Accelerating Loss Recovery for Content Delivery Network

Tong Li, *Member, IEEE,* Wei Liu, *Student Member, IEEE,* Xinyu Ma, Shuaipeng Zhu, Jingkun Cao, Duling Xu, Zhaoqi Yang, Senzhen Liu, Taotao Zhang, Yinfeng Zhu, Bo Wu, Kezhi Wang, *Senior Member, IEEE* Ke Xu, *Fellow, IEEE*

Abstract—Packet losses significantly impact the user experience of content delivery network (CDN) services such as live streaming and data backup-and-archiving. However, our production network measurement studies show that the legacy loss recovery is far from satisfactory due to the wide-area loss characteristics (i.e., dynamics and burstiness) in the wild. In this paper, we propose a sender-side Adaptive ReTransmission scheme, ART, which minimizes the recovery time of lost packets with minimal redundancy cost. Distinguishing itself from forward-error-correction (FEC), which preemptively sends redundant data packets to prevent loss, ART functions as an automatic-repeat-request (ARQ) scheme. It applies redundancy specifically to lost packets instead of unlost packets, thereby addressing the characteristic patterns of wide-area losses in real-world scenarios. We implement ART upon QUIC protocol and evaluate it via both trace-driven emulation and real-world deployment. The results show that ART reduces up to 34% of flow completion time (FCT) for delay-sensitive transmissions, improves up to 26% of goodput for throughput-intensive transmissions, reduces 11.6% video playback rebuffering, and saves up to 90% of redundancy cost.

Index Terms—loss recovery, QUIC, wide-area network, reinforcement learning, ARQ, FEC

1 Introduction

The ubiquitous wide-area packet loss is a critical factor affecting the performance of content delivery network (CDN) services including both delay-sensitive services (e.g., live streaming, web conferencing, interactive online gaming, and remote procedure call (RPC) based services) and throughput-intensive services (e.g., disaster recovery, cloud migration, and data backup-and-archiving). Take a famous CDN platform as an example, the flows incur an average 5.2% packet loss rate in Turkey and 3.8% in Brazil, respectively. The flow completion time (FCT) in these regions is enlarged because the high loss rate introduces head-of-line (HOL) blocking and even incurs service failure when loss recovery is excessively delayed.

There exist two fundamental ways of loss recovery in transmission control, i.e., forward-error-correction (FEC) [1] and automatic-repeat-request (ARQ) [2], [3]. However, it is well-studied that FEC is far from satisfactory due to the wide-area loss characteristics such as *burstiness* in the wild [1]. Although some arts are proposed to protect against bursts of losses [1], [4], they require dual-side (i.e., CDN server and client-side application) modifications. These FEC-based advancements might suffer from deployment issues in the multi-supplier CDN market. As shown in Figure 1, application operators (e.g., TikTok Live, Huya Live, and Tencent Meeting) usually apply the Multi-Supplier Strategy [5] in the CDN market. As a result, it is the CDN vendor's duty to optimize the transmission performance (e.g., loss recovery), according to which the application operators will choose the better-performing CDN vendors to carry more traffic (i.e., more significant market share). In this case, only server-side sending policies can be adjusted by the selected CDN vendors, which lack the proper authority to synchronize client-side control rules.

Most modern applications only apply the ARQ paradigm to control loss tolerance as the commercial solution, which retransmits data once any packet is detected lost. Unfortunately, from our production network measurement studies, we find that wide-area loss shows characteristics of dynamics and burstiness in the wild (see §3).

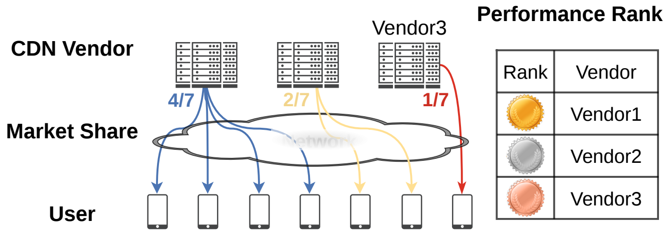


Fig. 1: An example of the multi-supplier market for wide-area applications.

This work is supported in part by the National Natural Science Foundation of China under No. 62202473 and No. 62441230, the Science Fund for Creative Research Groups of the National Natural Science Foundation of China under No. 62221003, the Key Program of the National Natural Science Foundation of China under No. 61932016 and No. 62132011, the National Science Foundation for Distinguished Young Scholars of China under No. 62425201. Corresponding author: Bo Wu.

- *T. Li, W. Liu, X. Ma, S. Zhu, J. Cao, D. Xu and Z. Yang are with the DEKE Lab and Information School, Renmin University, Beijing, China, 100872. E-mail: {tong.li, 2022104277, 2022104205, 2022104214, 2021201503, 2022000914, 2021201814}@ruc.edu.cn*
- *S. Liu, T. Zhang, and Y. Zhu are with the NetLab, ByteDance, Beijing, China, 100086. E-mail: {liusenzhen, zhangtaotao.1999, zhuyinfeng01}@bytedance.com*
- *K. Wang is with the Department of Computer Science, Brunel University of London, United Kingdom, UB8 3PH. E-mail: kezhi.wang@brunel.ac.uk*
- *B. Wu and K. Xu are with the Department of Computer Science and Technology, Tsinghua University, Beijing, China, 100084. E-mail: wub14@tsinghua.org.cn, xuke@tsinghua.edu.cn*

We further find that the legacy ARQ-based loss recovery induces unexpected latency due to poor adaption to these loss characteristics. Specifically, it still suffers from both *data reassembling starvation* and *receiving buffer starvation* in delay-sensitive transmissions and throughput-intensive transmissions, respectively (see §4).

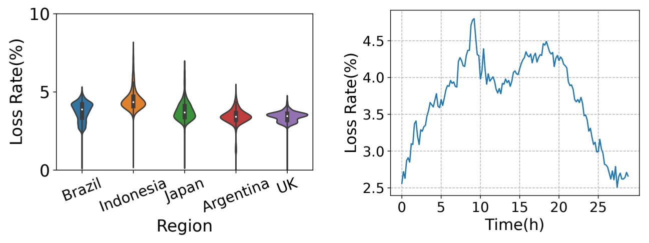
In this paper, we propose the adoption of FEC to improve loss recovery in ARQ-based protocols. Unlike the traditional FEC-based approach, which involves sending redundant unlost data packets, our method focuses on sending multiple packets replicating the content of packets declared as lost without the need for coding, thereby avoiding dual-side modifications and expediting the loss recovery process. Furthermore, our approach can potentially eliminate the costly traffic overhead typically associated with traditional FEC. This is because the number of lost packets is usually much smaller than the number of unlost packets. Introducing redundancy to retransmission may seem straightforward, but in practice, the dynamics and burstiness of packet loss present several challenges that must be addressed: (i) The trade-off between performance and cost is decided by the redundancy level, which refers to the number of replicas of a lost packet. The optimal redundancy level for each loss varies with the dynamics of the loss, and striking the right balance is crucial. (ii) A loss event usually contains multiple consecutive lost packets (i.e., burstiness), and the redundant replicas might be lost together again when burst losses occur.

To tackle these issues, we present ART[1] (Adaptive ReTransmission), a sender-side approach without requiring any modifications on the receiver side. Simple but useful, ART overcomes the above-mentioned challenges through two tightly coupled systems: Redundancy Adaption and Replica Scheduling. To adapt to loss dynamics, Redundancy Adaption alters the redundancy level step by step using the method of test and then verification (i.e., *test-and-verification method*). In addition, we have incorporated reinforcement learning into packet loss recovery, using it to adjust the redundancy level. To deal with the loss bustiness, Replica Scheduling schedules the replicas in a random number of sending cycles (one cycle equals the interval of sending one packet at a specific pacing rate). With Redundancy Adaption and Replica Scheduling, ART can achieve consistently rapid loss recovery with the minimized cost of redundant traffic.

The contributions are summarized as follows.

- We conduct large-scale measurement studies on the wide-area loss characteristics in the wild and disclose that dynamics and burstiness are two key features of loss in practice (see §3). We also declare that FEC-based loss recovery is far from satisfactory due to poor adaptions to burst losses as well as deployment issues (see §4).
- We identify the critical challenges when facing the wide-area loss characteristics in the wild, and then propose ART that can dynamically adjust the redundancy level according to the loss dynamics, and can randomly schedule the sending of each replica to protect against bursts of losses (see §5 and §6).

1. The open-source implementation of ART upon QUIC is maintained at https://github.com/litonglab/quic-art.



(a) Loss distribution across regions (b) Loss dynamics

Fig. 2: (a) Example of loss distribution in different regions. (b) Example of loss dynamics in the same region at different time periods in the wild.

- We implement ART prototype in the user-space QUIC protocol and deploy it on both the testbed and production network (see §6.3). The experimental results show the practicability and profitability of ART, in which ART can accelerate loss recovery with the minimum extra traffic overhead by adjusting the sender's loss recovery capability. For example, ART reduces the FCT by up to 34% for delay-sensitive transmissions when suffering data reassembling starvation. ART also improves up to 26% of goodput for throughput-intensive transmissions when suffering receiving buffer starvation. Furthermore, ART achieves up to 90% of lower redundancy cost with considerable recovery performance (see §7).

2 Related Work

FEC-based loss recovery. The field of packet loss recovery has seen many works related to FEC [1], [6]–[8]. However, most of them can not effectively address the issue of burst packet loss [6]–[8]. Although some work [1] has provided solutions to handle burst packet loss, they involve high implementation complexity and require corresponding modifications on the client side, making them unsuitable for multi-vendor scenarios. In contrast, ART is a lightweight design that only requires server-side (sender-side) modification.

ARQ-based loss recovery. The TCP stack has proposed a series of ARQ-based loss recovery improvements [9]–[12]. Among them, FACK [13] and RACK-TLP [14] have been proven to significantly reduce loss detection time in many cases. In addition to these sender-side loss detection algorithms, QUIC [15] and TACK [16] can detect loss at the receiver side according to the monotonically increasing packet number. These works, however, mainly rely on rapid loss detection (i.e., optimizing T_{single} in Equation (2)). In contrast, this paper focuses on reducing retransmission rounds (i.e., optimizing N in Equation (2)), which is also a key influence factor in ARQ-based loss recovery.

Redundant Retransmission. There also exist some studies that employ the idea of redundancy, most of which are applied in wireless sensor networks [17], [18]. Furthermore, their redundancy level is often set to a fixed value [19], [20]. [21] also follows a similar approach of adding redundancy only to lost packets. However, it is focused on WebRTC and does not tackle the new challenges present in QUIC-oriented scenarios, such as Receiving Buffer Starvation. The most related work to ART is OR3 [22], which adopts a rule-based way to decide the redundancy level according to the retransmission round. However, our comparison experiments show that there is still room to reduce the recovery time and redundant traffic due to the dynamics of packet loss in the wild. ART differs from OR3 in two main aspects. Firstly, adjusting redundancy in ART is based on the network state over a period of time, whereas OR3 adjusts redundancy based solely on the reception of individual packets. Therefore, ART's redundancy adjustment is more stable than OR3. Because the redundancy adjustment in ART will not vary significantly due to differences in the retransmission rounds of preceding and subsequent packets. Secondly, although both ART and OR3 use a random replica scheduling approach for packet scheduling, ART further restricts the dispersion of packets (their latest send time) based on the burst size. This additional constraint enables ART to better accommodate the characteristics of loss burstiness.

ML-based Transmission Control. Many works [23], [24] have already utilized deep learning, including some aiming at improving network transmission performance [25]. However, Reinforcement Learning (RL) has been used to tackle congestion control challenges. For instance, in [26], multi-armed bandit algorithms are employed to adjust the initial window for congestion control, aiming to enhance the transmission time of small flows. In [27], RL is employed to configure routing, addressing traffic engineering challenges in Software-Defined Networking (SDN). In [28], a framework based on Partially Observable Markov Decision Processes (POMDP) is used to model network congestion and improve energy efficiency in data centers. However, these endeavors have not applied RL to address packet loss recovery issues. This paper marks the first instance of applying RL specifically to loss recovery enhancement.

3 Loss in the Wild: A Measurement Study

Logs from our production network (i.e., the ByteDance public cloud) are collected from a random sample over two weeks. Each log corresponds to a QUIC connection, which contains a sequence number and packet sending and loss information of multiple QUIC streams. We measured over 200,000 connections from different application scenarios in different regions all around the world.

3.1 Loss Shows Dynamics

The standard deviation in loss rate can be used to indicate how evenly the loss rate is distributed during the measurement. We first explore the standard deviations of the loss rate (every 5 minutes) of each connection in different regions. Figure 2(a) shows the results. Although there are regions with similar average packet loss rates, they have different packet loss rate distributions. For example, Brazil and Japan both have an average packet loss rate of 3.78%, but Japan has the highest packet loss rate of 7.1%, while Argentina has only 5.7%. Figure 2(b) further gives an example of how the packet loss rate evolves. Specifically, the loss rate is never constant and varies from 2% to 5% during 24 hours. This confirms that loss shows dynamics in the wild.

3.2 Loss Shows Burstiness

Figure 3 gives an example of loss burstiness (denoted by red blocks) in a randomly selected connection. We further explore the burst loss distribution in the production network.

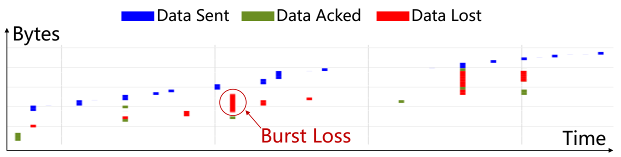
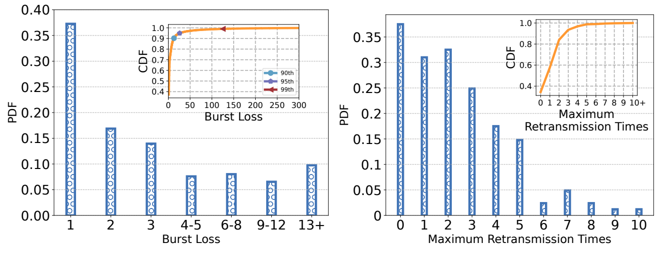


Fig. 3: An example of loss burstiness in the wild. Data is collected from real-world networks. Burst loss refers to consecutive packet losses, and the small red block on the left represents individual packet loss.



(a) Burst size of loss (b) Max Retransmission times

Fig. 4: (a) Burst loss distribution in the wild. The x-axis is the number of continuously lost packets. (b) Maximum retransmission times distribution in the wild.

Figure 4(a) shows the results. Surprisingly, the probability of only one packet being lost is not as high as imagined (i.e., 37.2%), instead, the probability of losing multiple packages ($\geq$ 2) together accounts for a larger proportion (i.e., 62.8%) (see Figure 4(a)). Specifically, as illustrated in the sub-figure of Figure 4(a), the 90^{th}, 95^{th}, and 99^{th} percentile burst size of loss (i.e., the number of continuously lost packets) is 13, 27, 125 respectively, showing extremely bursty packet loss. Numerous factors contribute to the emergence of the phenomenon of loss burstiness. Here we provide readers with a duo of potential explanations worthy of contemplation. First, it might be due to the queue management strategy [1] or traffic handling policy [29] employed by the Internet Service Provider (ISP). Moreover, wireless interference in unstable access networks of end users [30] may be a reason.

3.3 Retransmission Loss is Ubiquitous

A packet might be retransmitted more than once before it is correctly received by the receiver. By counting the number of occurrences of each frame according to the QUIC *stream offset*, we can further analyze how many times each packet was retransmitted before being successfully received. Given a connection, its maximum retransmission times can be computed as the maximum value of retransmission times among all packets in a connection. Figure 4(b) shows the results of the distribution of the maximum retransmission times in the production network. We find that the proportion of connections with maximum retransmission times of two or more exceeds 43%. Among them, many connections have certain packets that are retransmitted even more than 10 times (loss rate around 9.6%). This retransmission loss is harmful to both delay-sensitive applications and throughput-intensive applications, as we will discuss next.

4 Wide-Area Loss Recovery: Issues and Challenges

FEC and ARQ are two fundamental techniques for loss recovery. In this section, let's first discuss why we tend to

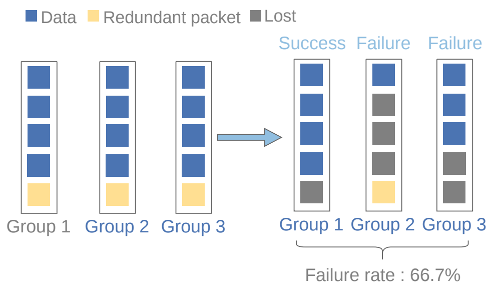


Fig. 5: An example of FEC failure when encountering burst loss. The group coding length is 4, and the number of redundant packets (redundancy level) is 1.

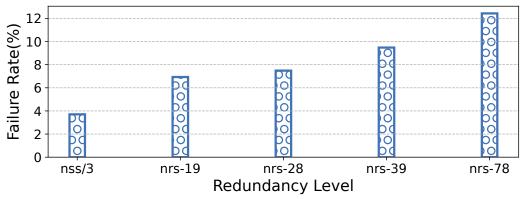


Fig. 6: FEC failure rate in the wild. The x-axis is the coding length, where nss refers to the number of source symbols and nrs refers to the number of repair symbols. The nss/3 means that for every 3 original packets, one redundant packet is encoded (the same applies to nrs).

use ARQ rather than FEC in production networks. Next, we will elaborate on the issues and challenges that must be addressed in the legacy ARQ paradigm.

4.1 Why Not FEC?

Poor adaption to burst loss. FEC adds redundant packets to the transmitted data, which allows the receiver to identify and correct errors in the received data. However, as illustrated in Figure 5, when encountering burst loss and the number of lost packets exceeds the FEC redundancy level, the receiver may fail to recover, which is called *FEC failure*. We further investigate the FEC failure rate of our production network in a specific region over some time (with burst loss as shown in Figure 4(a)), with the implementation of FEC referenced from [31]. Figure 6 shows the results. It is demonstrated that the percentage of connections experiencing FEC failures ranges from 3% to 10% of the total connections, with some cases having a failure rate of up to 12%. It is worth noting that a larger coding length leads to a higher failure rate for FEC. This occurs because if packet loss is detected and resolved through retransmission before the transmission of redundant packets in a group is completed, the failure of FEC takes place. The probability of FEC failure increases with the length of coding in the group.

Poor generalization to application. Some other notable features of FEC are important for reasoning why FEC is not suitable for our production network: (1) **Cost-inefficient when no loss occurs.** FEC continuously sends redundant packets even in the absence of packet loss, resulting in a complete waste of bandwidth. Although selectively enabling or disabling FEC can be a potential solution, the challenge lies in determining the optimal timing to turn it on or off [6]. (2) **Deployment issues in a multi-supplier CDN market.** FEC requires support from both the client and server sides [32], [33], which poses challenges for its universal deployment in our production network from the perspective of a CDN vendor. This is because, in a multi-supplier CDN market where different CDN vendors serve a certain application, the application operator may not necessarily include FEC support as a default option. (3) **Good for loss-sensitive CCAs but weak for non-loss-based.** FEC's ability to reduce the packet loss rate perceived by the sender side can be beneficial for congestion control algorithms (CCAs) that are sensitive to packet loss [32], [33], such as Reno and Cubic. However, the mainstream non-loss-based CCAs like BBR in our production network benefits less from FEC. This issue has already been addressed by [4] and will not be discussed further in this paper.

4.2 Why ARQ?

As FEC may not always be the optimal choice, let's consider using ARQ. Here are a few reasons why ARQ could be a better option.

Cost-efficient. From the perspective of CDN vendors, traffic cost is an important metric. To minimize traffic costs, ARQ is often considered a more effective approach than FEC. While FEC typically involves the addition of redundant data to the original transmission for error correction, ARQ relies on the sender responding to requests from the receiver for the retransmission of lost data only when necessitated. This helps to minimize unnecessary data transmission, reducing overall costs. Although spurious retransmissions can sometimes occur with ARQ, there are well-established detection algorithms available to identify and address these issues [34].

Single-side deployment. Another major advantage of ARQ is that it operates without requiring extra support from the receiver. Optimizing ARQ algorithms typically only requires modifications to the sender (i.e. CDN server), which is particularly valuable in multi-supplier CDN markets where receiver-side modifications can be expensive and unreliable. By operating independently of the receiver, ARQ offers a more flexible and resilient approach to loss recovery.

A single retransmission is not painful. One of the criticisms leveled at ARQ is that it can introduce delays during the packet recovery process. This is because it can take an extended period (typically greater than one RTT) to detect and retransmit lost packets. However, we argue that overall, the delay incurred by ARQ during retransmission is relatively insignificant and adds only a few milliseconds to the delivery time. This is largely due to the benefits afforded by CDNs, which bring content closer to users and reduce end-to-end latency for most Internet services. While multiple retransmissions can cause excessive delays for a particular packet, we believe that retransmitting a packet once is rarely a problem as long as successful packet delivery can be ensured. Considering the potentially high cost associated with implementing FEC, we believe that using an enhanced ARQ technology represents a more practical solution that can address these concerns.

Ideally, the successful delivery of a lost packet should only require a single retransmission to the receiver. However, our previous measurement studies have shown that retransmission loss is ubiquitous. Next, we will investigate the impacts on transmission performance when a single

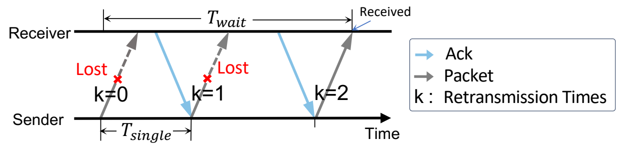


Fig. 7: An example of the retransmission loss.

retransmission attempt is insufficient to successfully deliver a lost packet.

4.3 Delay-Sensitive Transmission Suffers From Data Reassembling Starvation

Consider a distributed system that relies on RPC for inter-process communication. If a critical RPC call is lost and experiences a high recovery latency, the client's request will be delayed, and subsequent dependent operations may be blocked. This can result in a significant increase in flow completion time, negatively affecting the responsiveness and overall performance of the service. Real-world scenarios, such as financial trading platforms or online multiplayer games that heavily rely on quick response times, can suffer from degraded user experience and potential financial losses due to increased recovery latency.

Under these circumstances, the prioritizing loss recovery attempts to mitigate receiver-side waiting time (T_{wait}) for the lost data and enables delivery of the follow-up data (that has already been received) to the application layer. As illustrated in Figure 7, we define T_{wait} as the duration between when a packet is sent and when the packet is first received, and T_{single} as the duration between when a packet is sent and when the packet is detected lost. Then we have

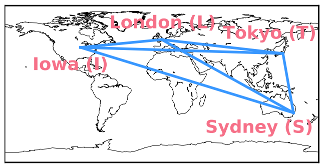
$$T_{wait} = K \cdot T_{single} + \frac{RTT}{2} \tag{1}$$

where RTT refers to the round-trip time (RTT) and K refers to the retransmission times of a specific packet before successfully being received at the receiver. If a packet is never lost and retransmitted, then $K = 0$. Given a certain loss detection algorithm, we have $T_{single} \propto RTT$. Furthermore, given the RTT, we have $T_{wait} \propto K \cdot RTT$.

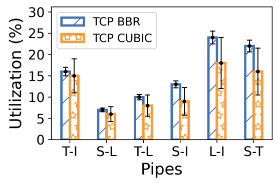
Based on the above analysis, we infer that delay-sensitive transmission suffers from data reassembling starvation in the case of the large K. However, the current loss recovery paradigms mainly rely on rapid loss detection (i.e., optimizing T_{single}) while ignoring retransmission times (i.e., optimizing K). For example, RACK [14] is regarded as the state-of-the-art loss detection advancement that assures a relatively deterministic T_{single}, i.e., $T_{single} \approx 1.25RTT$. In this case, Equation (1) is instantiated as $T_{wait} \approx (1.25K + 0.5)RTT$.

4.4 Throughput-Intensive Transmission Suffers From Receiving Buffer Starvation

Transport protocols like TCP and QUIC [15] provide reliable and ordered byte-stream transmission. As a result, before being handed off to upper applications, the subsequent packets (stored temporarily in the receiver's queue) of the lost packet will be stalled in the receiving buffer until the "hole" (the lost data sequence space) is filled via retransmissions. However, retransmissions might be lost again. Since the receiving buffer required by a connection is closely related to the maximum number of retransmissions, we focus on the metric of K_{max}, the maximum value of K among all packets in a connection, where K denotes the retransmission times of a specific packet. In general, given a certain loss ratio and period, the higher the throughput, the larger the K_{max} [22].



(a) Topology



(b) Bandwidth utilization

Fig. 8: Pipes are verified not full, but single TCP connection only achieves bandwidth utilization of less than 25%. Where bandwidth $bw = 1\ Gbps, rtt \in [100, 300]\ ms$, and average $loss$ is 0.5%. The maximum receiving buffer of TCP is 16 MB in the cloud end-hosts [22].

Based on the above analysis, we infer that throughput-intensive transmissions might suffer from receiving buffer starvation in the case of a large K_{max}. This issue is unremarkable for transmissions when both throughput and loss are low. However, when running under large-BDP and lossy network conditions, the buffer starvation issues may significantly impact the performance of throughput-intensive transmissions. Figure 8 shows an example of how high-throughput transmission bottlenecks the receiving buffer in the Pantheon [35]. Based on the fact that the receiving buffer is usually small (*e.g.*, 16 MB in the *c5.xlarge* instances of Amazon EC2) in modern wide-area Linux servers. It is observed that buffer limitation makes TCP only achieve bandwidth utilization of less than 25% even when there is sufficient available bandwidth.

It's worth noting that network operators often increase the end-host buffer through protocol tuning to address the capability mismatch between loss recovery and high-throughput requirements. However, this approach is ineffective when operators only have control over one side of the network, such as in public cloud services where CDN vendors cannot modify the end devices.

In summary, it may cause both data reassembling starvation and receiving buffer starvation when a single retransmission attempt is insufficient to successfully deliver a lost packet (i.e., when $K > 1$). Hence, it becomes crucial to carefully manage loss recovery at the sender side, minimizing K for each loss when data reassembling is excessively delayed and minimizing K_{max} for all the losses in the connection when the end-host buffer is insufficient. These efforts would provide a valuable contribution from the perspective of CDN vendors, which encourages the development of ART, as we will elaborate on next.

5 The ART Overview

The main objective of ART (Adaptive ReTransmission) is to ensure the successful reception of each lost packet by the receiver, minimizing the receiver-side waiting time of each packet. However, to effectively implement ART in production networks where cloud services (sender side) charge based on traffic volume, it is vital to not only prioritize the quick recovery of lost packets but also minimize

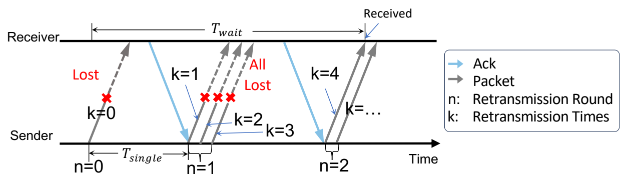


Fig. 9: An example of the retransmission loss when applying redundant retransmission, where the retransmission times $K = max(k) = 4$ and the retransmission rounds $N = max(n) = 2$.

redundancy cost and prevent any detrimental impact on regular packet transmission. Striking a balance between rapid recovery and efficient resource utilization is crucial for the practicality of ART in production networks.

5.1 Problem Formulation

The fundamental concept behind ART is to introduce redundancy during retransmissions, i.e., *redundant retransmission*. When considering redundant retransmissions, it is necessary to re-evaluate the variability in waiting time on the receiver side (represented by T_{wait}). To address this, we propose a novel concept called "retransmission round" (designated as N) to replace the retransmission times (K) in Equation (1) when computing T_{wait}:

$$T_{wait} = N \cdot T_{single} + \frac{RTT}{2} \tag{2}$$

where N refers to the retransmission rounds of a specific packet before successfully being received at the receiver. Figure 9 illustrates an example of the difference between N and K, where n represents the n^{th} retransmission round and k represents k^{th} retransmission times. For each packet, we have $N = max(n)$ and $K = max(k)$. Similar to K_{max}, we define the maximum retransmission round (N_{max}) for a connection that transmits multiple packets as

$$N_{max} = max(N) \tag{3}$$

In the absence of redundancy, these two terms (N and K) are essentially interchangeable. However, in the presence of redundancy, the retransmission times will accumulate within the same retransmission round. Specifically, in this example, the sender retransmits 3 replicas of the lost packet (i.e., $k = 1, 2, 3$) during the first retransmission round (i.e., $n = 1$). Generally, whenever a packet is retransmitted, the count of retransmission times for that packet increases by 1. However, only when all retransmitted packets for a specific packet in a complete retransmission round are identified as lost will the retransmission rounds for that packet be increased by 1.

Given M packets, each packet has a recovery time of T^m_{wait}, $(m = 1, 2, ..., M)$. c_m denotes the redundant traffic cost required to deliver the m^{th} packet successfully. This paper aims to achieve the lowest recovery time of lost packets with bounded redundant traffic costs. The problem can be simply modeled as follows:

$$Z = \min \sum_{m=1}^{M} T^m_{wait}, \quad s.t. \quad \sum_{m=1}^{M} c_m \leq C \tag{4}$$

where C refers to the cost budget to deliver these M packets.

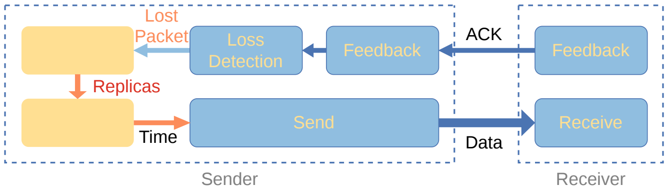


Fig. 10: Key modules in ART. The yellow modules are newly added by ART, and the blue modules are originally included in QUIC.

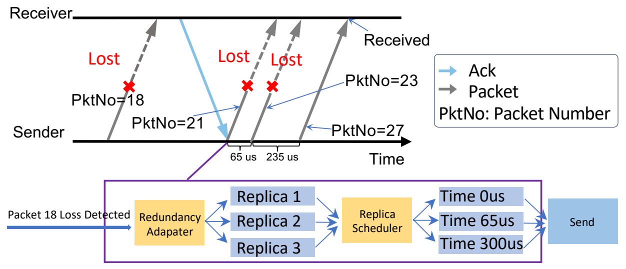
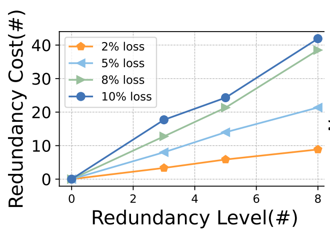


Fig. 11: An example of the ART workflow. The Redundancy Adapter module generated three replicas (packet numbers 21, 23, and 27), which were sent by the Replica Scheduling module at 0 us, 65 us, and 300 us, respectively.

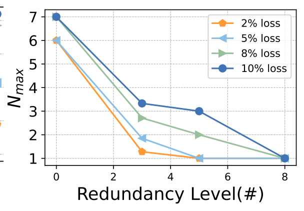
5.2 The ART Framework

ART is a sender-side modification to the protocol stack whose key modules are illustrated in Figure 10. ART is fully compatible with QUIC's encryption logic, and the Redundancy Adaptation becomes operational after the endpoint completes the QUIC decryption process. Particularly, once a loss is detected, ART adopts redundancy adaption to compute the number of replicas of the lost packet that should be retransmitted next. Given the number of replicas, ART then adopts replica scheduling to determine the specific time interval for sending out each replica from the sender. To explain this more clearly, we further give an example of the ART workflow in Figure 11. Assuming a packet (packet number = 18) is detected lost. In this case, ART first runs the redundancy adapter to compute the redundancy level for packet 18, we assume redundancy level = 3. Then, ART runs the replica scheduler to determine when to send each replica. For example, the 3 replicas should be sent in 0 us, 65 us, and 300 us, respectively. As a result, the replicas are sent out with packet numbers 21, 23, and 27, respectively. Since the 3 replicas are sent at various intervals, it can greatly enhance the likelihood of a successful retransmission.

Redundancy adaptation. The redundancy level, denoted as R, is defined as the number of replicas that should be resent for a specific lost packet. R_n represents the redundancy level within the n^{th} retransmission round. For instance, in the case of Figure 9, we have $R_1 = 3$. We define the *redundancy cost* as the total number of replicas that are sent during transmission. Initially, a naive approach for ART would be to use a fixed redundancy level for all rounds, i.e., $R_n = R$ $(n = 1, 2, ..., N)$. However, our deployment experiences have shown that this fixed approach to redundant retransmission is suboptimal (high redundancy cost or long recovery latency) due to the dynamic nature of packet loss in real-world scenarios (see §6.1). Consequently, we incorporate redundancy adaptation to allow the redundancy



(a) Redundancy cost



(b) N_{max}

Fig. 12: Cost and Performance of ART under different redundancy levels, where $bw = 100$ Mbps and $rtt = 100$ ms.

level to vary dynamically. To achieve this, we introduce the *Redundancy Adapter*, which applies a test-and-verification method to gradually learn the feature of loss dynamics and to carefully select the most appropriate redundancy level for each retransmission round of lost packets. This ensures that ART can adapt to the dynamics of packet loss while minimizing the redundancy cost.

Replica scheduling. For each round of retransmission, more than one replica might be injected into the network. Initially, a naive approach for ART would be to send all the replicas (with the same retransmission round) at once if the send window is sufficient. However, our deployment experiences have shown that this burst send pattern may fail to accelerate loss recovery due to the burst nature of packet loss in real-world scenarios (see §6.2), that is, a loss event usually contains multiple consecutive losses, the back-to-back replicas might be all lost again under burst loss. So, the replicas should be carefully scheduled. Consequently, we introduce the *Replica Scheduler*, which adopts randomization to enable the replicas to be sent out in a random number of sending cycles (one cycle equals the interval of sending one packet at a specific pacing rate). This ensures that ART can adapt to the burstiness of packet loss.

6 Detailed Design

In this section, we give the detailed design of ART for its practical deployment.

6.1 Redundancy Adapter

We first elaborate on the design of the Redundancy Adapter by answering the following questions below.

Why dynamic redundancy level matters? To answer this question, we conducted an investigation into the performance of ART under different loss rates ($p = 2\%, 5\%, 8\%, 10\%$) and different redundancy levels ($R = 0, 3, 5, 8$). As shown in Figure 12(a), when the redundancy level increases, there is a corresponding rise in the redundancy cost. On the other hand, the redundancy levels also significantly impact the maximum retransmission rounds (N_{max}). As shown in Figure 12(b), the higher the redundancy level, the lower the N_{max}. Intuitively, the optimal redundancy level should be set at the *inflection point*. However, the experimental results demonstrate that the inflection points vary with the loss rates. We then infer that the redundancy level should be set dynamically according to network dynamics. This greatly motivates the design of the Redundancy Adapter, as we will elaborate below.

How to set redundancy level dynamically? Primarily, we establish the replica loss rate (p_r) by dividing the number

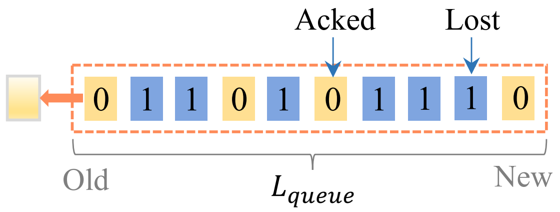


Fig. 13: An example of the sliding-window-based bitmap queue. If a replica is acknowledged, a "0" is appended to the queue; Otherwise, a "1" is appended to the queue.

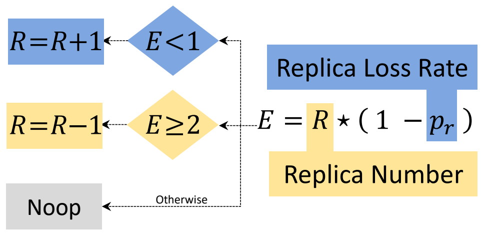


Fig. 14: The design rationale of the Redundancy Adapter.

of lost replicas by the total number of sent replicas. While it is acknowledged that redundancy levels should be adjusted according to the loss rates, it is imperative to consider p_r instead of the general packet loss rate (p) for a more accurate design of the Redundancy Adapter.

In this paper, we propose a test-and-verification method that uses a sliding window to predict p_r according to the historical packet delivery information. First of all, we set up a bitmap queue at the sender. As illustrated in Figure 13. When the sender receives an acknowledgment for a redundant packet, a "0" is appended to the queue; Otherwise, a "1" is appended to the queue, indicating that the redundant packet is lost again. In this case, p_r is computed as the proportion of "1" in the queue, i.e., $p_r = \frac{c}{L_{queue}}$, where L_{queue} is the queue length measured in the number of packets, and c is the number of "1" in the queue. Then, the expected number of successfully delivered replicas (denoted as E) can be computed as follows:

$$E = R \cdot (1 - p_r) \tag{5}$$

where R is the redundancy level within a retransmission round. In general, to ensure the effectiveness of redundant retransmission where we only send the number of replicas that is exactly required by the loss recovery, we should keep $E \approx 1$. This is because the unnecessary traffic cost arises when $E > 1$, and recovery latency arises when $E < 1$. To accomplish this, a step-by-step online algorithm is applied. As illustrated in Figure 14, when $E < 1$, then $R = R + 1$; When $E \geq 2$, then $R = R - 1$; Otherwise, R remains unchanged. It is foreseeable that under certain network conditions, R will fluctuate within a range rather than converge to a fixed value. However, the mean value of R will converge in a statistical sense.

Sensitivity analysis of L_{queue}. Intuitively, a larger L_{queue} results in higher performance and higher redundancy cost, albeit at the expense of increased runtime overhead. Conversely, a smaller L_{queue} yields lower performance, lower redundancy cost, and diminished runtime overhead. In Figure 15, we give examples of the influence of different L_{queue} sizes on redundancy across diverse lossy links. It

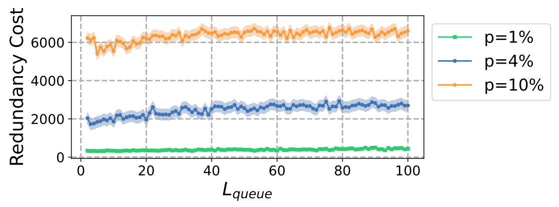


Fig. 15: Examples of the sensitivity analysis of L_{queue}. $bw = 100\ Mbps, rtt = 30\ ms, \bar{p} = 1\% \sim 10\%$. The loss pattern follows Gaussian distribution.

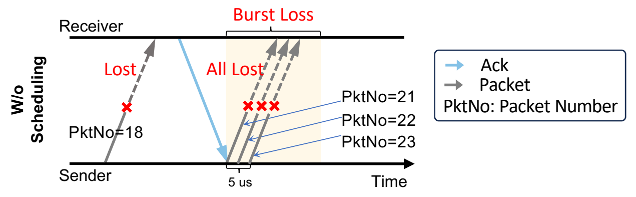


Fig. 16: An example of redundant replicas being sent consecutively.

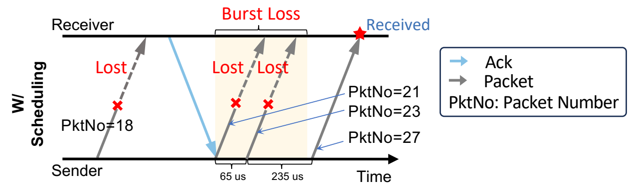


Fig. 17: An example of replicas being sent separately after being scheduled by the Replica Scheduler.

is evident that, within identical network conditions, the overall redundancy level remains relatively stable (goes up very slowly) with the increase of L_{queue} values. Note that L_{queue} should not be set too small because a small L_{queue} (e.g., $L_{queue} = 1$) increases the randomness in the statistics on p_r, resulting in a higher redundancy cost. We further find that from the perspective of performance, the results remain similar. Hence, in this paper, we by default set $L_{queue} \geq 10$.

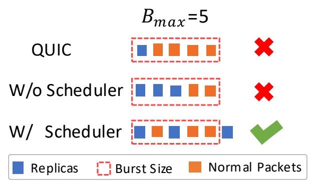
6.2 Replica Scheduler

Figure 16 shows an example where packet 18 is lost, and the Redundancy Adapter generates three replicas with packet numbers 21, 22, and 23. During a burst loss, these three replicas cannot be received by the receiver. Therefore, when burst loss occurs, relying solely on the redundant replicas generated by the Redundancy Adapter is not sufficient.

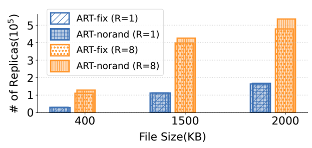
To withstand burst loss, we introduce the Replica Scheduler to ensure prompt delivery of replicas to the receiver. Instead of sending out all the replicas at once, the Replica Scheduler disperses the multiple replicas into a certain number of sending cycles (one cycle equals the interval of sending one packet at a specific pacing rate). This means replicas are interspersed with normal packets. Replica Scheduling now uses a random method to distribute redundant packets. In the future, we may implement a more scientific algorithm for distributing them.

Figure 17 explains how the Replica Scheduler functions in the event of burst loss. After packet 18 is lost, the Redundancy Adapter generates three replicas. The Replica Scheduler schedules these replicas to be sent at 0 us, 65 us, and 300 us. It can be seen that the replica sent after 300 us is successfully received by the receiver.

Given multiple replicas of a specific lost packet, we define the *escape space* (denoted by $escape\_space$) of this



(a) Design rationale



(b) Effect of random

Fig. 18: (a) The design rationale of the Replica Scheduler. (b) The effect of random scheduling, where ART-fix and ART-norand refer to a fixed redundancy level, with and without Replica Scheduling, respectively.

lost packet as the total number of sent packets from starting sending its first replica to finishing sending its last replica. For example, as shown in Figure 18(a), for the traditional QUIC, we have $escape\_space = 1$. For ART without applying the Replica Scheduler, the escape space of each packet equals the redundancy level, i.e., $escape\_space = R$.

As shown in Figure 18(b), we used ART with fixed redundancy and compared the scenarios with and without Replica Scheduling to transfer small files. We found that, under the same network conditions (including burst loss), more replicas are generated when Replica Scheduling is not used. This is because, without scheduling replicas, burst loss leads to repeated losses of replicas, resulting in an overall increase in redundancy.

We use B to denote the burst size of loss during transmission. Burst size is just one attribute of the loss pattern. Intuitively, when $B \leq R$, the Replica Scheduler is not a mandatory measure that should be taken. However, when it meets $B > R$, ART depends on the Replica Scheduler to expand the escape space, i.e., $escape\_space \geq B$. This assures that at least one replica "escapes" the burst loss and is successfully delivered. On the other hand, all replicas should be sent in one RTT to ensure an orderly control loop. Based on these observations, we finally give the guideline of how to determine the $escape\_space$ as follows:

$$escape\_space = \min(bdp, \alpha \cdot B_{max}) \tag{6}$$

where bdp refers to the bandwidth and delay product, α refers to a scaling coefficient ($\alpha \geq 1$), and B_{max} refers to the maximum burst size of the loss in the past several RTTs (e.g., $5 \sim 10$ RTTs). In general, setting a large scaling coefficient α (e.g., $\alpha = 5$) enhances the tolerance to burst loss but may also induce recovery latency. In most cases, we set $\alpha = 1$.

6.3 Implementation

We implement ART in the user-space QUIC protocol based on LSQUIC commit *850b0a3* [36], consisting of 600+ lines of code. The open-source implementation of ART upon QUIC is maintained at [37]. ART only requires modifying the retransmission logic at the sender without interfering with other components such as congestion controllers.

For the Redundancy Adapter implementation, we reuse the function of send_ctl_handle_regular_lost_packet() to generate replicas with newly assigned packet numbers. These replicas are incorporated into the packet chain called po_loss_chain. To manage retransmission rounds effectively, we add two additional variables to the packet properties. Specifically, the variable po_retrans_times keeps track of the retransmission rounds, while po_retrans_packet_number records the packet number in each round. We also maintain a queue specifically designated to record the transmission states of replicas. When an ACK for a replica is received (detected by lsquic_send_ctl_got_ack()), or when the packet is identified as lost (detected by send_ctl_detect_losses()), the corresponding states in the queue are updated accordingly.

For the Replica Scheduler implementation, we reuse the alarm function of lsquic_alarmset_set() and add a new alarm AL_ART_SCHE in the alarm_set. We also include an attribute called po_expected_sent to record the expected time at which the next replica should be sent. When the AL_ART_SCHE alarm expires, the Replica Scheduler sends out the replicas according to their po_expected_sent. To determine the value of po_expected_sent, we randomly select a time interval less than escape_space, which is updated each retransmission round according to the max_burst_size.

6.4 Discussion

Adaptability to traffic patterns. From the perspective of traffic patterns, there are at least two types of traffic patterns in real networks. One is the application-limited traffic pattern, where the data-sending rate is constrained by the application itself, and the network is not the bottleneck. The other is the congestion-control-limited traffic pattern, where the data-sending rate is limited by the available network bandwidth. For the application-limited traffic pattern, our solution is undoubtedly effective because it utilizes excess network bandwidth to accelerate loss recovery. However, for the congestion-control-limited traffic pattern, replicas might not be sent if there is no available congestion window. To address the issues under the congestion-control-limited traffic pattern, in this paper, we introduce *opportunism*, which allows replicas to be sent on time even when the congestion window is insufficient. *Opportunism* ensures that during congestion, replicas are given higher priority over regular packets for transmission, thus effectively speeding up loss recovery.

Interaction with congestion controller. Many congestion control algorithms, such as Cubic and Reno, are based on loss signals. Generally, the loss of each packet triggers a loss signal, which is fed back to the congestion controller. In the context of ART, there are multiple replicas in each retransmission round. ART only passes a loss signal to the congestion controller if and only if all replicas in a round are lost. This means that if only some replicas in a round are lost, the congestion controller will not receive a loss signal. This approach minimizes the impact of redundancy on the congestion controller.

Calculation granularity for N. QUIC does not retransmit "packets" but rather retransmits "frames." Therefore, even though we add redundancy to packets, all stream frames within the packets must be retransmitted. This means that the maximum retransmission round for a lost packet is determined by the round in which the last frame within the packet is received. For example, suppose a lost packet (with packet number 18) contains two frames, Frame 1 and Frame 2. After being split by QUIC's packet splitting mechanism, during retransmission, these frames are scattered into packets with packets 50 and 60, respectively. In this case, the maximum retransmission round for packet 18 would be the greater of the retransmission rounds of packets 50 and 60.

7 Evaluation

In this section, we conducted experiments to investigate the performance of ART in both the testbed and production networks. We focused primarily on the following questions: (1) What are the advantages of ART when compared with existing technologies? (2) How does ART accelerate packet loss recovery for delay-sensitive transmissions? (3) How does ART accelerate packet loss recovery for throughput-intensive transmissions? (4) What is the cost of using ART? (5) How does ART work in practice?

7.1 Methodology

Experiment setup. In our evaluation, we utilized two distinct networks to comprehensively assess the performance of our approach, namely: (1) Trace-driven testbed network using the Mahimahi [38] simulation software. This versatile software allowed us to replicate real-world network conditions by replaying publicly available network traces. Burst loss is also simulated using Mahimahi's limited buffer. The credibility and widespread adoption in numerous research papers [16], [39] further solidify its suitability for our simulations. (2) Production network deployment involved the implementation of ART on a QUIC server, which was provided by the ByteDance CDN service. This deployment encompassed a diverse array of network links and user profiles, providing a comprehensive and practical assessment of our approach's effectiveness in real-world scenarios.

Schemes. Throughout our subsequent analyses, we compare ART with the following baselines:

- **QUIC**: The traditional QUIC (LSQUIC commit *850b0a3* [36]) without ART or FEC. By default, QUIC adopts the traditional ARQ, where it retransmits only one replica at a time when a packet is detected lost.
- **OR3**: The QUIC with OR3 [22], the pioneering approach on redundant data transmission. Note that OR3 adopts a variable redundancy level ($R = 2n-1$), where n denotes the n^{th} retransmission round.
- **FEC**: The QUIC with FEC, which adopts the block codes [40] as the error correcting codes. Note that FEC can be implemented not only with block-based schemes but also with random linear coding schemes. Here, we use the more widely applied block-based schemes for illustration. The block codes approach divides the packets into different blocks. The redundancy ratio of FEC is quantified as the proportion between the count of regular data packets and the count of redundant packets present in each block. For example, a redundancy ratio of 9:1 means 9 data packets are used to create 1 redundant packet. When one of these 10 packets in the block is lost, the 9 data packets can still be recovered.
- We also introduce some variants of QUIC. For example, QUIC ($R = 1$) refers to the QUIC that adopts a fixed redundancy level ($R = 1$), and QUIC ($R = 2n+1$) refers to the QUIC that adopts a variable redundancy level ($R = 2n + 1$), where n denotes the n^{th} retransmission round.

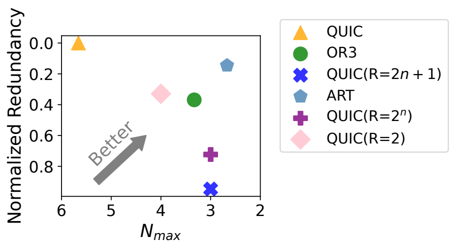


Fig. 19: Overall performance comparison between ART and other schemes. The network condition is set as bw = 100 Mbps and $p = 4\%$.

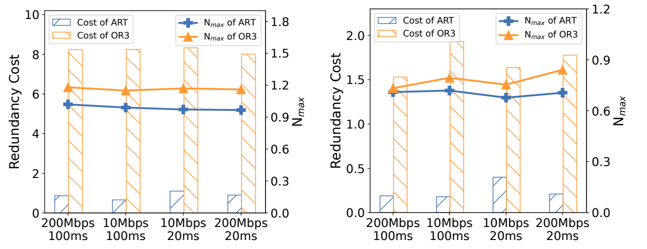
Traffic Patterns. In our evaluation, we measure various types of objects, encompassing both file transfers and RPC requests. File transfers have been subjected to evaluation in both testbed networks and production networks, whereas RPC requests have exclusively undergone production-network measurement. The size of RPC requests adheres to the actual body sizes observed on a real-world platform, ensuring the authenticity of our assessment. To ensure a comprehensive evaluation, we have included a diverse range of file sizes, spanning from tens of kilobytes to hundreds of megabytes, thus providing a thorough analysis of the system's performance across different data loads.

Metrics. In delay-sensitive scenarios, our attention is directed toward crucial metrics such as packet recovery time and FCT. Conversely, in throughput-extensive scenarios, our primary consideration centers around achieving optimal goodput. Furthermore, in both of these contexts, meticulous scrutiny is given to factors like the (maximum) retransmission rounds and redundancy cost. The former serves as a reflection of performance, while the latter provides insights into the associated overhead.

7.2 Overall Performance

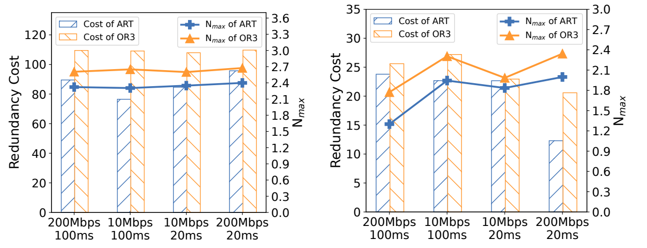
Comparison with QUIC and OR3. The performance is represented by the maximum retransmission round (N_{max}), and the cost is represented by the redundancy cost. We compare the performance and cost between ART and multiple QUIC variants by transferring 100MB files in the testbed with $bw = 100$ Mbps, $rtt = 30$ ms, and $p = 4\%$. Figure 19 shows the results. The findings from our investigation affirm the superiority of ART in multiple aspects. Notably, it boasts the smallest N_{max}, outperforming all the other QUIC variants. Additionally, ART exhibits a remarkable advantage by having less redundancy cost when juxtaposed with its counterparts. This demonstrates that ART can reduce the recovery time of lost packets without imposing significant redundancy costs.

Note that both OR3 and ART accelerate loss recovery; however, ART achieves better performance-cost efficiency. This is due to the adoption of the Redundancy Adaption in ART. To explain this more clearly, we further explore the differences between ART and OR3 under more types of network conditions. As shown in Figures 20 to 21, we run tests in the testbed with file sizes ranging from 200 KB to 1 MB and packet loss rates ranging from 1% to 10%. We observe that ART substantially reduces the redundancy costs, especially in scenarios with lower packet loss. For example in Figure 21, when $p = 10\%$, ART reduces the costs by $12\% \sim 40\%$. While in Figures 20, when $p = 1\%$, the reduction of costs of ART increases to $75\% \sim 90\%$. This is because ART is designed based on network feedback, adapting to the network conditions and requiring fewer replicas when the network performs well. In contrast, OR3 relies solely on the packets themselves and does not react to external network factors, leading to nearly constant redundancy levels across various packet loss rates.



(a) File size=1MB loss=1% (b) File size=200KB loss=1%

Fig. 20: (a) The average value of transmitting 1000 files of size 1MB each with a packet loss rate of 1%. (b) The average value of transmitting 1000 files of size 200KB each with a packet loss rate of 1%.



(a) File size=1MB loss=10% (b) File size=200KB loss=10%

Fig. 21: (a) The average value of transmitting 1000 files of size 1MB each with a packet loss rate of 10%. (b) The average value of transmitting 1000 files of size 200KB each with a packet loss rate of 10%.

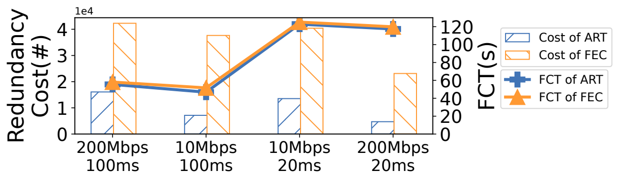
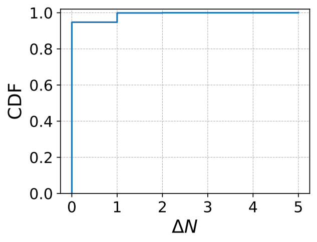
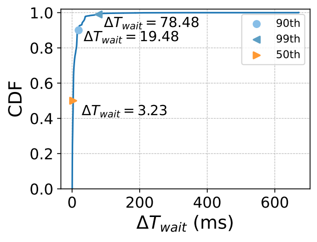


Fig. 22: Performance comparison between ART and FEC.

Comparison with FEC. Since ART is proposed as an ARQ-enhanced scheme that selectively incorporates FEC for lost packets, it is necessary to compare the performance and cost between ART and FEC. In this experiment, the redundancy ratio of FEC is set to 9:1 to assure a relatively low redundancy cost, and the network condition is set as $bw = 100$ Mbps, $rtt = 20$ ms, and $p = 4\%$. We investigate both the FCT as the performance and the redundancy cost as the cost. Figure 22 shows the results. It reveals that, on the whole, ART does not yield a noteworthy enhancement in FCT when juxtaposed with FEC. However, it significantly reduces the redundancy cost, minimizing the redundant packet occurrences in comparison to FEC. This demonstrates that ART can achieve a good trade-off between performance and cost.

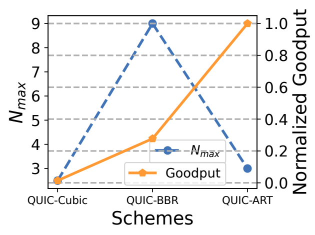


(a) ΔN

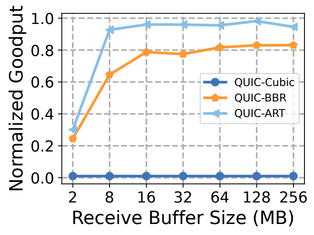


(b) ΔT_{wait}

Fig. 23: (a) The distribution of the reduction of retransmission rounds. (b) The distribution of the reduction of recovery latency.



(a) Goodput and N_{max}



(b) Goodput

Fig. 24: (a) An example of the relationship between N_{max} and goodput. (b) Performance in the case of receiving buffer starvation.

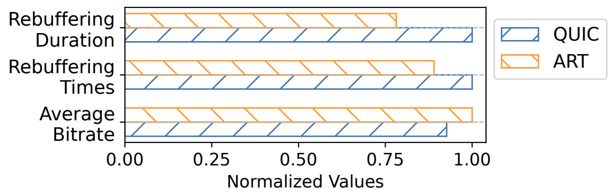


Fig. 25: QoE Comparison between ART and QUIC.

7.3 Mitigating Data Reassembling Starvation of Delay-Sensitive Transmissions

According to Equation (2), the delay-sensitive transmission suffers from data reassembling starvation in the case of a large N, the retransmission round of lost packets. To explore how ART tackles this issue, we first investigate how ART reduces N and then investigate how ART reduces the loss recovery latency T_{wait}. We have conducted a comprehensive series of measurements using the testbed to analyze the distributions of N and T_{wait} during the transmission of 100 MB files. Specifically, the packet loss rate is varied within the range of 1% to 10%. The bandwidth is varied within the range of 10 Mbps to 200 Mbps, while the RTT is set to span from 20 ms to 100 ms.

Reducing retransmission rounds of lost packets. We compute the reduction of the retransmission round (ΔN) as $\Delta N = N_{quic} - N_{art}$, where N_{quic} denotes the retransmission round of a packet in traditional QUIC without applying ART and N_{art} denotes the retransmission round of the packet in QUIC applying ART. As shown in Figure 23(a), we explore the distribution of ΔN by running tests with the traditional QUIC and ART. It is noteworthy that we have tried our best to make the measurement conditions similar for ART and QUIC in each test. It is observed that in most cases, ΔN keeps the value of 0; This is because most losses can be recovered via a single retransmission round. However, it also shows that ART may significantly reduce the retransmission rounds of lost packets, especially in the worst cases. For example, ΔN is up to 5 when the packets are excessively lost.

Reducing loss recovery latency. We compute the reduction of the loss recovery latency (ΔT_{wait}) as $\Delta T_{wait} = T^{quic}_{wait} - T^{art}_{wait}$, where T^{quic}_{wait} denotes the recovery latency of a packet in QUIC without applying ART and T^{art}_{wait} denotes the recovery latency of the packet in QUIC applying ART. As shown in Figure 23(b), we explore the distribution of ΔT_{wait} by testing the traditional QUIC and ART. It is observed that ART significantly reduces the recovery latency of lost packets. Specifically, the 50^{th}, 95^{th}, and 99^{th} percentile ΔT_{wait} are 3.23 ms, 32.83 ms, and 78.48 ms, respectively. Compared with the magnitude relative to RTT (i.e., $[20, 100]$ ms), this shows remarkable recovery latency reduction.

7.4 Mitigating Receiving Buffer Starvation of Throughput-Intensive Transmissions

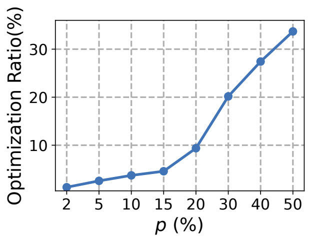
The throughput-intensive transmission suffers from receiving buffer starvation in the case of a large N_{max}, the maximum retransmission round among all lost packets. To explore how ART tackles this issue, we first investigate how ART reduces N_{max} and then investigate how ART improves the goodput.

Reducing maximum retransmission round. Figures 20 to 21 illustrate the average value of the maximum retransmission rounds under different network conditions. For example, in Figure 20, when $p = 1\%$, ART's reduction of N_{max} varies across different network environments, ranging from 3% to 16% for the transmission of 200KB-sized files and from 13% to 16% for the transmission of 1MB-sized files. Similarly, in Figure 21, when $p = 10\%$, ART's reduction of N_{max} varies across different network environments, ranging from 7% to 26% for 200KB-sized files and from 9% to 13% for 1MB-sized files. We can conclude that OR3 is already quite excellent in terms of the maximum retransmission rounds metric. However, our comparison experiments show that there is still room to reduce this metric.

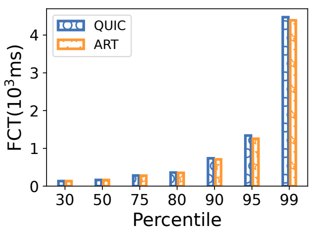
Improving goodput. As shown in Figure 24(a), we give an example of the relationship between N_{max} and goodput, where bw = 100 Mbps, rtt = 300 ms, p = 5%, and receiving buffer size = 8 MB. The average N_{max} is obtained from 100 runs of each scheme. It is demonstrated that ART greatly increases the goodput (orange line) by decreasing the N_{max} (blue dashed line). As mentioned in §4, a smaller N_{max} alleviates receiving buffer starvation. Figure 24(b) further shows the case of how ART performs under different receiving buffer sizes. It is observed that ART still fills up the pipe when the receiving buffer size is insufficient for QUIC. In particular, ART improves up to 28% of goodput when receiving buffer size = 8 MB. We believe that this can be attributed to ART's efficient loss recovery.

7.5 Improvement on QoE

In this experiment, we set up the video playback client using the DASH protocol [41]. Both the client and server utilize LSQUIC to transmit video chunks, and those associated with the same video are conveyed through a shared LSQUIC connection. Mahimahi [38] is employed to simulate



(a) Optimization ratio



(b) FCT ranges

Fig. 26: Real-world deployment. The optimization ratio is computed as $\frac{FCT_{quic} - FCT_{art}}{FCT_{quic}}$, where FCT_{quic} and FCT_{art} denote the FCT of QUIC and ART, respectively.

the network environment for transmission. The simulated network conditions are derived from Mahimahi's Verizon-LTE-driving trace file, incorporating a 30 ms RTT and a 10% packet loss rate. Furthermore, the DASH videos employed in our study were sourced from [42], with a duration of approximately 600 seconds, covering a bitrate spectrum from 44 Kbps to 3.9 Mbps. While there exist numerous bitrate adaptation algorithms for the DASH protocol [43], [44], this paper's primary focus is not on these algorithms but rather on the transmission protocol itself. Consequently, we chose to use the basic bitrate adaptation scheme that uses only the segment download rate for our assessments [44]. We compare the QoEs of video streaming when ART and QUIC are used as transport protocols, respectively. The QoE metrics include the average bitrate during video playback, rebuffering times, and rebuffering duration.

Figure 25 shows the results. It is demonstrated that ART outperforms QUIC by 7.9% in average bitrate, increasing from 254 Kbps to 274 Kbps. In terms of the times of video playback rebuffering, ART exhibits an 11.6% reduction compared to QUIC, decreasing from 82 to 73. Furthermore, concerning the duration of video playback rebuffering, ART demonstrates a 27.9% decrease compared to QUIC, decreasing from 322 s to 251 s. These improvements in QoE are attributed to the faster loss recovery of ART.

7.6 The CPU overhead of Using ART

ART is a lightweight packet loss recovery solution. It does not impose a large computational overhead on the CPU. To demonstrate this, we use two machines to act as the sender and the receiver. We set QUIC to run on a single core at the sender side. The CPU utilization of running QUIC, ART, and FEC on the sender side was counted separately, with the packet loss rate set to a constant value of 5% without limiting the bandwidth and delay. The use of ART brings about an additional CPU consumption of 0.18% and FEC brings about an additional consumption of 0.92% compared to the traditional QUIC (without applying any additional means of packet loss recovery). This reveals that the CPU overhead of using ART is negligible in most cases.

7.7 Real-world Deployment

To verify that ART indeed accelerates packet recovery speed, we used the actual FCT on real-network as our measurement metric. Particularly, we compute the optimization ratio as $\frac{FCT_{quic} - FCT_{art}}{FCT_{quic}}$, where FCT_{quic} denotes the FCT of QUIC without applying ART and FCT_{art} denotes the FCT of QUIC applying ART.

Setup. We conducted comprehensive A/B testing of two key scenarios: RPC transmission and small file downloading. In the RPC transmission scenario, we analyzed data from 3.3 million QUIC connections spanning an entire day on a 4G Mobile Network. This data provided valuable insights into the performance and efficiency of the RPC requests. For the small file transfer scenario, where each file size was approximately 3MB, we collected real-network data from users located in diverse geographical areas. The data was gathered using the bonree [45] platform and comprised over 26,000 requests spanning a week. Notably, these clients generated transfer requests at an impressive rate of 100 times per second.

Result. Figure 26(a) presents the average FCT improvement proportion of ART over native QUIC in an RPC transmission scenario under varying packet loss rates. The graph demonstrates that the optimization effect becomes more pronounced as the packet loss rate increases, with improvements reaching approximately 34%. This phenomenon can be attributed to the diminished effectiveness of the loss recovery algorithm at higher packet loss rates, and ART effectively compensates for this deficiency. Figure 26(b) provides a comparison of FCT across different quantiles. Additionally, in the small file downloading scenario, we observed an overall reduction in FCT of about 2%. Our deployment experience further shows that this reduction is especially significant for users with higher access bandwidths.

8 Further Discussions: Whether Reinforcement-Learning-based Redundancy Adapters Work Better?

By default, the Redundancy Adapter in ART uses the test-and-verification method to determine the redundancy level upon retransmissions. However, it is also interesting to answer the question of whether applying machine-learning-based Redundancy Adapters works better. Recognizing the absence of a universally defined "ground truth" for the retransmission problem, we have opted against employing supervised learning. Anticipating the next action according to detailed previous records of network state is a promising approach. Reinforcement learning (RL) stands out as a widely embraced machine learning technique ideally suited for our requirements. We used two typical reinforcement learning methods: multi-armed bandit and DQN (Deep Q-Network), which were referred to as ART-MAB and ART-DQN in the subsequent sections. ART-MAB includes the hyperparameter η, which represents the penalty factor, while ART-DQN includes the hyperparameter γ, which represents the magnitude ratio between rewards and penalties. For details on the design and implementation of these two methods, please refer to Appendix A.

8.1 Experimental Comparison

Experiment setup. We primarily evaluated the transmission performance among ART, ART-DQN, and ART-MAB in the testbed. Our testbed simulates the network environment using Mahimahi. We use reinforcement learning to determine the optimal redundancy level under varying packet loss patterns. To isolate the effects of bandwidth and RTT, we keep these conditions constant across experiments

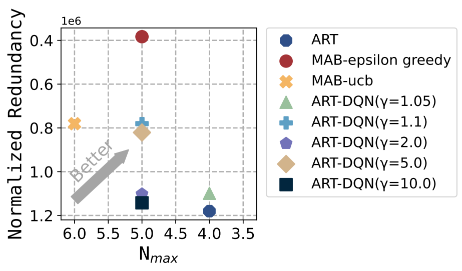


Fig. 27: Performance and redundancy cost comparison.

while systematically varying packet loss rates. Files were transmitted using these three algorithms under the same trace (Verizon-LTE in Mahimahi), the same delay (30 ms), and different packet loss rates (1%-10%). We transmitted both large and small files, with large files being around 200MB and small files around 1 MB. Each experiment lasted approximately 4 hours. In our experiments, we set $\eta = 1.01$.

Convergence time. To mitigate the impact of limited data on the model's performance, we compared various algorithms under consistent network conditions. The runtime for the multi-armed bandit algorithm exceeded 9 hours, and the training time for DQN surpassed 2 days.

Overall performance. Examining Figure 27, it is evident that the values of N_{max} for both the MAB-EG and the MAB-UCB surpass those of ART. In the case of ART-DQN with varying γ parameters, N_{max} aligns with ART only when γ is set to 1.05; otherwise, it exceeds the value observed for ART. Figure 27 also demonstrates that the MAB-EG achieves the lowest redundancy cost, whereas ART displays the highest redundancy cost. When considering the same N_{max}, ART-DQN's redundancy cost is only lower than ART when the γ parameter is set to 1.05, resulting in a reduction of 7.4%.

ART's performance with the test-and-verification method is slightly lower than ART-MAB, yet the gap is not considerable. It outperforms ART-DQN, albeit the enhancement is minimal. Please refer to Appendix B for more detailed experiments. Overall, the disparities in performance among these three approaches are minor.

CPU Overhead. We conducted CPU overhead measurements for ART, ART-MAB, and ART-DQN within a network simulation using the Mahimahi tool, factoring in both server and client components. Specifically, we utilized a trace named Verizon-LTE-driving, featuring a delay of 30ms and a packet loss rate of 10%. Our findings indicate that adopting ART results in an additional CPU overhead of 0.68% while employing ART-MAB incurs an extra 0.65% CPU overhead. Conversely, the implementation of ART-DQN introduces a significantly higher overhead of 54.64%, with model execution alone accounting for 53.69% of the additional overhead. This indicates that the CPU overhead of ART and ART-MAB are quite similar, while ART-DQN incurs significantly higher overhead compared to both of them.

Conclusion. The performance of the multi-armed bandit is slightly less favorable than ART, be it in terms of fine-grained packet recovery time or coarse-grained N_{max}. Nevertheless, the redundancy is noticeably reduced compared to ART. As for DQN, N_{max} under various hyperparameters can occasionally match ART, while the overall packet recovery time is slightly inferior to both ART and the multi-armed bandit. In summary, RL-based Redundancy Adapters work similarly to the test-and-verification Redundancy Adapters from the perspective of performance and redundancy cost. However, the test-and-verification method significantly outperforms the RL-based one from the perspective of convergence time and CPU overhead. This reveals that the original ART with the test-and-verification method works better than expected and seems more practical for production network deployment.

9 Conclusion

Retransmission itself does not hinder slow recovery, it is the loss of retransmission that becomes the real obstacle. Simple but useful, ART is essentially proposed as an ARQ-enhanced scheme that selectively incorporates FEC for lost packets without any modifications to the receiver/client side. The primary goal of ART is to address data reassembling starvation and receiving buffer starvation by considerably reducing the recovery time of lost packets without imposing significant redundancy costs. The real-world deployment experience reaffirms the viability of ART as an advantageous option for CDN vendors seeking to enhance their competitiveness in a diverse and competitive market landscape with multiple suppliers.

References

[1] M. Rudow, F. Y. Yan, A. Kumar, G. Ananthanarayanan, M. Ellis, and K. Rashmi, "Tambur: Efficient loss recovery for videoconferencing via streaming codes," in *USENIX NSDI*, 2023, pp. 953–971.
[2] B. S. Bakshi, P. Krishna, N. H. Vaidya, and D. K. Pradhan, "Improving performance of TCP over wireless networks," in *IEEE ICDCS*, 1997, pp. 365–373.
[3] J. Sarkar, S. Sengupta, M. Chatterjee, and S. Ganguly, "Differential fec and arq for radio link protocols," *IEEE Transactions on Computers*, vol. 55, no. 11, pp. 1458–1472, 2006.
[4] F. Michel, Q. De Coninck, and O. Bonaventure, "QUIC-FEC: Bringing the benefits of Forward Erasure Correction to QUIC," in *IEEE IFIP Networking*, 2019, pp. 1–9.
[5] Y. Arda and J.-C. Hennet, "Inventory control in a multi-supplier system," *IJPE*, vol. 104, no. 2, pp. 249–259, 2006.
[6] K. Park and W. Wang, "AFEC: an adaptive forward error correction protocol for end-to-end transport of real-time traffic," in *IEEE ICCCN*, 1998, pp. 196–205.
[7] M. Koul and K. Rao, "An N+1 redundant GOP based FEC algorithm for improving Streaming Video Quality due to Packet Loss and Delay Jitter," in *ICET*, 2007, pp. 102–107.
[8] H. Lundqvist and G. Karlsson, "TCP with end-to-end FEC," in *IZS*, 2004, pp. 152–155.
[9] E. Blanton, M. Allman, L. Wang, I. Jarvinen, M. Kojo, and Y. Nishida, "RFC 6675: A conservative loss recovery algorithm based on selective acknowledgment (SACK) for TCP," *IETF*, 2012.
[10] E. Blanton, "RFC 4653: Improving the robustness of TCP to non-congestion events," *IETF*, 2006.
[11] M. Allman, K. Avrachenkov, U. Ayesta, J. Blanton, and P. Hurtig, "RFC 5827: Early retransmit for TCP and stream control transmission protocol (SCTP)," *IETF*, 2010.
[12] P. Hurtig, A. Brunstrom, A. Petlund, and M. Welzl, "RFC 7765: TCP and stream control transmission protocol (SCTP) RTO restart," *IETF*, 2016.
[13] M. Mathis and J. Mahdavi, "Forward acknowledgement: refining TCP congestion control," *Acm Sigcomm Computer Communication Review*, vol. 26, no. 4, pp. 281–291, 1996.
[14] Y. Cheng, N. Cardwell, N. Dukkipati, and P. Jha, "The RACK-TLP Loss Detection Algorithm for TCP," RFC 8985, Feb. 2021. [Online]. Available: https://www.rfc-editor.org/info/rfc8985
[15] J. Iyengar and M. Thomson, "QUIC: A UDP-Based Multiplexed and Secure Transport," RFC 9000, May 2021. [Online]. Available: https://www.rfc-editor.org/info/rfc9000
[16] T. Li, K. Zheng, K. Xu, R. A. Jadhav, T. Xiong, K. Winstein, and K. Tan, "TACK: Improving Wireless Transport Performance by Taming Acknowledgments," in *ACM SIGCOMM*, 2020, pp. 15–30.

[17] H. Wen, C. Lin, F. Ren, Y. Yue, and X. Huang, "Retransmission or Redundancy: Transmission Reliability in Wireless Sensor Networks," in *IEEE MASS*, 2007, pp. 1–7.

[18] H. Wen, C. Lin, F. Ren, H. Yang, T. He, and E. Dutkiewicz, "Joint Adaptive Redundancy and Partial Retransmission for Reliable Transmission in Wireless Sensor Networks," in *IEEE IPCCC*, 2008, pp. 303–310.

[19] J. Zhu and S. Roy, "An adaptive two-copy delayed SR-ARQ for satellite channels with shadowing," in *IEEE VTC*, 2002, pp. 849–853.

[20] K. Liu and J. Y. Lee, "Improving TCP performance over mobile data networks with opportunistic retransmission," in *IEEE WCNC*, 2013, pp. 1992–1997.

[21] Z. Meng, X. Kong, J. Chen, B. Wang, M. Xu, R. Han, H. Liu, V. Arun, H. Hu, and X. Wei, "Hairpin: Rethinking packet loss recovery in edge-based interactive video streaming," in *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*. Santa Clara, CA: USENIX Association, Apr. 2024, pp. 907–926. [Online]. Available: https://www.usenix.org/conference/nsdi24/presentation/meng

[22] H. Xie and T. Li, "Revisiting Loss Recovery for High-Speed Transmission," in *IEEE WCNC*, 2022, pp. 1987–1992.

[23] J. Zou, T. Hao, C. Yu, and H. Jin, "A3c-do: A regional resource scheduling framework based on deep reinforcement learning in edge scenario," *IEEE Transactions on Computers*, vol. 70, no. 2, pp. 228–239, 2021.

[24] J. Wang, J. Hu, G. Min, W. Zhan, A. Y. Zomaya, and N. Georgalas, "Dependent task offloading for edge computing based on deep reinforcement learning," *IEEE Transactions on Computers*, vol. 71, no. 10, pp. 2449–2461, 2022.

[25] B. Mao, Z. M. Fadlullah, F. Tang, N. Kato, O. Akashi, T. Inoue, and K. Mizutani, "Routing or computing? the paradigm shift towards intelligent computer network packet transmission based on deep learning," *IEEE Transactions on Computers*, vol. 66, no. 11, pp. 1946–1960, 2017.

[26] X. Nie, Y. Zhao, Z. Li, G. Chen, K. Sui, J. Zhang, Z. Ye, and D. Pei, "Dynamic TCP Initial Windows and Congestion Control Schemes Through Reinforcement Learning," *IEEE Journal on Selected Areas in Communications*, vol. 37, no. 6, pp. 1231–1247, 2019.

[27] Y.-R. Chen, A. Rezapour, W.-G. Tzeng, and S.-C. Tsai, "RL-Routing: An SDN Routing Algorithm Based on Deep Reinforcement Learning," *IEEE Transactions on Network Science and Engineering*, vol. 7, no. 4, pp. 3185–3199, 2020.

[28] T. Huang, X. Lu, D. Zhang, H. Cheng, P. Dong, and L. Zhang, "ACC-RL: Adaptive Congestion Control Based on Reinforcement Learning in Power Distribution Networks with Data Centers," *Energies*, vol. 16, no. 14, 2023.

[29] N. Cardwell, Y. Cheng, C. S. Gunn, S. H. Yeganeh, and V. Jacobson, "BBR: Congestion-Based Congestion Control," *Commun. ACM*, vol. 60, no. 2, p. 58–66, 2017.

[30] K. Xie, X. Wang, X. Liu, J. Wen, and J. Cao, "Interference-aware cooperative communication in multi-radio multi-channel wireless networks," *IEEE Transactions on Computers*, vol. 65, no. 5, pp. 1528–1542, 2016.

[31] H. Hu, S. Cheng, X. Zhang, and Z. Guo, "LightFEC: Network Adaptive FEC with a Lightweight Deep-Learning Approach," in *ACM MM*, 2021, pp. 3592–3600.

[32] A. Cohen, G. Thiran, V. B. Bracha, and M. Médard, "Adaptive Causal Network Coding With Feedback for Multipath Multi-Hop Communications," *IEEE Transactions on Communications*, vol. 69, no. 2, pp. 766–785, 2021.

[33] F. Michel, "Flexible QUIC loss recovery mechanisms for latency-sensitive applications," Bonaventure, Olivier, 2023.

[34] Z. Wen and K. L. Yeung, "On Detection Algorithms for Spurious Retransmissions in TCP," in *IEEE WCNC*, 2010, pp. 1–6.

[35] F. Y. Yan, J. Ma, G. D. Hill, D. Raghavan, R. S. Wahby, P. Levis, and K. Winstein, "Pantheon: the training ground for Internet congestion-control research," in *USENIX ATC 18*, 2018, pp. 731–743.

[36] L. Tech, "LSQUIC," https://github.com/litespeedtech/lsquic/commit/850b0a3d100b2abe89cf054a9f8d13054fac34a3, 2022.

[37] LitongLab, "QUIC-ART," https://github.com/litonglab/quic-art, 2023.

[38] R. Netravali, A. Sivaraman, K. Winstein, S. Das, A. Goyal, and H. Balakrishnan, "Mahimahi: A Lightweight Toolkit for Reproducible Web Measurement," *SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 4, p. 129–130, 2014.

[39] V. Arun and H. Balakrishnan, "Copa: Practical Delay-Based Congestion Control for the Internet," in *ANRW*, 2018, p. 19.

[40] I. S. Reed and G. Solomon, "Polynomial codes over certain finite fields," *SIAM*, vol. 8, no. 2, pp. 300–304, 1960.

[41] S. Arisu and A. C. Begen, "Quickly Starting Media Streams Using QUIC," in *Proceedings of the 23rd Packet Video Workshop*, 2018, p. 1–6.

[42] "DASH Dataset," http://www-itec.uni-klu.ac.at/ftp/datasets/DASHDataset2014.

[43] T.-Y. Huang, R. Johari, N. McKeown, M. Trunnell, and M. Watson, "A buffer-based approach to rate adaptation: evidence from a large video streaming service," in *SIGCOMM '14*, 2014, p. 187–198.

[44] P. Juluri, V. Tamarapalli, and D. Medhi, "SARA: Segment aware rate adaptation algorithm for dynamic adaptive streaming over HTTP," in *2015 IEEE International Conference on Communication Workshop (ICCW)*, 2015, pp. 1765–1770.

[45] bonree, "Bonree," https://www.bonree.com, 2008.

BIOGRAPHIES

Tong Li is currently an associate professor at Renmin University of China. His research interests include network protocols, service computing, and distributed systems.

Wei Liu is currently pursuing his master's degree at Renmin University of China. His research interests are network protocols, service computing, and wide-area networks.

Xinyu Ma is currently pursuing her master's degree at Renmin University of China. Her research interests are computer networks and network protocols.

Shuaipeng Zhu is currently pursuing a master's degree at Renmin University of China. His research interests include network protocol and congestion control.

Jingkun Cao is currently pursuing her B.S.Eng degree at Renmin University of China. Her research interests include network protocols and machine learning.

Duling Xu is currently pursuing her Ph.D. degree at Renmin University of China. Her research interests include network protocols and distributed systems.

Zhaoqi Yang is currently pursuing his B.S.Eng degree at Renmin University of China. His research interests include wireless sensing and blockchain.

Senzhen Liu is a R&D engineer in Bytedance. His research interests include network protocols and congestion control.

Taotao Zhang is a R&D engineer in Bytedance. His research interests include network protocols and congestion control.

Yinfeng Zhu is a R&D engineer in Bytedance. His research interests include network protocols and congestion control.

Bo Wu received his Ph.D. degree from Tsinghua University, Beijing, China. His research areas include next-generation Internet, network security, and network transport.

Kezhi Wang is a Professor with the Department of Computer Science, Brunel University of London, U.K. His research interests include wireless communications, mobile edge computing, and machine learning.

Ke Xu (Fellow, IEEE) received the Ph.D. degree from Tsinghua University, Beijing, China. He is currently a Full Professor with the Department of Computer Science and Technology, Tsinghua University. He has published more than 200 technical articles in the research areas of next-generation internet, blockchain systems, and network security. He won the IWQoS 24 Best Paper Award and the Distinguished Paper Award at USENIX Security 23/24 and NDSS 25.

Appendix A
Design and implementation of reinforcement-learning-based Redundancy Adapters

We have developed two types of reinforcement-learning-based Redundancy Adapters: the Bandit-based Redundancy Adapter and the DQN-based Redundancy Adapter, where DQN refers to Deep Q-Network [1].

A.1 Method #1: Bandit-based Redundancy Adapter

The multi-armed bandit stands out as a straightforward algorithm in the realm of reinforcement learning, aiming to achieve a delicate equilibrium between exploration and exploitation. Notably, [2] has effectively incorporated it into congestion control strategies.

Design. We formalize the decision-making problem for redundancy levels as a multi-armed bandit problem, with various algorithms proposed for this context [3]. We refer to the Redundancy Adapter incorporating the use of a multi-armed bandit as Bandit-based Redundancy Adapter, and we refer to ART that utilizes the Bandit-based Redundancy Adapter as ART-MAB in the following sections. Upon encountering a packet loss event, the Bandit-based Redundancy Adapter will pull an arm based on a specific strategy, where each arm corresponds to a distinct redundancy level. This paper focuses on two strategies: the straightforward Epsilon greedy strategy and the empirically proven by [4] effective UCB (Upper Confidence Bound) strategy [5], which we respectively refer to as MAB-EG and MAB-UCB in the following sections. In the Epsilon greedy strategy, setting the ϵ variable to 0.1 means that, with a probability of 0.1, a random action will be chosen during decision-making to encourage exploration.

Our objective is to maximize returns while minimizing costs. In the realm of packet loss recovery, the primary benefit lies in reducing the recovery time of packets. However, this recovery time is significantly impacted by factors like round-trip time (RTT) and queuing delay. Consequently, we equate packet recovery time to the number of retransmission rounds, recognizing that fewer rounds result in greater returns. Conversely, the decision's cost is manifest in the number of replicas, representing the redundancy level. Higher redundancy levels incur higher costs. Furthermore, the previously mentioned reward calculation scenario presupposes that the redundant packets are received. In other cases, when redundant packets are detected as lost, the prior decision should be punished. Now, we can formulate our reward function of multi-armed bandit:

$$reward_{MAB} = \begin{cases} \dfrac{1}{N * R} \quad , & \text{if replicas received} \\ -\dfrac{\eta}{N * R} \quad , & \text{if replicas unreceived} \end{cases}$$

where N denotes the retransmission round of a packet and R denotes the redundancy level. The η represents the penalty factor, typically greater than 1.

Implementation. We implement a multi-armed bandit within the LSQUIC codebase, introducing two key functions: select_arm_with_policy() and calc_multi_armed_bandit_reward(). Moreover, the bandit's arms are housed in the lsquic_send_ctl structure, designated as sc_all_arms. Each arm is defined by the arm_of_bandit structure, encompassing two vital variables: use_number denotes the frequency of arm usage, and expect signifies the anticipated value of the arm. Upon the reception of an ACK on the sender side, the calc_multi_armed_bandit_reward() function is triggered to compute rewards on a round-by-round basis. Rewards are computed for the rounds that eventually reach acknowledgment, while any remaining lost rounds incur penalties. Subsequently, the use_number and expect variables for each arm are updated accordingly for each round of usage.

A.2 Method #2: DQN-based Redundancy Adapter

Due to its broad applicability and notably effective performance within discrete action spaces, we decided to employ DQN as another representative reinforcement learning algorithm.

Design. We leverage DQN [6] to determine the optimal redundancy level, which we refer to as DQN-based Redundancy Adapter, and we refer to ART that utilizes the DQN-based Redundancy Adapter as ART-DQN in the following sections. Two key challenges emerge: managing the interaction between the other components of ART and the DQN-based Redundancy Adapter, and striking a balance between the execution speeds of ART-DQN and DQN.

Addressing the first challenge, our solution employs a memory-mapped interaction between the other components of ART and the DQN-based Redundancy Adapter, with three mapped files: action, state, and records. The action file, written by DQN-based Redundancy Adapter, encapsulates neural network decision outcomes, which ART-DQN consults when determining redundancy level. The state file, generated by ART-DQN, conveys the current network status, serving as input for DQN's neural network. The records file acts as training data for the DQN model, aligning with the experience replay module. ART-DQN writes to this file, and DQN reads from it. Assuming DQN's decision, denoted as action a, based on the input network state $s1$, ART-DQN transmits redundant packets. Subsequently, it receives results one RTT later, records the current network state as $s2$, and computes the corresponding reward r based on the reception or loss of redundant packets. This information is then recorded in the records file in the format:$(s1, a, r, s2)$, and we designate it as a singular record.

Addressing the second challenge, our solution avoids seeking fine-grained decisions by the neural network for each packet's redundancy. Instead, the redundancy decision is influenced more by the dynamic nature of the network. Given that network states undergo minimal changes over short periods, state and action information between DQN and ART-DQN is transmitted not on a per-packet basis but on a per-connection basis.

The reward function for DQN draws inspiration from the design of multi-armed bandits' reward functions (see §A.1). Recognizing the impact of hyperparameters on neural network training effectiveness, we introduced an additional hyperparameter γ, whose purpose is to regulate the magnitude ratio between rewards and penalties. Consequently, the reward function of DQN is defined as follows:

$$reward_{DQN} = \begin{cases} \dfrac{1}{N * R} \quad , & \text{if replicas received} \\ -\dfrac{\gamma}{N * R} \quad , & \text{if replicas unreceived} \end{cases}$$

where N denotes the retransmission round of a packet, and R denotes the redundancy level.

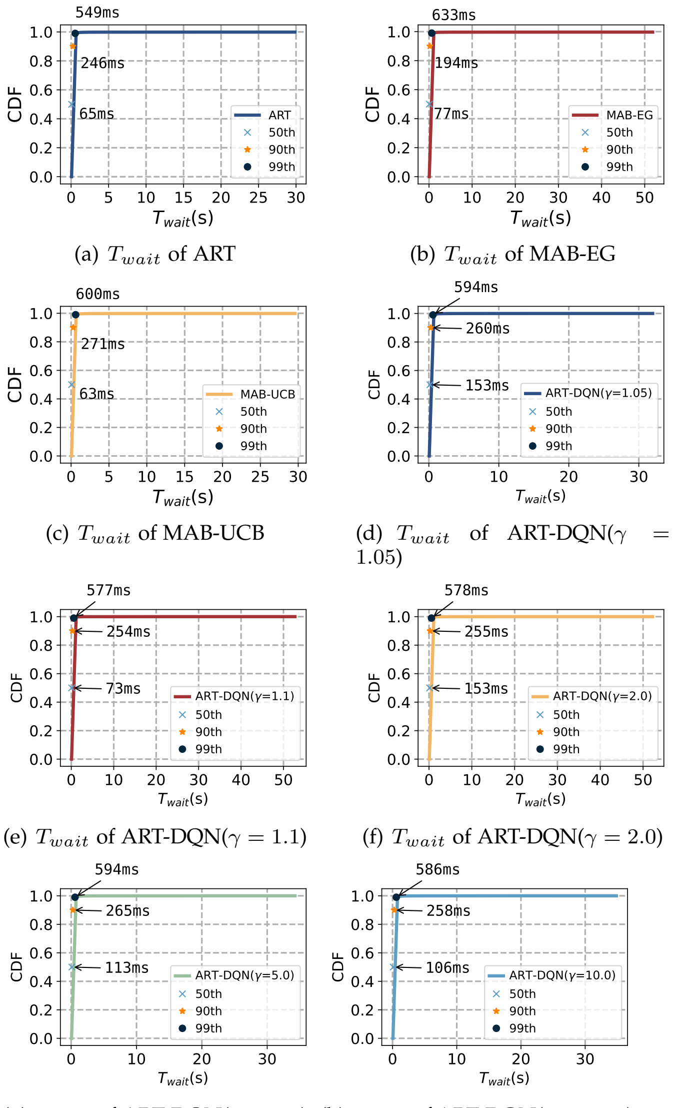
Implementation. Within the LSQUIC code, we introduced three interface functions: get_action(), insert_records_to_shm(), and write_state_to_shm(). These interfaces are tasked with retrieving actions, inserting record into the replay buffer for DQN model learning, and conveying network states as inputs to the DQN model, respectively. In implementing the DQN model, we drew inspiration from open-source code [6], adapting both the interaction and experience replay modules. Within the interaction module, two interfaces, sample() and write_action(), were incorporated to gather network states as inputs for the neural network and record neural network output results for LSQUIC to determine redundancy levels, respectively. In the experience replay module, the collect_memory() interface was introduced to refresh the replay buffer utilized in DQN training.

Appendix B
Performance of Reinforcement-Learning-Based Redundancy Adapters

Based on Figure 1(a), Figure 1(b) and Figure 1(c), we observe that the 50th percentile packet recovery times for ART, the MAB-EG, and the MAB-UCB are 65 ms, 77 ms, and 63 ms, respectively. The 99^{th} percentile values are 549 ms, 633 ms, and 600 ms, respectively. At the 50th percentile, the MAB-UCB demonstrates a 3% improvement over ART, while at the 99^{th} percentile, the MAB-EG and the MAB-UCB show reductions of 15% and 9%, respectively, compared to ART.

In Figures 1(d), 1(e), 1(f), 1(g), and 1(h), the 50^{th} percentile of packet recovery time is 153 ms, 73 ms, 153 ms, 113 ms, and 106 ms, respectively. From Figure 1(d) to Figure 1(h), we could observe that while different γ parameters influence packet recovery time, experimental results across various γ values indicate that, in the optimal scenario with $\gamma = 1.1$, packet recovery times are inferior to ART. Specifically, at the 50^{th} percentile and 99^{th} percentile, they decrease by 12% and 5%, respectively.

Overall, RL-based Redundancy Adapters do not show significant performance improvements compared to test-and-verification Redundancy Adapters. On the contrary, they may introduce some uncertainty in the results due to the incorporation of machine learning [7].



(a) T_{wait} of ART (b) T_{wait} of MAB-EG

(c) T_{wait} of MAB-UCB (d) T_{wait} of ART-DQN($\gamma = 1.05$)

(e) T_{wait} of ART-DQN($\gamma = 1.1$) (f) T_{wait} of ART-DQN($\gamma = 2.0$)

(g) T_{wait} of ART-DQN($\gamma = 5.0$) (h) T_{wait} of ART-DQN($\gamma = 10.0$)

Fig. 1: Performance comparison between ART, bandit-based ART and DQN-based ART.

References

[1] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing Atari with Deep Reinforcement Learning," 2013.
[2] X. Nie, Y. Zhao, Z. Li, G. Chen, K. Sui, J. Zhang, Z. Ye, and D. Pei, "Dynamic TCP Initial Windows and Congestion Control Schemes Through Reinforcement Learning," *IEEE Journal on Selected Areas in Communications*, vol. 37, no. 6, pp. 1231–1247, 2019.
[3] R. Sutton and A. Barto, "Reinforcement Learning: An Introduction," *IEEE Transactions on Neural Networks*, vol. 9, no. 5, pp. 1054–1054, 1998.
[4] A. Garivier and E. Moulines, "On Upper-Confidence Bound Policies for Non-Stationary Bandit Problems," 2008.
[5] P. Auer, N. Cesa-Bianchi, and P. Fischer, "Finite-time Analysis of the Multiarmed Bandit Problem," *Machine Learning*, vol. 47, pp. 235–256, 05 2002.
[6] MorvanZhou, "Python-DQN," https://github.com/MorvanZhou/Reinforcement-learning-with-tensorflow, 2021.
[7] M. Kläs and A. M. Vollmer, "Uncertainty in machine learning applications: A practice-driven classification of uncertainty," in *Computer Safety, Reliability, and Security*, B. Gallina, A. Skavhaug, E. Schoitsch, and F. Bitsch, Eds. Cham: Springer International Publishing, 2018, pp. 431–438.