumulation at the sender, the system employs an adaptive frame synchronization strategy. When the proxy buffer experiences moderate accumulation, the proxy notifies the CGS to skip frames, thereby reducing latency and preventing further congestion. If the buffer accumulation becomes severe, expired frames are proactively discarded, and a new key frame is requested. The key frame is transmitted via an idle connection selected from a pool of concurrent links to avoid head-of-line blocking. This mechanism ensures timely receiver-side frame synchronization and balances reliability with real-time frame delivery.

Nested congestion control. At transport layer, the TCP/RUDP connections also have their own CC module, which may result in nested congestion control. Specifically, when the sending rate of TCP/RUDP connections becomes lower than the rate determined by the rate controller, substantial packet queuing occurs within the proxy's transport-layer buffer. To alleviate this issue, the rate controller proactively synchronizes its rate with the TCP/RUDP connections in real time.

7 Related Work

Low latency congestion control. Many congestion control algorithms [22, 34–38] have been designed to reduce end-to-end latency while maximizing bandwidth utilization. In recent years, CC algrithms for real-time communication (RTC) [39–45] are proposed to support interactive video streaming such as video conferencing. However, they still cannot meet the stringent latency requirements of cloud gaming. Pudica [7], SQP [24], and Salsify [23] are designed specifically for cloud gaming and achieved notable success in reducing latency. Nevertheless, as noted in §2.4, they still face challenges with congestion in shallow buffers, leading to heavy packet loss and long tail delay. Our work LADR addresses this by detecting congestion in shallow buffer and using loss-based bitrate control to further reduce loss recovery time and end-to-end latency.

Accelerating retransmission. Previous work has been working on optimizing retransmission mechanism [12, 46, 47], improving loss detection schemes [21, 26, 48], or manipulating sending patterns to reduce packet losses [49, 50]. There are also many works (Hairpin [6] and also [51–54] jointly optimizating redundancy and retransmission to improve the tail performance for interactive video streaming. Different from them, we find network congestions are the major cause of packet losses in cloud gaming system, as illustrated in §2.4. LADR prevents repeated loss by performing rate limit to avoid further congestions.

FEC-based loss recovery. FEC has been widely used to recover lost packets for multimedia [12, 13] or real-time video communications [10, 11] and implemented in WebRTC [6, 14, 15] and QUIC [16–18]. However, conventional FEC schemes face significant challenges in accurately predicting packet loss patterns as measured in §2.6. This not only increases bandwidth overhead but also latency when FEC recovery fails. By selectively enabling FEC only in suitable scenarios, LADR significantly improves FEC efficiency and reduces latency.

8 Conclusion

We performed an in-depth study of the packet loss issue in large-scale cloud gaming by quantifying the relationship between packet losses and frame delay, and measuring the impact of transport protocol, network congestion, loss detection and loss recovery on packet losses. Differently from the common sense that FEC-based loss recovery is an elixir of all packet losses, we revealed the necessity of proper loss avoidance and loss detection schemes in tackling packet losses, and more importantly, the necessity of co-operated loss avoidance, loss detection and loss recovery schemes for them to limit the problem space and improve input quality of each other. Through the design and large-scale deployment of LADR, we demonstrate the effectiveness of such a systematic solution of the packet loss issue in cloud gaming and provide generalized guideline for other applications to follow our design principle and minimize the negative impact of packet losses.

References

[1] Google. Google stadia. https://stadia.google.com/.

[2] NVIDIA. Your games. your devices. play anywhere | nvidia geforce now. https://www.nvidia.com/en-us/geforce-now/.

[3] Microsft. Xbox cloud gaming. https://www.xbox.com/.

[4] Amazon. Amazon luna. https://luna.amazon.com/.

[5] Tencent. Tencent pioneer. https://gamer.qq.com/.

[6] 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 *NSDI*, pages 907–926, 2024.

[7] Shibo Wang, Shusen Yang, Xiao Kong, Chenglei Wu, Longwei Jiang, Chenren Xu, Cong Zhao, Xuesong Yang, Jianjun Xiao, Xin Liu, et al. Pudica: Toward Near-Zero queuing delay in congestion control for cloud gaming. In *NSDI*, pages 113–129, 2024.

[8] Google. Webrtc. https://webrtc.org/.

[9] Mo Zanaty, Varun Singh, Ali C. Begen, and Giridhar Mandyam. RTP Payload Format for Flexible Forward Error Correction (FEC). RFC 8627, July 2019.

[10] Sheng Cheng, Han Hu, Xinggong Zhang, and Zongming Guo. Deeprs: Deep-learning based network-adaptive fec for real-time video communications. In *2020 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–5. IEEE, 2020.

[11] 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 *Proceedings of the 13th ACM Multimedia Systems Conference*, pages 96–108, 2022.

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

[13] Marcin Nagy, Varun Singh, Jörg Ott, and Lars Eggert. Congestion control using fec for conversational multimedia communication. In *Proceedings of the 5th ACM Multimedia Systems Conference*, pages 191–202, 2014.

[14] Michael Rudow, Francis Y Yan, Abhishek Kumar, Ganesh Ananthanarayanan, Martin Ellis, and KV Rashmi. Tambur: Efficient loss recovery for videoconferencing via streaming codes. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*, pages 953–971, 2023.

[15] Congkai An, Huanhuan Zhang, Shibo Wang, Jingyang Kang, Anfu Zhou, Liang Liu, Huadong Ma, Zili Meng, Delei Ma, Yusheng Dong, et al. Tooth: Toward optimal balance of video QoE and redundancy cost by fine-grained fec in cloud gaming streaming.

[16] François Michel, Quentin De Coninck, and Olivier Bonaventure. Quic-fec: Bringing the benefits of forward erasure correction to quic. In *2019 IFIP Networking Conference (IFIP Networking)*, pages 1–9. IEEE, 2019.

[17] Pablo Garrido, Isabel Sanchez, Simone Ferlin, Ramon Aguero, and Ozgu Alay. rquic: Integrating fec with quic for robust wireless communications. In *2019 IEEE Global Communications Conference (GLOBECOM)*, pages 1–7. IEEE, 2019.

[18] François Michel and Olivier Bonaventure. Quirl: Flexible quic loss recovery for low latency applications. *IEEE/ACM Transactions on Networking*, 2024.

[19] Jinyang Li, Zhenyu Li, Ri Lu, Kai Xiao, Songlin Li, Jufeng Chen, Jingyu Yang, Chunli Zong, Aiyun Chen, Qinghua Wu, et al. Livenet: a low-latency video transport network for large-scale live streaming. In *Proceedings of the ACM SIGCOMM 2022 Conference*, pages 812–825, 2022.

[20] Sangtae Ha, Injong Rhee, and Lisong Xu. Cubic: a new TCP-friendly high-speed TCP variant. *ACM SIGOPS operating systems review*, 42(5):64–74, 2008.

[21] Yuchung Cheng, Neal Cardwell, Nandita Dukkipati, and Priyaranjan Jha. The RACK-TLP Loss Detection Algorithm for TCP. RFC 8985, February 2021.

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

[23] Sadjad Fouladi, John Emmons, Emre Orbay, Catherine Wu, Riad S Wahby, and Keith Winstein. Salsify: Low-latency network video through tighter integration between a video codec and a transport protocol. In *NSDI*, pages 267–282, 2018.

[24] Devdeep Ray, Connor Smith, Teng Wei, David Chu, and Srinivasan Seshan. SQP: Congestion control for low-latency interactive video streaming. *arXiv preprint arXiv:2207.11857*, 2022.

[25] Carsten Burmeister, Jose Rey, Noriyuki Sato, Joerg Ott, and Stephan Wenger. Extended RTP Profile for Real-time Transport Control Protocol (RTCP)-Based Feedback (RTP/AVPF). RFC 4585, July 2006.

[26] Sally Floyd, Jamshid Mahdavi, Matt Mathis, and Dr. Allyn Romanow. TCP Selective Acknowledgment Options. RFC 2018, October 1996.

[27] Irving S Reed and Gustave Solomon. Polynomial codes over certain finite fields. *Journal of the society for industrial and applied mathematics*, 8(2):300–304, 1960.

[28] Costin Raiciu, Christoph Paasch, Sebastien Barre, Alan Ford, Michio Honda, Fabien Duchene, Olivier Bonaventure, and Mark Handley. How hard can it be? Designing and implementing a deployable multipath TCP. In *9th USENIX symposium on networked systems design and implementation (NSDI 12)*, pages 399–412, 2012.

[29] Yueyang Pan, Ruihan Li, and Chenren Xu. The first 5g-lte comparative study in extreme mobility. *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, 6(1):1–22, 2022.

[30] Adam Langley, Alistair Riddoch, Alyssa Wilk, Antonio Vicente, Charles Krasic, Dan Zhang, Fan Yang, Fedor Kouranov, Ian Swett, Janardhan Iyengar, et al. The quic transport protocol: Design and Internet-scale deployment. In *Proceedings of the conference of the ACM special interest group on data communication*, pages 183–196, 2017.

[31] Jana Iyengar and Martin Thomson. QUIC: A UDP-Based Multiplexed and Secure Transport. RFC 9000, May 2021.

[32] Jia Zhang, Shaorui Ren, Enhuan Dong, Zili Meng, Yuan Yang, Mingwei Xu, Sijie Yang, Miao Zhang, and Yang Yue. Reducing mobile web latency through adaptively selecting transport protocol. *IEEE/ACM Transactions on Networking*, 31(5):2162–2177, 2023.

[33] Yunzhe Ni, Zhilong Zheng, Xianshang Lin, Fengyu Gao, Xuan Zeng, Yirui Liu, Tao Xu, Hua Wang, Zhidong Zhang, Senlang Du, et al. Cellfusion: Multipath vehicle-to-cloud video streaming with network coding in the wild. In *Proceedings of the ACM SIGCOMM 2023 Conference*, pages 668–683, 2023.

[34] Lawrence S Brakmo, Sean W O'Malley, and Larry L Peterson. TCP vegas: New techniques for congestion detection and avoidance. In *SIGCOMM*, pages 24–35, 1994.

[35] Keith Winstein, Anirudh Sivaraman, and Hari Balakrishnan. Stochastic forecasts achieve high throughput and low delay over cellular networks. In *NSDI*, pages 459–471, 2013.

[36] Radhika Mittal, Vinh The Lam, Nandita Dukkipati, Emily Blem, Hassan Wassel, Monia Ghobadi, Amin Vahdat, Yaogong Wang, David Wetherall, and David Zats. Timely: Rtt-based congestion control for the datacenter. *ACM SIGCOMM Computer Communication Review*, 45(4):537–550, 2015.

[37] Mario Hock, Felix Neumeister, Martina Zitterbart, and Roland Bless. TCP lola: Congestion control for low latencies and high throughput. In *2017 IEEE 42nd Conference on Local Computer Networks (LCN)*, pages 215–218. IEEE, 2017.

[38] Gautam Kumar, Nandita Dukkipati, Keon Jang, Hassan MG Wassel, Xian Wu, Behnam Montazeri, Yaogong Wang, Kevin Springborn, Christopher Alfeld, Michael Ryan, et al. Swift: Delay is simple and effective for congestion control in the datacenter. In *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication*, pages 514–528, 2020.

[39] Gaetano Carlucci, Luca De Cicco, Stefan Holmer, and Saverio Mascolo. Congestion control for web real-time communication. *IEEE/ACM Trans. on Networking*, 25(5):2629–2642, 2017.

[40] 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.

[41] 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 1–14, 2020.

[42] Huanhuan Zhang, Anfu Zhou, Yuhan Hu, Chaoyue Li, Guangping Wang, Xinyu Zhang, Huadong Ma, Leilei Wu, Aiyun Chen, and Changhui Wu. Loki: improving long tail performance of learning-based real-time video adaptation by fusing rule-based models. In *MOBICOM*, pages 775–788, 2021.

[43] Xiaoqing Zhu and Rong Pan. Nada: A unified congestion control scheme for low-latency interactive video. In *International Packet Video Workshop*, pages 1–8. IEEE, 2013.

[44] 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, 2022.

[45] Shinik Park, Jinsung Lee, Junseon Kim, Jihoon Lee, Sangtae Ha, and Kyunghan Lee. Exll: An extremely low-latency congestion control for mobile cellular networks. In *CoNEXT*, pages 307–319, 2018.

[46] Ethan Blanton, Vern Paxson, and Mark Allman. TCP Congestion Control. RFC 5681, Internet Engineering Task Force (IETF), 2009.

[47] Pasi Sarolahti, Markku Kojo, and Kimmo Raatikainen. F-RTO: an enhanced recovery algorithm for TCP retransmission timeouts. *ACM SIGCOMM Computer Communication Review*, 33(2):51–63, 2003.

[48] Nahur Fonseca and Mark Crovella. Bayesian packet loss detection for TCP. In *Proceedings IEEE 24th Annual Joint Conference of the IEEE Computer and Communications Societies.*, volume 3, pages 1826–1837. IEEE, 2005.

[49] Saad Biaz and Nitin H Vaidya. "De-Randomizing" congestion losses to improve TCP performance over wired-wireless networks. *IEEE/ACM Transactions on networking*, 13(3):596–608, 2005.

[50] Tobias Flach, Pavlos Papageorge, Andreas Terzis, Luis Pedrosa, Yuchung Cheng, Tayeb Karim, Ethan Katz-Bassett, and Ramesh Govindan. An Internet-wide analysis of traffic policing. In *Proceedings of the 2016 ACM SIGCOMM Conference*, pages 468–482, 2016.

[51] Luca Baldantoni, Henrik Lundqvist, and Gunnar Karlsson. Adaptive end-to-end fec for improving TCP performance over wireless links. In *2004 IEEE International Conference on Communications (IEEE Cat. No. 04CH37577)*, volume 7, pages 4023–4027. IEEE, 2004.

[52] Fan Zhai, Yiftach Eisenberg, Thrasyvoulos N Pappas, Randall Berry, and Aggelos K Katsaggelos. Rate-distortion optimized hybrid error control for real-time packetized video transmission. *IEEE Transactions on Image Processing*, 15(1):40–53, 2005.

[53] Mikhal Shemer. Commit - video coding robustness: Updating hybrid mode's settings. https://webrtc.googlesource.com/src/+/ae7a0522c59d932d72f3d3377c38bebab7ab2b31%5E%21/, 2011. Google Git.

[54] Tobias Flach, Nandita Dukkipati, Andreas Terzis, Barath Raghavan, Neal Cardwell, Yuchung Cheng, Ankur Jain, Shuai Hao, Ethan Katz-Bassett, and Ramesh Govindan. Reducing web latency: the virtue of gentle aggression. In *Proceedings of the ACM SIGCOMM 2013 conference on SIGCOMM*, pages 159–170, 2013.

Appendix

A Packet Loss Rate and Retransmission Accumulation

Consider clouding gaming flow across a bottleneck node with a shallow bottleneck buffer. Assume the forwarding rate of the bottleneck node is R_2, and the incoming flow rate of the video stream is R_1. Consider the sequence of video frames that experience packet loss before the sender reduces its transmission rate. The first video frame to encounter packet loss corresponds to the moment when the buffer at the bottleneck becomes fully occupied. At this point, the subsequent video frames will experience a packet loss rate of $\frac{R_2-R_1}{R_2}$, as only a fraction $\frac{R_1}{R_2}$ of the incoming packets can be forwarded while the remainder are dropped due to persistent buffer overflow.

For the initial frame in which packet loss occurs, the buffer is just filled, and only the latter portion of the frame's packets are subject to loss, while the earlier portion is transmitted successfully. Consequently, the packet loss rate for this first affected frame is lower than that of the subsequent frames, where the buffer remains persistently full and the loss rate stabilizes at $\frac{R_2-R_1}{R_2}$.

Furthermore, if background flows appear and cause a sudden burst of background traffic, the total arrival rate at the bottleneck would increase, further exceeding the forwarding capacity. This leads to an even higher packet loss rate for the subsequent video frames, as a larger proportion of packets are dropped due to intensified buffer overflow. This behavior is different from random packet loss, where the loss probability remains statistically uniform and independent of traffic dynamics, and thus does not exhibit such a temporal escalation in loss rate. Therefore, congestion loss leads to a higher packet loss rate for subsequent frames. We can conclude that the loss rate ratio is greater than 1 under congestion loss.

Retransmission storm: When the sender forwards and retransmits packets, exceeding the link's capacity, it can aggravate network congestion. More congestion triggers more packet losses, implying the need to retransmit more packets, thus resulting in a vicious cycle. Thus, the in-flight ratio becomes greater than 1 under congestion loss. A deeper analysis reveals that excessive in-flight retransmission accumulation is primarily driven by retransmission storms: when congestion loss occurs, retransmitted packets are also frequently dropped, often requiring multiple retransmission attempts before successful delivery. This exacerbates congestion and leads to substantial in-flight retransmission packet buildup.

B Preliminary Packet Loss Analysis in Cellular Networks

We collected cellular data from 12219 online users with the bitrate cap set to 20 Mbps and conducted preliminary analy-

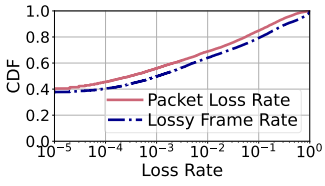


Figure 31: Dist. of packet loss rate and lossy frame rate.

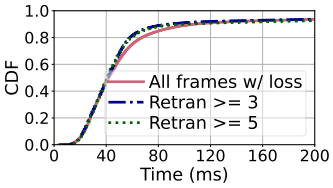


Figure 32: Dist. of end packet queuing time.

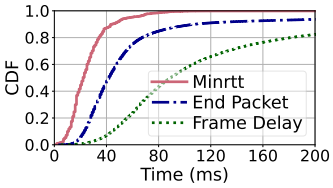


Figure 33: Retx ≥ 1.

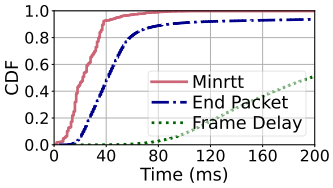


Figure 34: Retx ≥ 3.

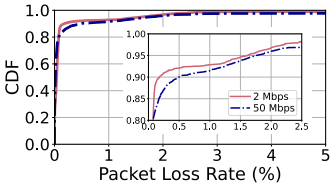
Algo.	Loss rate	Lossy frame	Lossy and >100/200ms	Lossy in >100/200ms
Copa	0.86%	5.35%	1.41%/1.05%	55.9%/59.3%
Salsify	1.34%	4.96%	1.46%/1.01%	52.6%/48.2%
SQP	0.67%	4.97%	1.36%/0.10%	65.5%/69.7%
Pudica	**0.31%**	**4.51%**	**0.18%/0.11%**	**44.4%/43.7%**

Table 3: Comparison of delay-based CC.

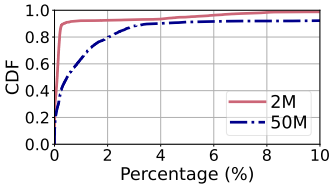
ses. Despite reducing the bitrate cap and adopting the Pudica algorithm to reduce latency, the 100ms stall rate among cellular network users remains as high as 4.8%, of which 23.9% is affected by packet loss. Among all frames, 2.6% are affected by packet loss, and 17.3% are frames with at least three retransmissions. Nearly 80% of packet losses occur at queuing delays of less than 30ms, as shown in Fig. 32. Since the RTT before packet loss is relatively high, with a median of 50ms, retransmissions triggered by packet loss are highly prone to causing stalls. For 90% of frames with $Retx \geq 3$, their latency exceeds 100ms. In summary, due to the high volatility of cellular networks, the proportion of impact caused by packet loss is relatively small.

C Tuning Parameters of the FEC Scheme

In our FEC scheme, a larger k value requires more loss packets in long loss events to be observed in order to enable FEC, and a larger t_q value effectively prevents packets sent when RTT is lower from protected by FEC. Such adjustment makes the FEC scheme more precise (lower unnecessary overhead), at the cost of the amount of packets that it could protect (lower performance gain). To find the value of k and t_q that best balance overhead and performance gain, We employ different k and t_q values and plot the outcome of different parameter values in Tab.4 and Tab.5, respectively. We use the largest possible value that avoids significant drop in performance gain, thus we set k to 1 and t_q to 50ms, respectively.



(a) Dist. of packet loss rate, with different protocols.



(b) Dist. of frame lossy rate, with different protocols.

Figure 35: Difference in loss with different bitrate.

k	Ethernet		Wi-Fi	
	#Frames w/ FEC	#Frames Recovered	#Frames w/ FEC	#Frames Recovered
0.5	5724672	2535742 (44.29%)	2001392	327990 (16.39%)
1	3879562	1910912 (49.26%)	668441	251586 (37.64%)
2	1844785	1032303 (55.96%)	462037	207635 (44.94%)
3	1399700	814801 (58.21%)	394029	184670 (46.87%)
4	1063117	634119 (59.64%)	355609	168798 (47.47%)
5	667591	368162 (55.15%)	339670	163206 (48.05%)

Table 4: k value versus effect

t_q	Ethernet		Wi-Fi	
	#Frames w/ FEC	#Frames Recovered	#Frames w/ FEC	#Frames Recovered
30	5496177	4751749 (86.46%)	2490653	1300250 (52.21%)
50	4183545	3610372 (86.30%)	2188619	1125044 (51.40%)
100	2667473	2340660 (87.75%)	1556906	833984 (53.57%)
200	1411146	1269421 (89.96%)	786929	480590 (61.07%)
500	400032	345846 (86.45%)	233126	167667 (71.92%)

Table 5: t_q value versus effect