

# Supplementary Materials

## A FURTHER MEASUREMENT STUDY ON VIDEO FREEZING OF LIVE STREAMING

Live streaming has a high requirement for data timeliness that can affect and reflect client-side QoE, in which the traffic data that fails to reach receivers in time will introduce video freezes of players, e.g., Tiktok Live and Youtube Live. To better depict data timeliness, both freezing frequency and duration on the client side are usually employed by live platforms to evaluate their selected CDN performance. We (as a CDN vendor) can learn video freezing occurs once the player buffer becomes empty, i.e., no live data can be uploaded to the player before all cached data in the player has been consumed. This is because (i) the player rendering speed exceeds network goodput from senders to receivers; (ii) the lost data cannot be recovered in time, causing longer intra-stream HOL blocking time, especially in the common designs of TCP and QUIC protocol. This paper mainly focuses on enhancing the timeliness of loss recovery by optimizing recovery latency without introducing significant overhead. In this paper, client-side video freezing specifically refers to the freeze caused by HOL blocking due to untimely loss recovery. Video rendering freezes are out of our research.

To further explore the performance of live streaming in terms of video freezing, we have made further large-scale measurements and collected one-year player freezes of a famous live platform in 4 areas (e.g., East Asia, Southeast Asia, Latin America, and the Middle East). For more convenient presentations, we aggregate the measurement results and show every-100s freezing times and duration for each live stream, as Figure 1 shows. The following observations can be obtained.

**Observation #1: The current player freezing of live streams is less-than-perfect.** For every 100s<sup>1</sup>, clients suffer from video freezing of over 1.3s (worse still, ~2.5s in Area 4), on average (as Figure 1(a) shows), in which nearly 1-time freeze usually makes customers intolerable. Meanwhile, video freezes of more than 2s or 2 times often appear after 20:00, as Figure 1(b) shows, which shows the freezing changes in the "best-performed" Area 1.

**Observation #2: The same transmission control incurs differentiated video freezes.** With differentiated network conditions, the identically employed sending and recovery policy introduces diverse freezing results in 4 areas, where the clients in Area 4 experience poorer QoE and more HOL blockings than the other 3 areas, as Figure 1(a) shows. In particular, each video freeze of Area 4 lasts over 2s, on average, significantly higher than others. Besides, the dynamic network status over time in a day also introduces obvious fluctuations of video freezes. For example, nearly 2× freezing in terms of both times and duration can easily arise in the evening (e.g., 20:00 ~ 23:00) compared to in the morning (e.g., 3:00 ~ 7:00), as Figure 1(b) shows.

Therefore, the pre-configured and fixed transmission (e.g., loss recovery) policy cannot well adapt to differentiated conditions and the always-changed status of networks.

<sup>1</sup>Our measurements also show the average QUIC connection lifetime of each live stream is 99.6s.

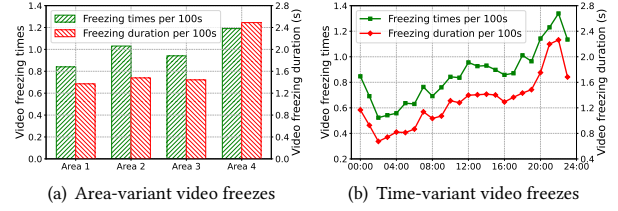


Figure 1: The times and duration of player freezes.

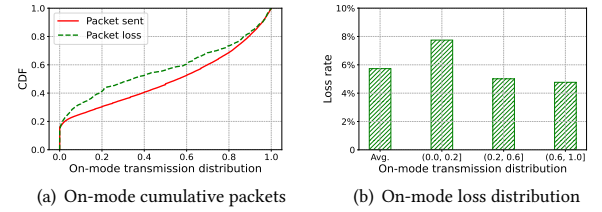


Figure 2: The off-mode is control-unfriendly.

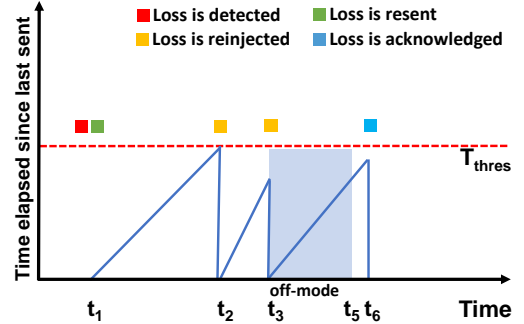


Figure 3: An example of how AutoRec reinjects packets.

## B FURTHER ANALYSIS OF ON-OFF MODE

### B.1 On-Off Mode Switching is Challenging

The on-off mode switching is unfriendly to transmission control. In this experiment, we conduct large-scale measurements in a famous live-streaming application. Figure 2(a) describes the cumulative distribution of lost (or sent) packets in different on-mode stages. We can learn more aggressive actions are performed at the early stage of on-mode, which results in more detected packet losses. For example, in the first 20% of each on-mode duration, 30% of packets are sent out, incurring 40% of all detected losses. Besides, this early stage also introduces more deteriorated loss\_rate (i.e., 7.8%) compared to the average values (i.e., 5.8%, 5.0% and 4.8%, respectively) of the entire and other stages in each on-mode, as shown in Figure 2(b). This is because the existing transmission controls were originally designed for bulk traffic, which cannot be well matched with the ubiquitous on-off mode of live streams.

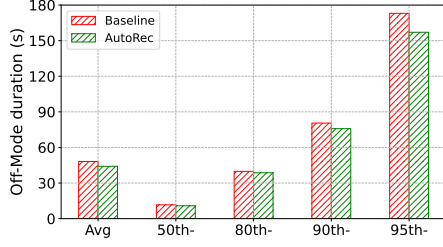


Figure 4: The average cumulative duration of off-mode for each stream.

## B.2 Off-Mode Can be Used for Reinjection

Figure 3 illustrates an example of AutoRec-based loss reinjection with several critical events related to packet loss handling. In this figure, a lost packet is detected and resent out at  $t_1$ . At time  $t_2$ , the time elapsed since the last transmission of this packet has exceeded threshold  $T_{thres}$ . Therefore, we will reinject a replica of this packet even if the sender is currently not in off-mode. From  $t_3$  to  $t_5$ , the AutoRec sender enters off-mode, which enables another loss duplicate to be reinjected. At  $t_6$ , one of the reinjected replicas is acknowledged and the lost data will be deleted from the reinjection queue.

## B.3 AutoRec Reduces Off-Mode Duration

The proposed AutoRec aims to leverage ubiquitous off-mode to reduce the unsatisfied recovery latency of live streaming, in which this on-off mode switching has already been demonstrated unfriendly to the existing transmission controls in [1, 2]. During our real-network evaluations, we also found the control-unfriendly off-mode can be lowered in terms of its duration, as Figure 4 shows. The average cumulative duration can be decreased by 8.47% with the value of 4.1s, whose 80th-, 90th- and 95th-percentile values are reduced by 1.2s (3.0%), 4.7s (5.9%) and 16.0s (9.2%), respectively.

## C FURTHER IMPLEMENTATION DETAILS OF QUIC-BASED AUTOREC

To achieve the required function of reinjection queue, we have added the following attributes for each data in the existing unacknowledged packet queue (unack\_pkt\_queue) in LSQUIC: (i) data\_resent\_unacked that is used to mark the lost data that has been resent but unacknowledged by its receiver.; (ii) rein\_times that records the performed reinjection times; (iii) last\_resent\_time that shows the last retransmission timestamp. The first data (with data\_resent\_unacked = 1) in the head of unack\_pkt\_queue will be fetched and then be resent out when the off-mode is entered or time elapsed since it last sent exceeds threshold  $T_{thres}$ . If the rein\_times exceeds the threshold  $A_{thres}$ , data\_resent\_unacked will be marked as 0. The overview of the AutoRec implementation can be depicted as Figure 5 shows, in which the orange and blue areas represent newly added and upgraded functions, respectively. The send\_ctl\_can\_rein function can decide whether to perform loss reinjection. Besides, we can leverage send\_ctl\_unacked\_remove and send\_ctl\_unacked\_append to maintain the reinjection queue by inserting and deleting lost data. In particular, the presented Redundancy Adapter is implemented and achieves the reinjection

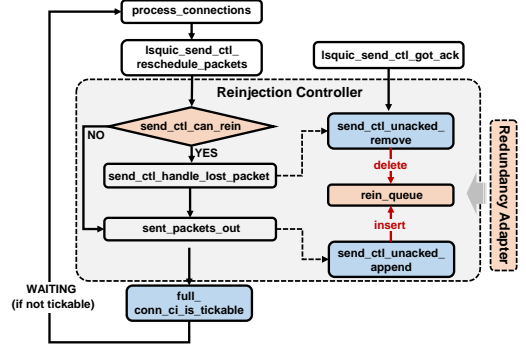


Figure 5: The AutoRec implementation.

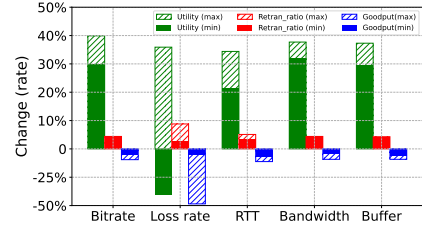


Figure 6: Utility and overhead in testbed experiments.

policy (controlled by  $A_{thres}$ ) by adjusting the conditions of insert, delete and move operations for rein\_queue.

## D FURTHER EVALUATION

In this section, we continue to use the following two metrics defined in the main body of this article to evaluate the performance of AutoRec.

*Recovery latency* is defined as the duration from when any data is detected lost to when resending a recovery packet *that will be successfully received*.  $T_{unit}$  is defined as the sum of the delayed time of ACK packets, RTT, and loss detection time, representing the time elapsed for a data packet from being sent to being retransmitted. recovery latency consists of zero or more  $T_{unit}$ , reflecting the additional time for loss recovery besides first  $T_{unit}$ .

*Recovery deterioration rate* is defined as the ratio between the amount of lost data ( $D_k$ ) that takes two or more  $T_{unit}$  (i.e., recovery latency  $\geq T_{unit}$ ) to be recovered, to the amount of all lost data.

### D.1 Further Testbed Evaluation

The designed utility function  $U(A_{thres})$  is employed to evaluate the tradeoff of recovery latency and reinjection overhead. Figure 6 shows the utility value can be optimized by the ratio of 21.3% ~ 39.9% under different network environments (except for loss\_rate = 15%). In these conditions, retrans\_ratio is additionally introduced by 3.5% ~ 5.1% while goodput is reduced by 3.8% ~ 11.2%. Besides, we can also find higher loss\_rate (e.g., 15%) makes AutoRec hard to achieve a better balance between recovery latency and reinjection cost. For example, when loss\_rate  $\leq 10\%$ , AutoRec can optimize the utility by 11.1% ~ 35.9% while loss\_rate = 15% results in the serious deterioration of utility, goodput and retrans\_ratio by 39.9%, 48.5% and 8.8%, respectively.

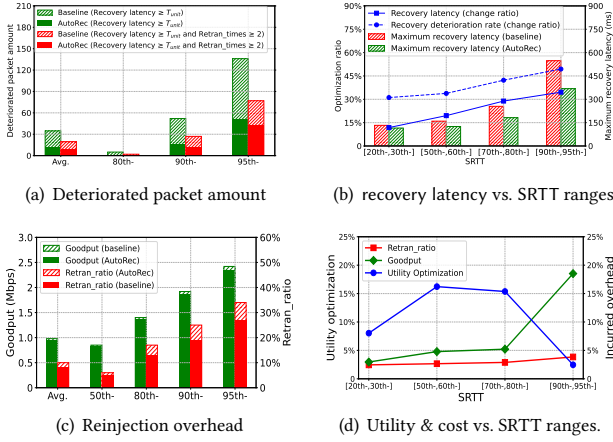


Figure 7: AutoRec benefits and overhead in real networks.

## D.2 Further Real-Network Evaluation

To further demonstrate the recovery deterioration rate benefits, we evaluate the amount of deteriorated packets for each live stream, as Figure 7(a) shows. We can learn 34.7 lost packets, on average, become deteriorated (i.e., recovery latency  $> T_{unit}$ ) in the baseline scheme, which is optimized to 11.8 (by AutoRec) with a ratio of 66.0%. Besides, 5% and 10% live streams suffer from 85 and 36 deteriorated packets less, respectively. This figure also depicts the amount of deteriorated losses that actually require two or more retransmissions (i.e., retran\_times  $\geq 2$ ) for their recoveries. We can learn the average packet amount with recovery latency  $> T_{unit}$  and retran\_times  $\geq 2$  is optimized by 54.3%, whose 90th- and 95th-percentile is lowered by 15 and 34, respectively. As RTT have significant effects on client-side waiting time for loss recovery, we evaluate the recovery latency benefits under the different ranges of SRTT. As Figure 7(b) shows, as SRTT increases, the optimization effect of AutoRec on recovery latency also improves. For streams with a larger SRTT (i.e., SRTT in the last 20%), AutoRec can reduce the recovery deterioration rate and recovery latency by 50% and 35% respectively. Therefore, AutoRec can achieve targeted optimization for recovery latency of high-RTT live streams.

As for the incurred cost, Figure 7(c) shows AutoRec makes the average goodput to deteriorate only with a ratio of 3.1% (from 983.5Kbps to 952.9Kbps). In particular, the incurred goodput deterioration keeps controllable within a stable range from 2.89% to 3.41%, which is based on the observed 50th-, 80th-, 90th- and 95th-percentile values. Besides, only 2.5% retran\_ratio is additionally introduced, on average, in which only 10% live streams require an extra 7.1% retran\_ratio for their loss recoveries. As Figure 7(d) shows, For streams where SRTT is in the top 80%, AutoRec demonstrates a substantial optimization effect on utility (up to 17%), causing at most a 5.1% deterioration in goodput and a 3.6% increase in the additional retran\_ratio. However, for streams with a larger SRTT (i.e., SRTT in the last 20%), AutoRec's optimization effect on utility is not significant and the goodput is seriously affected by loss re-injection. Therefore, we can see that AutoRec is capable of effectively keeping the introduced overhead within our acceptable range in most case.

## REFERENCES

- [1] Tao Zhang, Jianxin Wang, Jiawei Huang, Jianer Chen, Yi Pan, and Geyong Min. Tuning the aggressive TCP behavior for highly concurrent HTTP connections in intra-datacenter. *IEEE/ACM Transactions on Networking*, 25(6):3808–3822, 2017.
- [2] Yuchung Cheng, Neal Cardwell, Soheil Hassas Yeganeh, and Van Jacobson. Delivery rate estimation. *IETF. Internet-Draft draft-cheng-icrg-delivery-rate-estimation-02*, 2022.