Hairpin: Rethinking Packet Loss Recovery in Edge-based Interactive Video Streaming

Zili Meng[1,2,3], Xiao Kong[1,3], Jing Chen[1,3], Bo Wang[1], Mingwei Xu[1],
Rui Han[3], Honghao Liu[3], Venkat Arun[4], Hongxin Hu[5], Xue Wei[3]
[1]*Tsinghua University,* [2]*Hong Kong University of Science and Technology,*
[3]*Tencent,* [4]*UT Austin,* [5]*University at Buffalo, SUNY*

Abstract

Interactive streaming requires minimizing stuttering events (or deadline misses for video frames) to ensure seamless interaction between users and applications. However, existing packet loss recovery mechanisms uniformly optimize redundancy for initial transmission and retransmission, which still could not satisfy the delay requirements of interactive streaming, but also introduces considerable bandwidth costs. Our insight is that in edge-based interactive streaming, differentiating retransmissions on redundancy settings can often achieve a low bandwidth cost and a low deadline miss rate simultaneously. In this paper, we propose Hairpin, a new packet loss recovery mechanism for edge-based interactive streaming. Hairpin finds the optimal combination of data packets, retransmissions, and redundant packets over multiple rounds of transmissions, which significantly reduces the bandwidth cost while ensuring the end-to-end latency requirement. Experiments with production deployments demonstrate that Hairpin can simultaneously reduce the bandwidth cost by 40% and the deadline miss rate by 32% on average in the wild against state-of-the-art solutions.

1 Introduction

Edge-based interactive video streaming is coming to age. Emerging interactive streaming services, such as cloud gaming, has been attracting considerable attention from both academic [24, 28, 44, 53, 63] and industrial communities [3, 8, 9]. These services run applications (e.g., video games) on remote servers, stream the contents (e.g., screens) to users, and interact with users in real time. To provide a seamless interaction between servers and users, the streaming needs to be delivered with low latency. To reduce the network round-trip time (RTT) for interactive streaming, operators deploy servers on edge computing nodes. For example, cloud gaming applications render gaming scenes on remote servers and deliver these scenes to users over the Internet, where these servers could be deployed near the users to shorten the physical distance of the streaming. By deploying interactive streaming services at the edge, the *median* network RTT of these edge-based applications could be reduced to 10-20ms.

To provide a satisfactory user experience, controlling *the ratio of frames violating the latency requirement* is critical for interactive streaming applications [56]. This is due to the continuous interactions between users and applications: Interactive streaming users will start to experience stutters when the end-to-end delay is above 50-200ms (i.e., the deadline of delivering a frame). However, even if only 0.1% of the frames stutter, users may experience stutters every one minute at a frame rate of 24fps. Thus, to improve user experience, we need to reduce the ratio of deadline misses in an interactive streaming service [57].

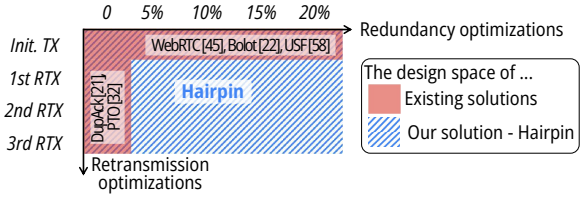


Figure 1: An illustration of the design space of existing solutions and Hairpin. By co-designing the redundancy and retransmission at the transport layer, Hairpin is able to break the existing trade-off between bandwidth cost and deadline miss rate.

In this case, a major challenge to control the deadline misses comes from the high instantaneous loss rate on the Internet. Due to the spatial dependency within video frames and temporal dependency between video frames, interactive streaming expects packets to be reliably delivered [57]. However, from our measurement of our edge-based cloud gaming service in production with O(10,000) users, sessions can experience a drastically high *instantaneous* loss rate. Although the average loss rate is considerably low by mechanisms such as proper rate control, our measurement observes that more than 2% of video frames suffer from an instantaneous loss rate of 20% or higher (§2.1). It indicates that those lost packets are concentrated on a few frames. Thus, although the network RTT can be very low with edge deployments, retransmissions of lost packets take additional time and will consequently violate the deadline. Thus, it is essential to optimize the loss recovery mechanisms to control the deadline miss rate (DMR) of video frames.

Unfortunately, existing solutions to recover packet losses cannot meet the stringent DMR requirements with a reasonable bandwidth cost. As shown in Figure 1, one line of research efforts (the vertical dimension) is devoted to quickly retransmitting lost packets, such as probe timeout (PTO) [33], from the transport layer. However, merely retransmitting lost packets cannot meet the requirement of interactive streaming – the DMR is much higher than 0.1% (§4.3). Another line of effort (the horizontal dimension) is devoted to adaptive forward error correction (FEC) so that the client might be able to recover packets based on redundant packets without retransmission [5]. Yet, redundancy-based solutions come with the price of a considerable bandwidth cost of 20% or more due to the high instantaneous loss rate. For content providers, such a high bandwidth cost will drastically increase operating expenses and

degrade users' video quality. To the best of our knowledge, none of the existing solutions *jointly optimized* retransmission and redundancy. Such an orthogonal design of redundancy and retransmission, even when adopted together, still cannot meet the needs of bandwidth cost and DMR for interactive streaming.

Our key insight is to break the trade-off by differentiating retransmission packets. Edge-based interactive streaming services can achieve an average RTT of 10-20ms between application servers and users by deploying the servers on the edge [27, 58, 89]. In this case, limited times of retransmissions (but not too many) are tolerable for applications that have a deadline of 50-200ms (§2.1). But the strategy for retransmission packets must be different for the initial transmission packets. The volume of retransmission packets is much less than initial transmission packets since packet loss is always the minority. Yet, retransmission packets have a much tighter time requirement since they have already consumed time. This brings new changes to reduce the bandwidth cost and the DMR at the same time (§2.4). By differentiating the strategies for initial transmission and retransmission packets, we can break the trade-off between bandwidth cost and DMR, and improve the performance significantly (§4.4).

We then propose Hairpin[1], a new packet loss recovery mechanism to jointly optimize packet retransmission and redundancy for edge-based interactive streaming (§3.3). However, as later elaborated in §3.2, to further analytically optimize the performance, we still face the challenge of (1) the dependency of decisions and future states, (2) the multi-dimensionality of decisions, and (3) the convoluted goal of DMR and bandwidth cost. In response, Hairpin further formulates the problem into a Markov decision process (MDP), which is known for efficiently optimizing the temporal dependency [79]. We then encodes the decisions and states into nodes of MDP to reduce the complexity and achieve the optimal result.

We conduct a week-long packet-level measurement campaign on Tencent START cloud gaming service to motivate the design of Hairpin (§2.3 and §2.4). We then implement Hairpin and evaluate it with both trace-driven simulators and real-world deployments in production (§4.1). Experiments demonstrate that Hairpin could significantly push forward the Pareto frontier [1] by reducing the DMR by 67%-80% and achieve comparable bandwidth costs simultaneously compared with state-of-the-art baselines (§4.3). Preliminarily testing Hairpin in Tencent START cloud gaming in production also shows significant and consistent performance improvements in different types of networks (§4.7). We will release the simulation codes of Hairpin.

Our main contributions are summarized as follows:

- We motivate the need for joint optimization of retransmission and redundancy through the operating experiences of a production edge-based interactive streaming service (§2).
- We present challenges in the joint optimization over retransmissions and redundancy for edge-based interactive streaming, and then propose Hairpin with MDP formulation (§3).
- We implement and integrate Hairpin in a cloud gaming application in production, and extensively evaluate its performance with trace-driven simulation and real-world deployments (§4).

[1] In badminton, a hairpin shot is played when the shuttle is very near to the ground and the net (the deadline of a shot) [76].

2 Background and Motivations

We introduce the interactive streaming (§2.1), present our measurement of packet losses (§2.2), analyze why existing solutions are insufficient (§2.3), and motivate the design of Hairpin (§2.4).

2.1 Interactive Video Streaming

Interactive streaming applications are increasingly attracting interest in many scenarios. Examples include cloud gaming [3, 8, 9], remote driving [2, 54], cloud phone / PC [13, 18, 39], and regional videoconferencing [4], forming a considerable market value of billions of dollars. Compared with legacy live video streaming, with the intensive deployment on edge nodes (or content generators in VR), the network delay over the wide-area network could be reduced for interactive streaming (e.g., an average RTT of 10-20ms [27, 58, 89]). With the recent emergence of the metaverse and so on, these interactive video streaming applications are going to be increasingly dominant on the Internet. Edge-based interactive streaming imposes specific requirements on transport, as summarized below.

Stringent deadline requirements. Since interactive streaming applications continuously interact with humans, controlling end-to-end delay is critical for a seamless user experience. For example, videoconferencing may expect an end-to-end delay of $<$130ms for network [49, 56], while cloud gaming would argue for a latency of $<$96ms [50][2]. In practice, server- and client-side processing usually take $\approx$30 ms [14, 42, 74, 83]. Therefore, the end-to-end round-trip delay *for the network* should not exceed 50-150ms (depending on scenarios), which is the deadline required by the application [11, 75].

This also corroborates our measurement study with users in our production cloud gaming service. We measure our cloud gaming service in production for one week (details in Appendix A), with O(10,000) users every day, and collect a variety of metrics. Unless other specified, the analysis using online data in this paper is also from this measurement campaign. We categorize the measured round-trip interaction delay of each video frame into several intervals. We present the appearance distribution of the position of those frames in a flow for each category in Fig. 2, where the x-axis is the position of that frame in a session normalized by the length of that session. Compared to the uniform distribution of low-delay frames (solid lines), frames with an end-to-end delay of $>$100ms (dashed lines) have a higher probability to appear around the end of a flow. We hypothesize that this is because users tend to exit a session if they have a high end-to-end delay. User's exiting behavior is a critical metric for user's experience in real-time video streaming [32]. In the meantime, setting a deadline for the delivery and reducing the fraction of higher than that specific value has also been widely adopted

[2] Based on the statistics of the majority of people.

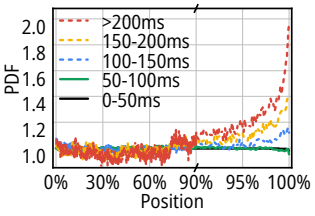


Figure 2: Probability density function (PDF) of the position of frames with different total delays.

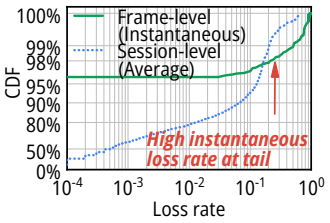


Figure 3: Session-level and frame-level loss rate distributions (both axes are log-scaled).

*Note: Unless otherwise specified, all measurements in §2 and §3 are from the production in the wild (measurement details in §4.1).

in real-time video streaming [56, 57]. The similarity between the 50ms and 100ms curve in Fig. 2 also indicates that, as long as packets could be delivered within the deadline (∼100ms in this case), faster delivery barely improves the user's experience.

Thus, we should minimize the *deadline miss rate* (DMR) to enable a seamless experience for users in interactive streaming, where in our cloud gaming service, the deadline for interaction delay is around 100ms. For interactive streaming, it is essential to minimize the occurrence of deadline misses for frames to an ultra-low level. For example, even a DMR of 10^{-3} still leads to a poor experience every 1000 frames (17 seconds at 60 fps), which drastically degrades the user's experience [11].

Reliable delivery. Meanwhile, interactive streaming also requires reliable delivery for each frame. For commercial video codec, failing to deliver a part of the frame will lead to severe image quality degradation. Moreover, the loss of one frame would also lead to blurring for the subsequent frames due to the dependency between frames[3]. Therefore, existing interactive streaming services usually try their best to reliably deliver frames. For example, industrial frameworks (e.g., WebRTC) [5, 46] and academic efforts [22, 40, 61] propose to employ forward error correction (FEC) to recover lost packets at the receiver if possible, and will retransmit lost packets if the recovery fails [7].

Low bandwidth cost. The bandwidth cost is still one of the largest operating expenses in our and other cloud gaming service [17]. Moreover, to achieve a satisfactory user experience, interactive streaming must stream with high video resolution and frame rate (e.g., 60fps and >1080p for cloud gaming), which requires high goodput to support. Given the requirements of low operating expenses and high video quality for users, we need to control the bandwidth cost in packet loss recovery.

2.2 Packet Losses in Edge-based Interactive Streaming

Our observation from our cloud gaming service is that although the median loss rate is as low as 10^{-3}, the *instantaneous* loss rate could be very high. In our measurement campaign as described in §2.1, we also calculate the *session-level loss rate*, which is the ratio of total lost packets in one user session (minutes to hours, containing at least O(10,000) frames), to reflect the average loss rate over a long timescale. We then calculate *frame-level loss rate*, which is the ratio of lost packets within one frame (tens of milliseconds), to show the instantaneous loss rate over a short timescale. For example, if a session has 1M packets and 10 of them are lost, the session-level average loss rate is 0.01%. Meanwhile, if these 10 packets belong to the same video frame which has 50 packets in total, the frame-level instantaneous loss rate will be 20% for that frame and 0% for other frames.

As shown in Fig. 3, the session-level loss rate is 0.05% at the median, which is comparable to similar measurements [45]. However, the instantaneous frame-level loss rate could be very high: 2% frames lose more than 20% of their packets within one frame. Such a high instantaneous packet loss poses a great challenge in controlling the deadline miss rate to 10^{-3} or lower – we can no longer ignore these transient behaviors and have to deliver video frames in time even when the instantaneous loss rate is high.

Moreover, these packet losses cannot be easily mitigated by reducing the sending rate. To achieve a low latency, most CCAs in interactive streaming use *delay* as the signal to reduce the sending rate (e.g., BBR [25], Copa [15], GCC [26]). In this case, congestion losses rarely happen since the sending rate has already been reduced based on an increasing delay in advance, which has also been measured in related work [26]. Our online measurements unveil similar observations: our cloud gaming service has already adopted a delay-based CCA similar to GCC [26], which is widely deployed in interactive streaming applications such as Chrome and Stadia. We further demonstrate the weak correlation between RTT[4] increases and packet losses in our measurement in §2.4. As shown in Fig. 3, losses are still outstanding at the tail, indicating that merely controlling the bit rate or frame rate is still insufficient to avoid packet losses for edge-based interactive streaming.

2.3 Why Existing Solutions Fail?

As we discussed in §1, packet losses contribute a lot to deadline misses. Thus, we investigate why existing packet loss recovery mechanisms are insufficient for edge-based interactive streaming. Existing solutions mainly fall into two categories as follows.

Retransmissions. Existing transport protocols (e.g., TCP) rely on retransmissions to cope with packet losses. Merely relying on retransmissions is insufficient to achieve an extremely low DMR for interactive streaming frames at the magnitude of 0.1% or lower. For example, when the packet loss rate is instantaneously 20%, there would still be 0.16% packets lost even after 3 retransmissions. Note that since there could be tens to hundreds of packets per frame, being unable to deliver even one packet would violate the deadline requirement of that frame since interactive streaming requires all packets to be reliably delivered (§2.1). Thus, the DMR of frames is still considerably high when relying on retransmissions and rate controls only. Our evaluation in §4 also demonstrate the performance degradation.

Redundancy-based algorithms. There are also several solutions

[3] Mechanisms such as scalable video coding (SVC) allow limited packet losses, yet reduce the bandwidth efficiency and require client support [70].

[4] In this paper, we use RTT to represent the delay at the network layer that does not contain the time of retransmission. We use application delay to refer to the delay at the application layer that contains the retransmissions.

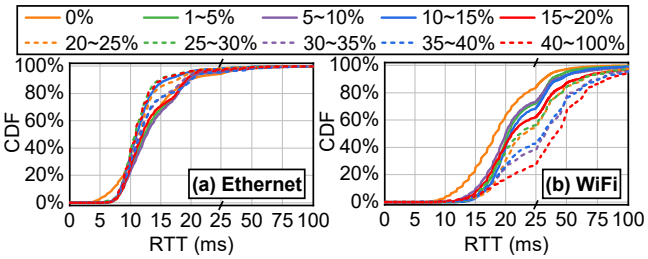


Figure 4: RTT distributions measured in production, categorized by the frame-level loss rate. Note that retransmissions are not counted.

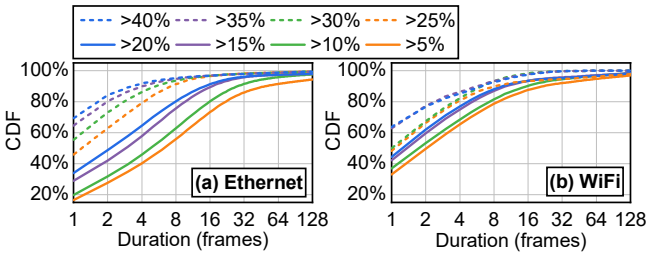


Figure 5: The distribution of the duration of each loss event measured in production. We measure the duration of each time when the loss rate is larger than different thresholds (5%, ..., 40%). Loss rates are measured at the frame level. The network type is reported from our cloud gaming clients. Better viewed in color.

in interactive streaming with redundancy mechanisms such as FEC. However, existing adaptive FEC solutions from both the industry [5,46] and academia [22,40,61] optimize the FEC parameters only for the initial transmission. They adjust the number of FEC packets according to loss rate and retransmit packets as usual when packet loss occurs. Note that packet losses are not deterministic: when the transient loss probability increases to 20%, it does not mean precisely one packet loss every five packets. In this case, to achieve an extremely low DMR of 10^{-3} or lower, FEC rates need to be much higher than the loss rate, leading to severe bandwidth cost (§4). For example, WebRTC, a state-of-the-art interactive streaming framework, will send 100% redundant packets during this short timescale of high instantaneous loss rate for initial transmissions. In this case, there will be considerable bandwidth cost while the DMR might still not be satisfied. We further evaluate the performance of other baselines in §4.2.

2.4 Motivations

Therefore, with the reduced RTT, retransmissions are tolerable to some extent for edge-based interactive streaming. In this case, we have the following observations on how and what to retransmit.

RTT being much lower than the deadline enables the joint optimization of redundancy and retransmission. As we discussed before, with an RTT of 10-20 ms and a deadline of 50-150 ms, multiple retransmissions are tolerable to some extent. This enables the joint optimization of redundancy and retransmission, which results in benefits in two folds:

- Reduce the deadline miss rate. In existing FEC mechanisms, many of the deadline misses come from the packet losses in the retransmissions. When adding redundancy packets over retransmission packets, we could effectively avoid the loss of retransmission packets and further reduce the deadline miss rate.
- Save bandwidth costs. To achieve the same DMR, the bandwidth cost of adding redundancy to retransmissions is significantly lower than that of only adding redundancy to initial transmissions. This is because retransmission packets are always the minority in bandwidth consumption – redundifying retransmissions will only introduce a little bandwidth cost, but could have significant DMR improvements.

When more rounds of retransmissions are tolerated (e.g., with smaller RTTs), the joint optimization will have more significant benefits (later presented in §4.5). We are thus motivated to utilize the retransmission chances enabled by edge deployments and jointly optimize the redundancy and retransmission mechanisms.

Loss recovery adaptions at the server are possible. Dynamically optimizing the tail cases of high instantaneous loss rate needs quick adaption. According to our measurement, the feedback loop between the server and client is smaller than the duration of loss events, making the joint optimization of redundancy and retransmission practical. This comes in two folds:

- The feedback loop does not inflate with the increase in the loss rate. We measure the RTT of our cloud gaming service and categorize them into different frame loss rate intervals. As shown in Fig. 4(a), the distribution of RTT does not significantly vary with the frame loss rate. The RTT in WiFi increases with the increase of frame loss rate (e.g., due to retransmissions at the link layer [31]). Nevertheless, even when the frame-level loss rate is 30% (the dashed green curve in Fig. 4(b)), 60% of those acknowledged packets have an RTT of less than 25ms. This indicates (i) the server is able to quickly detect the network condition changes, and (ii) there are still multiple transmission chances when the instantaneous loss rate increases.
- The duration of loss events is transient but still longer than several feedback loops. We measure the duration of lossy frames in our cloud gaming service and present the results in Fig. 5. According to our measurements, most loss events span multiple RTTs. For example, 70% of frames with a frame-level loss rate of >10% will last more than 2 frames in Ethernet sessions, which is several times the median RTT (12ms) at the frame rate of 60fps. Therefore, the reaction from the server is still effective to alleviate packet losses by adjusting the redundancy parameters.

3 Hairpin Optimizer

As we discussed above, edge-based interactive streaming needs to reduce the deadline miss rate and bandwidth cost. For clarity, we first present the formula of frame deadline miss rate (DMR) and bandwidth cost (BWC):

$$\begin{aligned} DMR &= \frac{\text{\#Frames arrive after the deadline}}{\text{\#Total frames}} \\ BWC &= \frac{Redundancy_{byte} + Retransmission_{byte}}{Data_{byte}} \end{aligned} \tag{1}$$

A higher DMR or BWC means more frequent stutters or higher operating expenses respectively, both of which interactive streaming service providers will try to avoid. Note that pushing

DMR to an extremely low level is critical since the lower it is, the better user's experience is going to be.

In this section, we first summarize some intuitions in the design space of joint optimization of redundancy rate and retransmission and present a strawman solution (§3.1). We then present the design challenges in the joint optimization of retransmission and redundancy (§3.2). We address these challenges by providing a Markov chain-based optimization algorithm to efficiently improve both the DMR and BWC (§3.3). We finally discuss how Hairpin handles the inaccuracy in measurement, the overhead in online deployment, and other practical issues in §3.4.

3.1 Basic Idea and Strawman Solution

Differentiating retransmissions from initial transmissions. The most important insight in this paper is to understand the significance of differentiating retransmissions from initial transmissions. In other words, we want an adaptive redundancy rate based on the planning of multiple transmission chances. The short RTT of edge-based interactive streaming enables packets to have more than one transmission chance without violating the deadline. The ratio of RTT and remaining time t indicates the potential number of (re)transmissions. For example, when the current RTT is 20ms and packets still have 40ms towards their deadline, the ratio follows $\frac{t}{RTT} = \frac{40ms}{20ms} = 2$, indicating that these packets could be approximately transmitted twice before the deadline. Packets with more transmission chances could better utilize the potential retransmissions to deliver packets before the deadline, which has already been discussed in §2.4. Therefore, our basic idea is to take future transmission chances into consideration when optimizing the redundancy rate. When one batch of packets has more foreseeable transmission chances (i.e., the deadline is still far away), we could reduce the redundancy rate to save bandwidth costs. When the remaining time of these packets is getting closer to the deadline due to retransmissions, we could further increase the redundancy rate to avoid deadline misses.

Strawman solution: RTT-aware adaptive FEC algorithm. Therefore, a strawman solution is to (i) add redundancy to both initial transmissions and retransmissions, and (ii) consider the remaining transmission chance in the optimization of the redundancy rate. Since there have already been existing solutions on the redundancy rate based on network conditions [5, 22, 61], we could introduce a multiplier controlled by the transmission chance over the existing redundancy rate optimizations, i.e. a strawman solution is to reduce the redundancy rate when there are many transmission chances, and increase it when transmission chances are few. Thus, we could enhance these algorithms by introducing a factor over the results from existing algorithms.

FEC consists of two parameters (d,k), where d data packets and k redundant packets are sent as a *block*. Block is composed to the convenience of FEC encoding. If there are up to k packets lost in an FEC block (d,k), an ideal FEC decoder can recover all data packets with any remaining packets [64, 65, 85]. We denote $\beta = \frac{k}{d}$ as the FEC redundancy rate, and d as the FEC block size.

Specifically, given a packet loss rate α and bitrate B, assume one of the state-of-the-art solutions has already determined that $\beta_0(\alpha,B)$ should be the optimized redundancy. We could then increase or decrease the redundancy rate $\beta_0(\alpha,B)$ based on the remaining transmission chance $\frac{t}{RTT}$, i.e.:

$$\beta(\alpha,B,RTT,t) = k \cdot \frac{RTT}{t} \cdot \beta_0(\alpha,B) \tag{2}$$

where k is a coefficient to adjust how aggressive the strawman solution is going to increase or decrease the redundancy rate.

In fact, according to our evaluation in §4.6, such a strawman solution is enough to push the Pareto frontier of DMR-BWC forward. However, it confronts a series of shortcomings, which prevents the operator from further improvements in performance. We will elaborate on these challenges in the following section.

3.2 Design Challenges

Although we have presented a heuristic RTT-aware adaptive FEC algorithm as above, it is still challenging to optimize these parameters due to the following reasons.

Temporal dependency: cascading decision-making between transmission rounds. When considering multiple transmission chances, the decision of FEC parameters of one round of transmission would cascadingly affect the optimization of the next round. For example, if we aggressively add a high redundancy rate to a group of packets, the number of packet losses will then be decreased. On the contrary, a low redundancy rate for the same group of packets would probabilistically increase the number of packet losses under the same network condition. However, these packet losses bring more packets to retransmit in the next round. If we consider all actions for F packets for the foreseeable L rounds of transmission, the action space will be extremely large: Since for each redundancy decision, there are F possible scenarios of the number of packets to transmit in the next round (depending on how many packets are lost), the number of variables that we need to optimize will be $O(F^L)$[5]. Therefore, in the enlarged action space over multiple retransmissions, it is challenging to efficiently optimize. Moreover, the conditional probability between scenarios is not linear (e.g., hypergeometric for individually independently and identically distributed losses). Therefore, using traditional optimization methods such as integer programming in an extremely large action space is impractical. We need to coordinate the choices in different rounds of transmission to achieve optimal performance.

Spatial dependency: redundancy rate and block size are tightly coupled. Even in a single round, different variables (e.g., redundancy rate, block size, etc.) still have complicated dependencies on each other. This goes to the following aspects:

(a) Number of packets to transmit in one round affects redundancy rates. The number of packets to transmit in the different rounds is varying, depending on how many data packets are lost

[5]For a frame with 50 packets (F=50), and 5 potential transmission rounds (L=5, e.g., RTT is 20ms and deadline is 100ms), this turns into 10^8 variables.

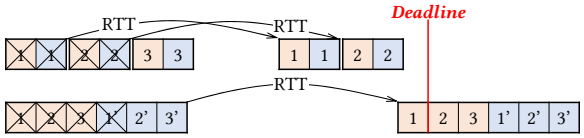
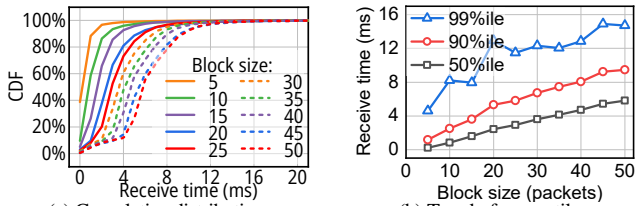


Figure 6: Smaller block sizes in one frame could have better performance. Scenarios above and below represent using small and large blocks. Data and FEC packets are shaded orange and blue.



(a) Cumulative distribution. (b) Trend of percentiles.

Figure 7: Block receiving time with different block sizes. FEC blocks are burstily sent out at the server side. Fig. 7(b) is processed from Fig. 7(a). Measurement details in §4.2. Better viewed in color.

during the last transmission. The penalty of redundancy rate on BWC also varies according to the number of packets to retransmit. For example, when there are few packets to retransmit, even adding a redundancy rate of 100% for retransmissions would not consume too much bandwidth, as also discussed in §2.4. Therefore, fewer data packets to retransmit would encourage a more aggressive redundancy rate. The strawman solution is not aware of the dependency here, leading to its suboptimal result.

(b) Dispersion of blocks might lead to deadline misses when using larger blocks. Due to the bandwidth limit at the bottleneck link, packets sent out at the same time could be dispersed [47] and arrive at the receiver one by one. In this case, constructing large blocks will increase the delay to wait for all packets at the receiver. Since packet losses can only be determined after the completion of one block, smaller blocks may know earlier whether they need retransmission and enjoy additional transmission chances before the deadline. For example, in Fig. 6, due to the early determination of packet loss, the retransmission of data packets for small blocks could arrive at the receiver before the deadline, while no packets could arrive before the deadline for large blocks. We quantify the influence of block size by measuring the receiving time of FEC blocks from our service online with different block sizes. As we can see in Fig. 7 and 7(b), with a block size of 50 packets, more than 10% blocks could span 10ms at the receiver, which is even comparable to the RTT. Also, smaller block sizes might also be beneficial when the loss rate is higher than the redundancy rate. As illustrated in Fig. 6, when the first four packets are lost during the transmission, data packet #3 could still be successfully delivered for a small block size (the case above in Fig. 6). For large blocks, there is no way to recover any lost packet if the loss rate is larger than the redundancy rate.

Convoluted goal: deadline miss rate and bandwidth cost. Unlike latency or throughput which we can directly measure, the estimation of the expected deadline miss rate needs to consider multiple potential rounds of transmission. In this way, the strawman solution, without explicitly estimating whether that frame is going to miss the deadline or not, will have suboptimal results. For example, the relationship between the packet loss rate and the success rate of delivering a video frame with tens of packets in a single round is *hypergeometric*, even under the identical and independent distribution (i.i.d.) assumption. Considering multiple future rounds together will only make the relationship between deadline miss rate and network conditions more convoluted. Moreover, some applications or even the same application in different operating regions may have different preferences over deadline miss rate v.s. bandwidth cost. The traffic cost in some regions might be higher than in another, and some applications may give it all for the user's experience while others may not. Therefore, we need to explicitly optimize towards the goal to achieve the optimal result.

3.3 Model Formulation and Optimization

We have the following designs to address the challenges above.

Encode the temporal dependency in multi-round planning into edges in Markov chain. Markov chain is widely used in the optimization of the sequential decision-making process (e.g., reinforcement learning [79]). With the Markov chain, we can formulate the loss detection between two rounds of (re)transmission into the transition between two Markov nodes. In this case, by only focusing on the optimal parameters between the transition of the current state and its potential states in the next round, we could decouple the cascading effects of the transitions between neighbor nodes, which reduces the action space significantly. We further show in Appendix B.1 that, in such a Markov chain, locally focusing on the neighbor nodes could still have globally optimal results.

Encoding the spatial dependency between variables into nodes in Markov chain. To ensure the number of packets to transmit is considered in the optimization, we build a 2-D Markov chain, with two dimensions as the transmission chance and the number of packets to transmit. We present the state transition of our Markov chain in Fig. 8. Each node is represented by (d,l), where d denotes the number of remaining data packets to transmit, and l represents the remaining transmission chance for those packets. Our goal is to find out the optimal redundancy rate for node (B,L), where B is a given block size, and L is the remaining transmission chance from Eq. 3. In this case, both the temporal dependency and spatial correlation between variables could be formulated into this 2-D Markov chain.

Explicitly optimize deadline miss rate and bandwidth cost with Markov chain formulation. We finally provide an explicit expression of the deadline miss rate and bandwidth cost for multi-round optimization within the formulation of MDP. We inversely calculate the DMR and BWC at different states from the last chance to transmit (as the last layer of the Markov chain), to the first chance to transmit (as the first layer of the Markov chain). In this way, the transition probabilities between states could be directly iterated. We further decouple the optimization of redundancy rate and block size to improve the optimization efficiency.

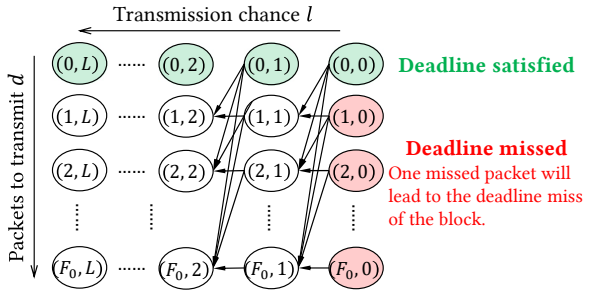


Figure 8: The absorbing Markov chain in redundancy rate optimization at given loss rate and frame size. l is the estimated remaining transmission chances for the packets to transmit.

We present the analytical model and the algorithm below. In interactive streaming, frames are continuously generated and sent out from the server. There are thousands to millions of frames within one stream, depending on the specific application, where the retransmission of previous frames overlaps with the transmission of subsequent frames. Therefore, similar to the finite element analysis in mechanics [10], we pick one frame from the stream, and analyze the expected DMR and BWC of that frame. The expected DMR and BWC of one frame should be consistent with the DMR and BWC of a stream. We list all notations that will use in Table 2 in Appendix B.1. Specifically, Hairpin optimizes the FEC parameters as follows:

Step 1: Calculating remaining transmission chance. Given current network RTT, the remaining time towards deadline T, the bottleneck bandwidth Θ, and a certain block size d, the remaining transmission chance L could be calculated as:

$$L = \frac{T - d/\Theta}{RTT} \tag{3}$$

Step 2: Generating absorbing Markov chain. We then calculate the optimal redundancy rate given the current loss rate α and frame size F. We iteratively calculate the absorbing Markov chain from layer $l-1$ to layer l. For the node (d,l), at a certain redundancy rate β,we respectively calculate the $DMR(d,l;\beta)$ and $BWC(d,l;\beta)$ based on the DMR and BWC from the $l-1$-th layer. We leave the detailed equations to Appendix B.1. Then, we calculate the optimal β for (d,l):

$$\beta_{opt}(d,l) = \text{argmin}_{\beta}\, utility(DMR(d,l;\beta), BWC(d,l;\beta)) \tag{4}$$

Here, $utility(DMR, BWC)$ is the utility function to balance preference for low DMR and low BWC. For simplicity, we adopt a linear combination of DMR and BWC as the optimization goal:

$$utility(DMR,BWC) = DMR + \lambda \cdot BWC \tag{5}$$

Note that Hairpin does not fall into the same trade-off between DMR and BWC as baselines, but improves both DMR and BWC, as we will evaluate later in §4.3. In practice, service providers can adjust the coefficient λ to balance stuttering events and bandwidth costs in different scenarios. A lower λ indicates that users prefer the deadline miss rate more than bandwidth costs. We also evaluate performance with different utility functions in §4.5.

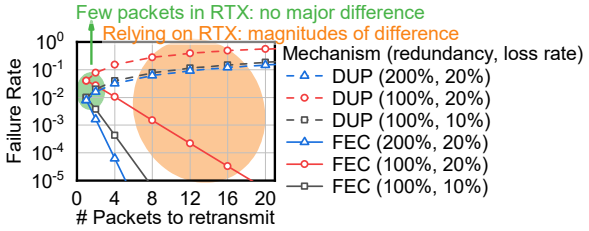


Figure 9: A theoretical illustration of the failure rate of retransmitting different numbers of packets by per-packet duplication or constructing FEC blocks. The failure rate of DUP increases with the number of packets to retransmit, since we need to ensure every data packet is delivered. We vary the redundancy rate and loss rate.

Therefore, the redundancy rate for (B,L) could be optimized accordingly. After calculating all nodes at the layer l, we could then calculate the DMR and BWC at the layer $l+1$, until the node (B,L) has been calculated. Since the iterations between nodes are linear, as long as the utility function is monotonic to DMR and BWC (e.g., linear relationship), the optimality still holds.

We set $DMR(d,0)$ to 1 for $d > 0$ since one missed packet would lead to the miss of the block (shaded green). We also set all $DMR(0,l)$ to 0 since there is no remaining packet to transmit. The BWC for all these boundary nodes is set to 0. Note that different block sizes and remaining transmission chance could multiplex the same chain to accelerate the optimization, since the chain only depends on loss rate α and frame size F.

Step 3: Calculating optimal block size. We enumerate the possible block sizes from 1 to the frame size, calculate the DMR and BWC for each block according to the chain in Step 2, and finally find the optimal block size in terms of a given utility function. We leave the mathematical details to Appendix B.2. According to our evaluation in §4.6, not surprisingly, when the bottleneck bandwidth is high (i.e., the dispersion is insignificant), the optimal block size for most scenarios is the frame size. Nevertheless, when the dispersion is significant, constructing smaller blocks could achieve better DMR. Operators could optimize the block size for improvements at the last mile.

During the optimization of block sizes, we also optimize the trade-off of when a loss has been detected, whether to retransmit that packet as soon as possible or wait for other packets to formulate an FEC block. On recovery ability, constructing several lost packets into one FEC block might be more effective than individually retransmitting (or duplicating, if with redundancy) each packet. We calculate the failure rate of delivering these packets when there are different numbers of packets to retransmit at different redundancy rates and loss rates and present the results in Fig. 9. When there are few packets that need retransmission, whether duplicating or constructing FEC blocks has no major difference (dashed line and solid lines shaded green). However, when optimizing at the *tail* for interactive streaming, there could be multiple packet losses within one frame. Therefore, considering each frame could contain tens of packets, it is possible to suffer losses of 4 packets or more at the tail. Constructing FEC blocks for these retransmission packets could reduce the failure rate of delivering packets by several magnitudes.

Step 4: Getting the optimal parameters. Finally, based on network conditions and remaining time towards a deadline, Hairpin can calculate the optimal block size based on Step 3, and the optimal redundancy rate with the block size based on Step 2.

3.4 Deployment Discussions

In §3, we analytically optimize the FEC parameters given certain network conditions. The reality might be more complicated than the theoretical model. In this section, we discuss several practical concerns of Hairpin based on our operational experiences. Our further trace-driven simulation and deployments in production in §4 also demonstrate the effectiveness of Hairpin in the wild.

Reducing computational overhead online. Hairpin adopts an optimization-based algorithm, which might not scale to production-scale deployments in terms of computational overhead. Since the optimization needs to run frequently (approximately every frame) and scale to tens of thousands of users simultaneously, it should be computation-efficient and time-efficient. In response, we do an offline step of enumerating the state space and solving each specific instance. Then, in the online step, the algorithm will be reduced to a simple table lookup towards pre-computed optimized redundancy parameters. We enumerate the state space of Hairpin as below.

1. Remaining transmission chance: 1 to 10.
2. Loss rate: 0% to 50% with quantization of 1%.
3. Frame size: 5 to 60 packets with quantization of 5 packets.
4. Number of packets to (re)transmit: 5 to 60 packets with quantization of 5 packets.

Hairpin then stores the best redundancy rate and block size under different conditions. We found that the benefits of finer quantization are marginal. Our further evaluation in the real world in §4.7 shows that the memory consumption (2MB) and table lookup time are negligible for online deployment.

Handling network fluctuations. We discuss how Hairpin handles the fluctuations in network conditions. For RTT, as presented in Fig. 4, RTT does not increase too much – the median RTT always allows Hairpin to have 3-5 transmission chances no matter the loss rate. Moreover, we further measure the network conditions in Hairpin with a short sliding window to make sure Hairpin has the most recent network conditions. We set the measurement window to 2 frames and evaluate the sensitivity of this parameter in §4.5. In this case, the transient fluctuation of RTT could be reflected in the optimization results immediately. We later demonstrate in §4 that Hairpin behaves well with real-world traces and production deployments.

Handling various loss patterns. In this paper, when given a certain loss rate, Hairpin assumes the pattern of packet losses is identically and independently distributed (in the transition probability of Eq. 6). Note that the duration of a certain loss rate still follows the results of the online measurement in Fig. 5. In practical deployment, working with FEC codecs that could recover from different loss patterns (bursty or arbitrary) [65], Hairpin could also handle different loss patterns since Hairpin only focuses on how many packets within a block are lost. Since our data is collected frame by frame, if the burstiness spans over several frames, it will be directly reflected on the value of loss rates. If the burstiness spans within the frame, no matter how the pattern changes, the number of lost packets will not change, which does not affect the recovery efficiency of the FEC codec. For example, when there are 4 packet losses in one block, no matter whether these losses are consecutive or separated in the block, as long as there are 4 additional FEC packets in the same FEC block, the client would be able to recover these packet losses. Therefore, Hairpin does not rely on the assumption of underlying loss patterns, but only focuses on the number of lost packets. Packet losses might be consecutive across several frames. In this case, due to the short feedback loop enabled by edge deployments, Hairpin should have already timely reacted as analyzed in §2.4.

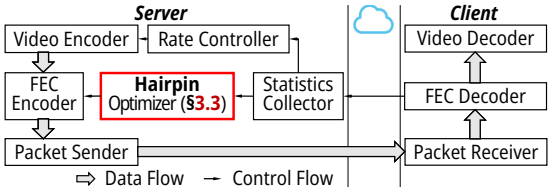


Figure 10: Overview of Hairpin implementation.

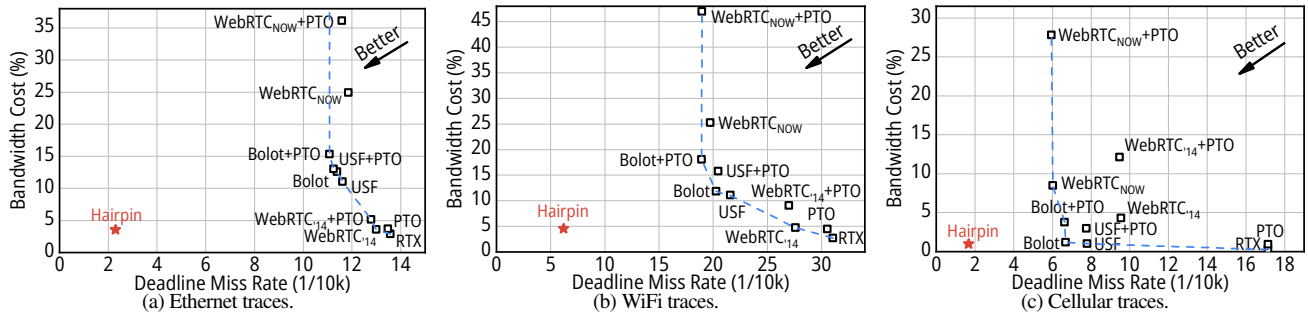
4 Evaluation

We introduce the implementations in §4.1 and experiment settings in §4.2. We further answer the following questions:

- How does Hairpin perform under real-world traces? We demonstrate that Hairpin could push forward the Pareto frontier of baselines on DMR and BWC (§4.3).
- Is Hairpin sensitive to the settings of parameters? We investigate the performance variation of Hairpin with different parameters, and demonstrate that Hairpin has performance improvements in a wide range (§4.5).
- Which design in Hairpin contributes to the performance improvement against other baselines? In §4.4, we break down the performance improvements of Hairpin.
- How does Hairpin make decisions and how each optimization affects the user's experience? We further dive into the details of Hairpin's design in §4.6.
- How does Hairpin perform well in the wild? Finally, we deploy Hairpin in production servers and find that Hairpin significantly improves both DMR and BWC in the real world (§4.7).

4.1 Hairpin Implementation

We implement Hairpin in both an `ns3`-based WebRTC simulator [88] and our cloud gaming application in production. We present the workflow of Hairpin in the network stack in Fig. 10. Without Hairpin, interactive contents are first encoded with `Video Encoder` by the application, and then sent out at the transport layer frame-by-frame. Then the video frames could be received by the protocol stack at the client. `Packet Sender` and `Packet Receiver` abstract the network stack at the transport layer for connection management. After that, `Video Decoder` decodes the streaming contents and displays



(a) Ethernet traces. (b) WiFi traces. (c) Cellular traces.

Figure 11: Trace-driven simulation. The blue dashed line is the envelope of all baselines on the Pareto frontier.

them to users. Meanwhile, network conditions (e.g., RTT, packet loss events) will be measured at the server, collected by the `Statistics Collector`, and reported to `Rate Controller` to adaptively adjust the streaming bit-rate according to network conditions [26]. Hairpin inserts between the existing application layer and transport layer, and optimizes the redundancy parameters based on current network conditions, as shown in Fig. 10. The network statistics is still passed to the congestion controller (rate controller) without modification. The underlying transport protocol in our cloud gaming service is a customized version of the RTP protocol [69] based on UDP to allow the loss of redundant packets without modifying the kernel at the client. We implement Reed-Solomon FEC due to its recovery performance when the redundancy rate is $<$100% [65], and implement a customized FEC codec for the redundancy rate of $>$100%. Note that Hairpin could also work with other codecs (e.g., XORFEC, FlexFEC, etc.) as long as their parameters are exposed to Hairpin. We discuss FEC codecs in Appendix C.

4.2 Experiment Setup

Traces. As for simulation traces, we collect a dataset in one production server in the wild on our cloud gaming service in two weeks in January and August 2021, resulting in more than 100M video frames and more than 600 hours of playtime. This also supports our measurements in §2 and §3. Users access the service via either Ethernet, WiFi, or cellular connection, which we collect from our cloud gaming client. The cloud gaming service streams at the frame rate of 60fps and the bit rate ranging from 2Mbps to 30Mbps. The network conditions are recorded on the server of our cloud gaming service, including the average RTT, average bit rate, and loss rate at the frame level (approximately every 16 ms). The traces contain 1,995 Ethernet gaming sessions, 741 WiFi sessions, and 572 cellular sessions in total, each lasting from minutes to hours. To the best of our knowledge, we are the first to collect online traces from an interactive streaming service for weeks at both the frame level and packet level.

Baselines. We orthogonally review the public adaptive FEC mechanisms and retransmission mechanisms with deployments in practice. On the axis of retransmission optimization, we implement the following baselines.

- `Out-of-order`. Traditionally, packet losses are detected by checking the out-of-order packets, such as TCP duplicated ACK [21]. We use it as our default loss detection mechanism.
- `Probe timeout (PTO)`. Besides, to quickly detect packet losses of tail packets, recent researchers also propose an aggressive timeout-based loss detection mechanism [33].

On the axis of redundancy parameter optimization, we implement the following mechanisms:

- `WebRTC'14` comes from the research paper published by Google in 2014 [46].
- `WebRTC`$_{NOW}$ is the adaptive FEC mechanism used in WebRTC now (adopted by Google Stadia [35], Meet [23], etc.), replacing the `WebRTC'14`. The difference is that `WebRTC'14` is aware of RTT and will reduce the redundancy rate when RTT is low, while `WebRTC`$_{NOW}$ is more aggressive on adding redundancy. We migrate the implementation of the m88 version of Chromium released in December 2020 [7].
- `Bolot` [22] and `USF` [61] are two heuristic adaptive FEC algorithms from the research community. Unlike Hairpin, they do not add redundant packets for retransmissions.
- `RTX` adds no redundancy, but fully relies on retransmissions.

Note that none of these baselines optimize the redundancy for retransmissions here. Since these two lines of work are orthogonal to each other, we combinatorially implement 2 (retransmission) $\times$ 5 (redundancy) $=$ 10 baselines.

Hairpin Setup. In our simulation, we set the coefficient in the utility function in Eq. 5 to $\lambda = 10^{-4}$, the measurement window of network conditions to 2 frames, and the deadline to 100ms. We evaluate the sensitivity of these parameter settings in §4.5.

4.3 Trace-driven Simulations

To evaluate the performance of Hairpin in dynamic network conditions, we simulate Hairpin over real-world traces as introduced in §4.2. We emulate the collected traces of loss rate and RTT with `ns-3`, and evaluate whether Hairpin could capture the network dynamics of loss and RTT variations and effectively adapt in real traces. We first present the trade-off between DMR and BWC over three sets of traces in Fig. 11.

As shown in Figure 11, `RTX` has the lowest bandwidth cost since `RTX` only retransmits a packet after it is lost. However, it also has the highest deadline miss rate among all baselines. Meanwhile, `WebRTC`$_{NOW}$ working with `PTO` has the lowest DMR among all baselines but also the highest BWC. Other baselines stay on the Pareto frontier in the trade-off between DMR and BWC. In contrast, Hairpin could break the trade-off

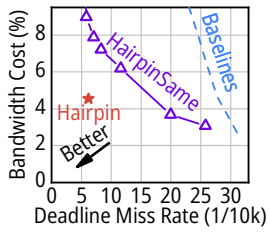


Figure 12: Breaking down the performance improvements of Hairpin.

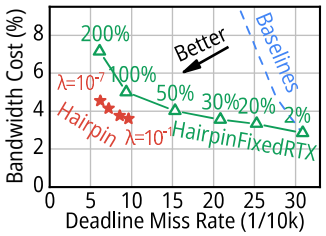


Figure 13: Parameter sensitivity of λ in Hairpin, and the effectiveness of differentiating retransmission.

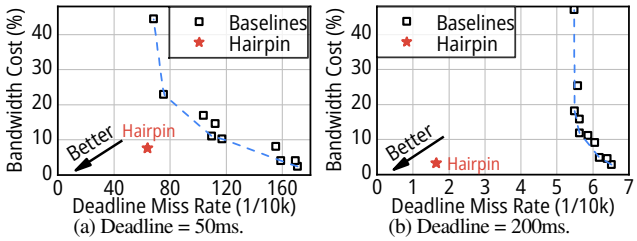


Figure 14: The performance of Hairpin and all baselines (labels omitted for brevity) on WiFi traces when the deadline requirement from the application is different.

and achieve a much better DMR and BWC, as the red stars denoted in Figure 11: 67%-80% lower than the lowest DMR (WebRTC$_{NOW}$), and comparable BWC as RTX. Thus, as we analyzed above, Hairpin could effectively improve both DMR and BWC significantly compared with all other baselines.

Note that the traces here are collected from our production servers, including the network RTT and instantaneous loss rate, with a fined granularity of every 16ms. The results in WiFi traces are worse than in Ethernet traces since WiFi traces have higher loss rates and RTTs, as measured in §2.4. Results over cellular traces are surprisingly good. This is because, during our online measurements, we just started to provide cloud gaming service for cellular users and had admission control over network conditions during that time. In all, Hairpin could significantly push forward the Pareto frontier of existing baselines in all traces.

4.4 Performance Breakdown

As discussed before, the performance improvement of Hairpin comes from two aspects: a carefully crafted optimization model, and the design space of differentiating retransmissions from initial transmissions. To investigate the contribution of these two components, we set up another candidate: HairpinSame. HairpinSame adopts the same Markov Chain-based model as Hairpin, but enforces the redundancy rate of all rounds to be the same. We further sweep the choice of λ from$\{10^{-4},10^{-2},10^{-1},1,5,10\}$ for HairpinSame to show the trade-off between bandwidth cost and deadline miss rate in this non-differentiating scenario. We also present the envelope of all baselines from Figure 11(b) as the blue dashed line in Figure 12.

As shown in Figure 12, even without differentiating retransmissions, HairpinSame is still able to significantly push the frontier forward. For example, when the bandwidth cost is 4.5%, Hairpin is able to reduce the deadline miss rate from to 0.28% to 0.17%, as shown in the gap between the purple solid line and blue dashed line. We are not going to argue that the proposed Markov chain algorithm outperforms all the baselines due to the optimization algorithm. We hypothesize that the improvement is due to the change of the optimization goal. Previous baselines focus on the optimization of tail latency, which is different from deadline miss rate as discussed in §2. In fact, we further present the latency distribution of Hairpin and all baselines in Appendix D (Figure 22), and the results of Hairpin and baselines are comparable for most percentiles below the 99.9th. Therefore, with the focus mostly on the extreme tails after the 99.9th percentile, Hairpin is able to significantly reduce the deadline miss rate, or reduce bandwidth cost for the same deadline miss rate.

Nevertheless, with the new design space of differentiating retransmissions, Hairpin can further reduce the deadline miss rate from 0.17% to 0.06% when the bandwidth cost is 4.5%, as shown in the gap between the red star and purple line in Figure 12. This further demonstrates the importance of differentiating retransmissions. Both designs are critical to performance improvement.

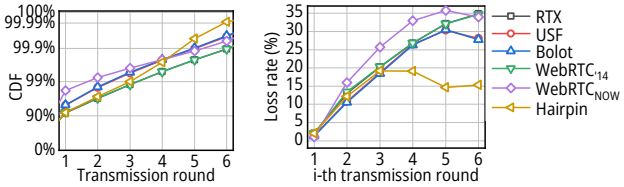
We further want to see how far a naive algorithm, which does nothing but differentiates retransmissions, will go. To this end, we have HairpinFixedRTX, which only adds FEC packets to retransmissions with a fixed ratio, and never adds FEC packets to initial transmissions, in contrast to all existing solutions in §4.2. We vary the static redundancy rate for retransmissions from 2% to 200%. As shown in the gap between the green solid line and the blue dashed line in Figure 13, HairpinFixedRTX significantly improves the trade-off between DMR and BWC against existing baselines. The series of red stars will later be explained in §4.5. This demonstrates that differentiatedly adding FEC over initial transmission and retransmission packets can effectively improve performance. As we discussed in §3, even naively differentiating the retransmissions with another fixed redundancy rate would already be helpful, illustrating the necessity of differentiating retransmissions.

4.5 Parameter Sensitivity

We also evaluate how Hairpin performs with different parameters.

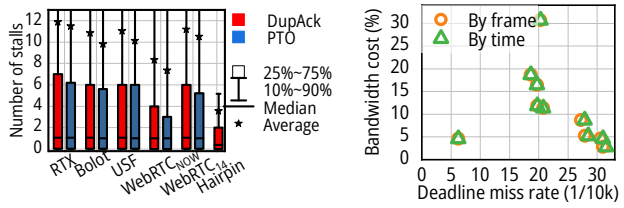
Utility coefficient λ. For the utility coefficient λ in Eq. 5, as introduced in §4.2, it could adjust the preference over the trade-off between the DMR and BWC. A higher λ indicates that users prefer the BWC more, while a lower λ indicates that the DMR is outweighing the BWC. Therefore, we change λ from 10^{-1} to 10^{-7}, and present the DMR and BWC of Hairpin with different λ over WiFi traces in Fig. 13. Note that Fig. 13 is zoomed in from Fig. 11(b). As shown in the red stars in Fig. 13, the BWC is decreasing with the increase of λ, while the DMR is increasing by a little. Thus, operators could adjust λ to balance the DMR and BWC according to the requirements of applications.

The setting of the deadline. In the evaluation in §4.3, the deadline is set to 100ms. We also investigate how Hairpin performs when the deadline is shorter or longer. Thus, we present the results of DMR and BWC of Hairpin and baselines over WiFi traces when the deadline is set to 50ms (Fig. 14(a)) or 200ms



(a) Number of rounds. (b) Loss rates by round.

Figure 15: The loss rate in each transmission round.



(a) Number of stalls per session. (b) DMR by frame and by time.

Figure 16: The effect of DMR on other metrics.

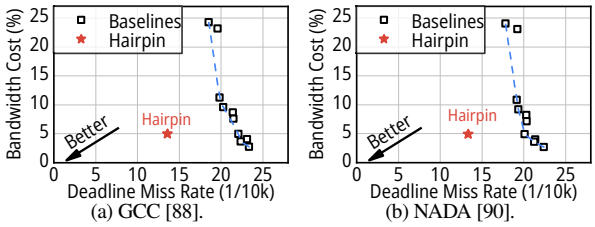
(Fig. 14(b)). As presented in Fig. 14, given the same trace, when the deadline is shorter (50ms), the advantages of Hairpin over baselines are a little less than when the deadline is 100ms. This is because the retransmission chance is less and the design space is smaller when the deadline is shorter. Nevertheless, Hairpin is still much better than all existing baselines. When the deadline is longer, the benefits are even larger due to the larger design space in retransmission. Results over other sets of traces are similar.

4.6 Hairpin Deep Dive

We further provide a deeper understanding of Hairpin in the following aspects.

Understanding Hairpin's decisions. In Appendix D, we present the redundancy rate and block size results of Hairpin to provide a deeper understanding of how Hairpin optimizes in different scenarios. Besides, we present the number of transmission rounds of Hairpin and baselines in Fig. 15(a). When Hairpin gradually increases the redundancy rate in future transmission rounds, most frames could therefore be delivered. Thus, the 99.9th percentile of the number of transmission rounds in Hairpin is less than all other baselines by more than one. Similarly, when we inspect the loss rate in each round as shown in Fig. 15(b), Hairpin also successfully maintains the lowest loss rate when the transmission round goes up. Note that the loss rate here is significantly high due to the survivorship bias – only lost packets will have another transmission round, while loss has already indicated a degraded network performance. This also indicates that the loss is not i.i.d. but bursty in the experiments.

Optimizing towards extremely low DMR. We further illustrate why we need to achieve an extremely low DMR and how it affects user's experience. As analyzed in §3.1, a lower DMR approaching zero directly indicates fewer stall events in a gaming session. We measure the number of *stall events* in each gaming session, where stall event is only counted once if there are multiple missed frames in one second or if it lasts longer than one second. As shown in Fig. 16(a), Hairpin can reduce the



(a) GCC [88]. (b) NADA [90].

Figure 17: The performance of Hairpin and all baselines (labels omitted for brevity) on WiFi traces with different deadline requirements.

average and median number of stall events (which is also critical for user's opinion scores [67]) by a half or more against baselines. By having a DMR of 0.06%, Hairpin is able to reduce the 75th percentile number of stalls in a session to 2. Considering the duration of a gaming session (minutes to hours), this will considerably improve the user's experience.

We also show the difference of calculating DMR *by frame* and *by time* in Fig. 16(b). In this paper, we do not argue using a new metric (DMR by frame) is better – we calculate DMR by the number of missed frames over total frames because of the simplicity in the formulation in §3.3. Calculating DMR by time is almost equivalent to DMR by frame since the stalled time is the number of stalled frames (missed frames) times the interval between frames. Therefore, we replot Fig. 11(b) using two different DMRs. As shown in Fig. 16(b), the results are almost the same with each other.

Integrating with congestion control. To further investigate the performance of Hairpin when interacting with the CCA, we integrate the Hairpin with two CCAs in the WebRTC framework, GCC [26] and NADA [90], in our simulation. We then replay the collected traces by setting their bandwidth, RTT, and loss rate to the link in ns-3. The bandwidth ranges from 2Mbps to 30Mbps. As shown in Fig. 17, Hairpin could still achieve significant advantages over all existing baselines.

4.7 Real-World Experiments

Finally, we deploy the Hairpin in a production server in our cloud gaming service. We conduct an A/B test in production of Hairpin against the $\texttt{WebRTC}_{NOW}$ baseline. The bit rate of the cloud gaming service also supports the range of 2-30Mbps as simulated in §4.3. The A/B test runs for one week in September 2021, covering 17k sessions in total, all of which have a duration of at least 4 minutes. Hairpin has been integrated into the UDP-based connections of our cloud gaming service since then. Since other optimizations are also deployed into our service after we deploy Hairpin, to make a fair comparison, we only present the results from the controlled A/B test in September 2021.

Performance. As shown in Table 1, Hairpin is able to improve both the average DMR and the average BWC compared to $\texttt{WebRTC}_{NOW}$. Specifically, for Ethernet sessions, Hairpin could improve the DMR by 32% while also reducing the BWC by 40% against $\texttt{WebRTC}_{NOW}$. For WiFi sessions, the improvements on DMR and BWC are 30% and 43%. We also measure the ratio of sessions with an average DMR of larger than 1%, i.e.

Ethernet	DMR	BWC	P(DMR>1%)	#Session
WebRTC$_{NOW}$	0.34%	30.4%	6.9%	8380
Hairpin	0.23%	3.0%	4.6%	7306
WiFi	DMR	BWC	P(DMR>1%)	#Session
WebRTC$_{NOW}$	0.72%	31.8%	19.3%	652
Hairpin	0.51%	3.0%	15.3%	613

Table 1: Real-world experiment results. P(DMR>1%) denotes the ratio of sessions with an average DMR of larger than 1%.

tail sessions. Hairpin could also reduce the tail sessions by 34% and 21% for Ethernet and WiFi sessions respectively compared to WebRTC$_{NOW}$. Note that the DMRs in real-world experiments are a little higher than those in simulations (§4.3). This might be because of other external factors (e.g., user devices) that could affect the DMR. Nevertheless, Hairpin could significantly improve the users' experiences on both the DMR and BWC compared to WebRTC$_{NOW}$.

Overhead. We further measure the overhead of the optimization of Hairpin. As introduced in §3.4, to accelerate the optimization online, we precompute the optimized FEC parameters and store the result table for online look-up. At our quantization granularity of the table, it takes 1.98MB to store the table, which is negligible on servers since the table is static and could be shared by all connections. Moreover, according to our measurements, the time of looking up the table is always less than 1ms, which is also negligible since the table is looked up at the granularity of the frame.

5 Related Work

We first discuss the limitations in Appendix E, and then discuss some pieces of related work here.

Deadline-aware optimization. Optimizing transport protocols for deadline-aware flows has been intensively studied in the networking community. Research efforts have been devoted to the optimization of transport protocols under the assumption of delivery deadline in video streaming [72, 87], space network [73], and others. There are also deadline-aware optimization in datacenters for flow completion time [30,78,82] and job completion time [36]. These research efforts mainly focus on the priority-aware scheduling between different packets, jobs, or flows. Instead, Hairpin is orthogonal to them and optimizes within one stream.

Loss recovery optimization. There are many previous research efforts in individually optimizing the retransmission mechanisms [21,62,68] or redundancy strategies [5,22,29,40,51,59,61]. Even for the joint optimization of redundancy and retransmission, we are not the first work to propose similar strategies. In wireless communications, there are already previous efforts in the joint optimization of redundancy and retransmission [12, 60]. There are also researches trying to combine retransmission and redundancy (e.g., WebRTC$_{2011}$ [71] and also [16, 37, 86]). However, as discussed in §2.4 and evaluated in §4, without optimizing redundancy over retransmissions, the *tail* performance is far from satisfactory for edge-based interactive streaming. Besides, there are also research efforts trying to adopt loss-resilient video codec [41, 80], which are unfortunately not deployable for services in the wild due to their hardware support. To the best of our knowledge, Hairpin is the first work to (i) jointly optimize the retransmission and redundancy towards the tail performance, and (ii) deploy in a real interactive streaming application in production. There are also researches to reduce packet losses by manipulating sending patterns [20, 38], which can work together with Hairpin.

Low-latency interactive streaming. Finally, as an emerging direction, low-latency interactive streaming also attracts much attention. At the transport layer, intensive efforts have also been devoted to the optimization of CCAs [26, 41, 48, 90], or the cross-layer optimization for interactive streaming to link layer [31]. As for the infrastructure, recent efforts propose to introduce edge computing for shorter latency [58, 77, 89], which are orthogonal to Hairpin. In contrast, Hairpin is inserted between the transport layer and the application layer, and is designed to optimize the redundancy of edge-based interactive streaming. In the evaluation and deployment of Hairpin (§4), we have already integrated Hairpin with some efforts above for a better user experience. Moreover, there are other research efforts [34, 84] that optimize other application metrics for interactive video streaming in image or video quality (e.g., SSIM [81] or PSNR [6]), which are orthogonal to the interaction delay (the delay for each frame) we focus on in this paper – they focus on the sharpness of the video but we focus on the interaction lag that users may have

6 Conclusion

We propose Hairpin, a packet loss recovery mechanism for edge-based interactive streaming to differentiate retransmissions and jointly optimize redundancy with retransmissions. Hairpin motivates the joint optimization with real-world measurements, and optimizes the redundancy and retransmissions with Markov decision process. Both trace-driven simulations and real-world deployments show that the joint optimization significantly reduces the DMR and BWC compared with state-of-the-art solutions.

This work does not raise any ethical issues.

Acknowledgements

We sincerely thank our shepherd Junchen Jiang, anonymous reviewers, and labmates in the Routing Group from Tsinghua University for their valuable feedback. This work is sponsored by the National Natural Science Foundation of China (No. 62221003 and 62372261) and the Tsinghua-Tencent Collaborative Grant. Bo Wang and Mingwei Xu are the corresponding authors.

References

[1] Pareto front - wikipedia. https://en.wikipedia.org/wiki/Pareto_front.

[2] 5g can make remote driving a reality, telefonica and ericsson demostrate at mwc. https://www.telefonica.com/en/web/press-office/-/5g-can-make-remote-driving-a-reality-telefonica-and-ericsson-demostrate-at-mwc, 2017.

[3] Cloud gaming (beta) with xbox game pass — xbox. https://www.xbox.com/en-US/xbox-game-pass/cloud-gaming, 2020.

[4] Critical services report: Video conferencing (uk) — blog. https://samknows.com/blog/critical-services-report-video-conferencing-uk, 2020.

[5] Issue 93006: Update to media_opt_util: - code review. https://webrtc-codereview.appspot.com/93006, 2020.

[6] Peak signal-to-noise ratio - wikipedia. https://en.wikipedia.org/wiki/Peak_signal-to-noise_ratio, 2020.

[7] Psa: Webrtc m88 release notes. https://groups.google.com/g/discuss-webrtc/c/A0FjOcTW2c0/m/UAv-veyPCAAJ, 2020.

[8] Stadia - one place for all the ways we play. https://stadia.google.com/, 2020.

[9] Your games. your devices. play anywhere — nvidia geforce now. https://www.nvidia.com/en-us/geforce-now/, 2020.

[10] Finite element method - wikipedia. https://en.wikipedia.org/wiki/Finite_element_method, 2021.

[11] Optimizing 5g for a new class of low-latency experiences [video]. https://www.qualcomm.com/news/onq/2021/07/20/optimizing-5g-new-class-low-latency-experiences, 2021.

[12] Ashfaq Ahmed, Arafat Al-Dweik, Youssef Iraqi, Husameldin Mukhtar, Muhammad Naeem, and Ekram Hossain. Hybrid automatic repeat request (harq) in wireless communications systems and standards: A contemporary survey. *IEEE Communications Surveys & Tutorials*, 2021.

[13] Amit. Huawei news — huawei launched cloud mobile phone. https://www.huaweiupdate.com/huawei-launched-cloud-mobile-phone/, 2020.

[14] ArsTechnica. Nvidia gtx 1080 review: The new performance king. https://arstechnica.com/gadgets/2016/05/nvidia-gtx-1080-review/4/, 2016.

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

[16] Luca Baldantoni, Henrik Lundqvist, and Gunnar Karlsson. Adaptive end-to-end fec for improving tcp performance over wireless links. In *Proc. IEEE ICC*, 2004.

[17] Matthew Ball and Jacob Navok. Challenge #3: Enormous bandwidth costs and operational burdens — cloud gaming: Why it matters and the games it will create. https://www.matthewball.vc/all/cloudmiles, 2020.

[18] Asha Barbaschow. Alibaba unveils cloud 2.0, wuying cloud computer, and xiaomanlv logistics robot. https://www.zdnet.com/article/alibaba-unveils-cloud-2-0-wuying-cloud-computer-and-xiaomanlv-logistics-robot/, 2020.

[19] Apurv Bhartia, Bo Chen, Feng Wang, Derrick Pallas, Raluca Musaloiu-E, Ted Tsung-Te Lai, and Hao Ma. Measurement-based, practical techniques to improve 802.11 ac performance. In *Proc. ACM IMC*, 2017.

[20] Saad Biaz and Nitin H Vaidya. "de-randomizing" congestion losses to improve tcp performance over wired-wireless networks. *IEEE/ACM Transactions on Networking*, 2005.

[21] Ethan Blanton, Dr. Vern Paxson, and Mark Allman. TCP Congestion Control. IETF RFC 5681, 2009.

[22] J-C Bolot, Sacha Fosse-Parisis, and Don Towsley. Adaptive fec-based error control for internet telephony. In *Proc. IEEE INFOCOM*, 1999.

[23] brianhu. Google meet troubleshooting playbook - network and hardware troubleshooting. https://www.googlecloudcommunity.com/gc/Workspace-Product-Articles/Google-Meet-Troubleshooting-Playbook-Network-and-Hardware/ta-p/165810, 2021.

[24] James Bulman and Peter Garraghan. A cloud gaming framework for dynamic graphical rendering towards achieving distributed game engines. In *Proc. USENIX HotCloud*, 2020.

[25] Neal Cardwell, Yuchung Cheng, C Stephen Gunn, Soheil Hassas Yeganeh, and Van Jacobson. Bbr: Congestion-based congestion control. *ACM Queue*, 2016.

[26] Gaetano Carlucci, Luca De Cicco, Stefan Holmer, and Saverio Mascolo. Congestion control for web real-time communication. *IEEE/ACM Transactions on Networking*, 2017.

[27] Marc Carrascosa and Boris Bellalta. Cloud-gaming: Analysis of google stadia traffic. *arXiv:2009.09786*, 2020.

[28] Hao Chen, Xu Zhang, Yiling Xu, Ju Ren, Jingtao Fan, Zhan Ma, and Wenjun Zhang. T-gaming: A cost-efficient cloud gaming system at scale. *IEEE Transactions on Parallel and Distributed Systems*, 2019.

[29] Ke Chen, Han Wang, Shuwen Fang, Xiaotian Li, Minghao Ye, and H. Jonathan Chao. Rl-afec: Adaptive forward error correction for real-time video communication based on reinforcement learning. In *Proc. ACM MMSys*, 2022.

[30] Li Chen, Kai Chen, Wei Bai, and Mohammad Alizadeh. Scheduling mix-flows in commodity datacenters with karuna. In *Proc. ACM SIGCOMM*, 2016.

[31] Wei Chen, Liangping Ma, and Chien-Chung Shen. Congestion-aware mac layer adaptation to improve video teleconferencing over wi-fi. In *Proc. ACM MMSys*, 2015.

[32] Sheng Cheng, Han Hu, Xinggong Zhang, and Zongming Guo. Rebuffering but not suffering: Exploring continuous-time quantitative qoe by user's exiting behaviors. In *Proc. IEEE INFOCOM*, 2023.

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

[34] Mallesham Dasari, Kumara Kahatapitiya, Samir R. Das, Aruna Balasubramanian, and Dimitris Samaras. Swift: Adaptive video streaming with layered neural codecs. In *Proc. USENIX NSDI*, 2022.

[35] Andrea Di Domenico, Gianluca Perna, Martino Trevisan, Luca Vassio, and Danilo Giordano. A network analysis on cloud gaming: Stadia, geforce now and psnow. *Network*, 2021.

[36] Andrew D Ferguson, Peter Bodik, Srikanth Kandula, Eric Boutin, and Rodrigo Fonseca. Jockey: guaranteed job latency in data parallel clusters. In *Proc. ACM EuroSys*, 2012.

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

[38] Tobias Flach, Pavlos Papageorge, Andreas Terzis, Luis Pedrosa, Yuchung Cheng, Tayeb Karim, Ethan Katz-Bassett, and Ramesh Govindan. An internet-wide analysis of traffic policing. In *Proc. ACM SIGCOMM*, 2016.

[39] Mary Jo Foley. Microsoft marches toward launching its 'cloud pc' service, possibly this summer. `https://www.zdnet.com/article/microsoft-marches-toward-launching-its-cloud-pc-service-possibly-this-summer/`, 2021.

[40] Silas L Fong, Ashish Khisti, Baochun Li, Wai-Tian Tan, Xiaoqing Zhu, and John Apostolopoulos. Optimal streaming codes for channels with burst and arbitrary erasures. *IEEE Transactions on Information Theory*, 2019.

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

[42] GFXBench. 3d graphics performance of google pixel c. `https://gfxbench.com/device.jsp?D=Google+Pixel+C`, 2017.

[43] Moinak Ghoshal, Pranab Dash, Zhaoning Kong, Qian Xu, Y.Charlie Hu, Dimitrios Koutsonikolas, and Yuanjie Li. Can 5g mmwave enable multi-user ar apps? In *Proc. PAM*, 2022.

[44] Yu Guan, Chengyuan Zheng, Xinggong Zhang, Zongming Guo, and Junchen Jiang. Pano: Optimizing 360 video streaming with a better understanding of quality perception. In *Proc. ACM SIGCOMM*, 2019.

[45] Osama Haq, Mamoon Raja, and Fahad R Dogar. Measuring and improving the reliability of wide-area cloud paths. In *Proc. WWW*, 2017.

[46] Stefan Holmer, Mikhal Shemer, and Marco Paniconi. Handling packet loss in webrtc. In *2013 IEEE International Conference on Image Processing*, 2013.

[47] Manish Jain and Constantinos Dovrolis. End-to-end available bandwidth: Measurement methodology, dynamics, and relation with tcp throughput. In *Proc. ACM SIGCOMM*, 2002.

[48] Ingemar Johansson and Zaheduzzaman Sarker. Self-Clocked Rate Adaptation for Multimedia. IETF RFC 8298, 2017.

[49] Alan Jones, Peter Sevcik, and Rebecca Wetzel. Internet connection requirements for effective video conferencing to support work from home and elearning — netforecast. `https://www.netforecast.com/wp-content/uploads/NFR5137-Videoconferencing_Internet_Requirements.pdf`, 2021.

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

[51] Balázs Kreith, Varun Singh, and Jörg Ott. Fractal: Fec-based rate control for rtp. In *Proc. ACM Multimedia*, 2017.

[52] Tong Li, Kai Zheng, Ke Xu, Rahul Arvind Jadhav, Tao Xiong, Keith Winstein, and Kun Tan. Tack: Improving wireless transport performance by taming acknowledgments. In *Proc. ACM SIGCOMM*, 2020.

[53] Xiaofei Liao, Li Lin, Guang Tan, Hai Jin, Xiaobin Yang, Wei Zhang, and Bo Li. Liverender: A cloud gaming system based on compressed graphics streaming. *IEEE/ACM Transactions on Networking*, 2016.

[54] Ruilin Liu, Daehan Kwak, Srinivas Devarakonda, Kostas Bekris, and Liviu Iftode. Investigating remote driving over

the lte network. In *Proceedings of the 9th International Conference on Automotive User Interfaces and Interactive Vehicular Applications*, pages 264–269, 2017.

[55] Bill Marczak and John Scott-Railton. Move fast and roll your own crypto: A quick look at the confidentiality of zoom meetings - the citizen lab. https://citizenlab.ca/2020/04/move-fast-roll-your-own-crypto-a-quick-look-at-the-confidentiality-of-zoom-meetings/, 2020.

[56] Zili Meng, Yaning Guo, Chen Sun, Bo Wang, Justine Sherry, Hongqiang Harry Liu, and Mingwei Xu. Achieving Consistent Low Latency for Wireless Real Time Communications with the Shortest Control Loop. In *Proc. ACM SIGCOMM*, 2022.

[57] Zili Meng, Tingfeng Wang, Yixin Shen, Bo Wang, Mingwei Xu, Rui Han, Honghao Liu, Venkat Arun, Hongxin Hu, and Xue Wei. Enabling High Quality Real-Time Communications with Adaptive Frame-Rate. In *Proc. USENIX NSDI*, 2023.

[58] China Mobile and ZTE. Powered by sa: 5g mec-based cloud game innovation practice. GSMA 5G Case Studies (https://www.gsma.com/futurenetworks/wp-content/uploads/2020/03/Powered-by-SA-5G-MEC-Based-Cloud-Game-Innovation-Practice-.pdf), 2020.

[59] Marcin Nagy, Varun Singh, Jörg Ott, and Lars Eggert. Congestion control using fec for conversational multimedia communication. In *Proc. ACM MMSys*, 2014.

[60] Hoang Anh Ngo and Lajos Hanzo. Hybrid automatic-repeat-request systems for cooperative wireless communications. *IEEE Communications Surveys & Tutorials*, 16(1):25–45, 2013.

[61] Chinmay Padhye, Kenneth J Christensen, and Wilfrido Moreno. A new adaptive fec loss control algorithm for voice over ip applications. In *Proc. IEEE INFOCOM*, 2000.

[62] Colin Perkins, Orion Hodson, and Vicky Hardman. A survey of packet loss recovery techniques for streaming audio. *IEEE Network*, 1998.

[63] Feng Qian, Bo Han, Qingyang Xiao, and Vijay Gopalakrishnan. Flare: Practical viewport-adaptive 360-degree video streaming for mobile devices. In *Proc. ACM Mobicom*, 2018.

[64] Vincent Roca, Mathieu Cunche, Jerome Lacan, Amine Bouabdallah, and Kazuhisa Matsuzono. Simple Reed-Solomon Forward Error Correction (FEC) Scheme for FECFRAME. IETF RFC 6865, 2013.

[65] Vincent Roca, Jani Peltotalo, Jerome Lacan, and Sami Peltotalo. Reed-Solomon Forward Error Correction (FEC) Schemes. IETF RFC 5510, 2009.

[66] Carolyn Rowe, Diana Hanson, Chiffers Craig, David Coulter, Justin Gilmore, David Byrd, Ajayan Borys, Kelly Baker, Baard Hermansen, Serdar Soysal, et al. Microsoft teams call flows - microsoft teams — microsoft docs. https://docs.microsoft.com/en-us/microsoftteams/microsoft-teams-online-call-flows, 2021.

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

[68] Pasi Sarolahti, Markku Kojo, and Kimmo Raatikainen. F-rto: an enhanced recovery algorithm for tcp retransmission timeouts. *ACM SIGCOMM Computer Communication Review*, 2003.

[69] Henning Schulzrinne, Stephen L. Casner, Ron Frederick, and Van Jacobson. RTP: A Transport Protocol for Real-Time Applications. IETF RFC 3550, 2003.

[70] Heiko Schwarz, Detlev Marpe, and Thomas Wiegand. Overview of the scalable video coding extension of the h. 264/avc standard. *IEEE Transactions on circuits and systems for video technology*, 2007.

[71] Mikhal Shemer. Commit - video_coding robustness: Updating hybrid mode's settings — google git. https://webrtc.googlesource.com/src/+/ae7a0522c59d932d72f3d3377c38bebab7ab2b31%5E%21/, 2011.

[72] Hang Shi, Yong Cui, Feng Qian, and Yuming Hu. Dtp: Deadline-aware transport protocol. In *Proc. APNet*, 2019.

[73] Hang Shi, Lei Zhang, Xutong Zuo, Qian Wu, Hewu Li, and Yong Cui. Multipath deadline-aware transport proxy for space network. *IEEE Internet Computing*, 2021.

[74] Shu Shi, Cheng-Hsin Hsu, Klara Nahrstedt, and Roy Campbell. Using graphics rendering contexts to enhance the real-time video coding for mobile cloud gaming. In *Proc. ACM Multimedia*, 2011.

[75] Ivan Slivar, Lea Skorin-Kapov, and Mirko Suznjevic. Cloud gaming qoe models for deriving video encoding adaptation strategies. In *Proceedings of ACM International Conference on Multimedia Systems (MMSys)*, 2016.

[76] Kaarmukilan S.P. What is hairpin net shot in badminton? - quora. https://www.quora.com/What-is-hairpin-net-shot-in-badminton/answer/Kaarmukilan-S-P, 2020.

[77] Zhaowei Tan, Yuanjie Li, Qianru Li, Zhehui Zhang, Zhehan Li, and Songwu Lu. Supporting mobile vr in lte networks: How close are we? *Proc. ACM SIGMETRICS*, 2018.

[78] Balajee Vamanan, Jahangir Hasan, and TN Vijaykumar. Deadline-aware datacenter tcp (d2tcp). In *Proc. ACM SIGCOMM*, 2012.

[79] Martijn van Otterlo and Marco Wiering. *Reinforcement Learning and Markov Decision Processes*, pages 3–42. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[80] Tingfeng Wang, Zili Meng, Mingwei Xu, Rui Han, and Honghao Liu. Enabling high frame-rate uhd real-time communication with frame-skipping. In *Proc. ACM Workshop on Hot Topics in Video Analytics and Intelligent Edges*, 2021.

[81] Zhou Wang, Alan C Bovik, Hamid R Sheikh, and Eero P Simoncelli. Image quality assessment: from error visibility to structural similarity. *IEEE Transactions on Image Processing*, 2004.

[82] Christo Wilson, Hitesh Ballani, Thomas Karagiannis, and Ant Rowtron. Better never than late: Meeting deadlines in datacenter networks. In *Proc. ACM SIGCOMM*, 2011.

[83] Raphael Wimmer, Andreas Schmid, and Florian Bockes. On the latency of usb-connected input devices. In *Proc. ACM CHI*, pages 1–12, 2019.

[84] Saman Zadtootaghaj, Steven Schmidt, Saeed Shafiee Sabet, Sebastian Möller, and Carsten Griwodz. Quality estimation models for gaming video streaming services using perceptual video quality dimensions. In *Proc. ACM MMSys*, 2020.

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

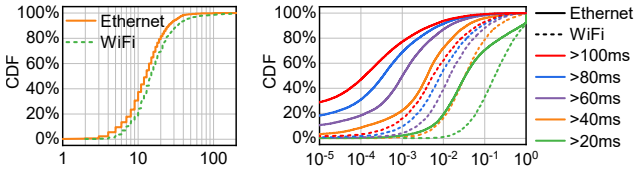
[86] Fan Zhai, Yiftach Eisenberg, Thrasyvoulos N Pappas, Randall Berry, and Aggelos K Katsaggelos. Rate-distortion optimized hybrid error control for real-time packetized video transmission. *IEEE Transactions on Image Processing*, 2006.

[87] Lei Zhang, Yong Cui, Junchen Pan, and Yong Jiang. Deadline-aware transmission control for real-time video streaming. In *Proc. IEEE ICNP*, 2021.

[88] Songyang Zhang. Soonyangzhang/webrtc-gcc-ns3: test google congestion control on ns3. `https://github.com/SoonyangZhang/webrtc-gcc-ns3`, 2020.

[89] Xu Zhang, Hao Chen, Yangchao Zhao, Zhan Ma, Yiling Xu, Haojun Huang, Hao Yin, and Dapeng Oliver Wu. Improving cloud gaming experience through mobile edge computing. *IEEE Wireless Communications*, 2019.

[90] Xiaoqing Zhu, Rong Pan, Michael A. Ramalho, and Sergio Mena de la Cruz. Network-Assisted Dynamic Adaptation (NADA): A Unified Congestion Control Scheme for Real-Time Media. IETF RFC 8698, 2020.



(a) Frame-level delay (b) Session-level delay ratio

Figure 18: Network delay distributions of the interactive streaming service of company T. Delay ratio is the ratio of frames with a delay of >20,>40,>60,>80 and >100 ms in each session. Note that the delay here is measured at the application layer (details in §4.2).

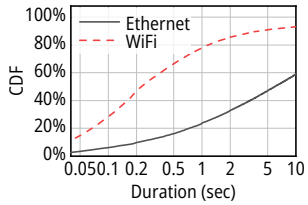


Figure 19: Distribution of network RTT maintenance duration in our interactive streaming service.

Appendices

A Measurements in Production

We present our measurement results on the cloud gaming service X in production to support some claims in the paper.

To investigate the effect of edge acceleration of interactive streaming in the wild, we conduct a measurement campaign on the cloud gaming service X. The measurements last for one week with thousands of sessions (containing heterogeneous users through Ethernet, WiFi with Windows and MacOS systems) and are presented in Fig. 18. As shown in Fig. 18(a), the majority of network delay collected at the granularity of video frame falls into 10-20ms for both Ethernet and WiFi. We also measure the flow-level delay ratio at different thresholds and present the results in Fig. 18(b). With the edge acceleration, the ratio of frames with longer than 100ms delay in most flows is less than 10^{-2}. Among them, Ethernet flows perform slightly better than WiFi flows. This validates the effectiveness of edge acceleration: the average network delay could be reduced to 10-20ms with a proper edge acceleration.

We further measure the fluctuation of RTT by the duration when RTT is roughly kept at the same level. We quantify it by calculating the transmission chance (i.e., layer L) for the RTT measured by each frame, and calculate the duration when the chance is kept the same. For example, given a deadline of 100ms in this paper, when the RTT measurements are [26ms, 18ms, 17ms, 22ms, 17ms, 19ms, 19ms], the transmission chances are [3, 5, 5, 4, 5, 5, 5]. In this case, the durations of each transmission chance are [1, 2, 1, 3], which are denoted as *RTT maintenance durations*. We present the distribution of RTT maintenance duration measured in our cloud gaming service in Fig. 19. The RTT maintenance duration of Ethernet is much longer than that of WiFi, indicating that Ethernet has a more stable end-to-end delay. Meanwhile, the median duration of both Ethernet and WiFi is above hundreds of milliseconds, which is

Notation	Explanation
Inputs:	
α	Network loss rate.
T	Remaining time till the deadline.
RTT	The network round-trip time.
Θ	The network bottleneck bandwidth.
F	The frame size of that frame.
Intermediate variables:	
L	Remaining transmission chance.
$l(n,r)$	The number of lost packets at the r-th layer with n data packets.
$k(n,r)$	The number of redundant packets at the r-th layer with n data packets.
DMR	Deadline miss rate.
BWC	Bandwidth cost.
Outputs:	
β_i	Redundancy rate at the i-th layer.
b_i	FEC block size at the i-th layer.

Table 2: Notations in §3 and Appendix B.

much higher than the feedback loop of Hairpin. This indicates that *RTT does not frequently change, and Hairpin is able to detect and react to the fluctuations of RTT*.

B Optimization Model

In this section, we present the notations used in the Markov chain in §3.3. We list all notations in Table 2. We further present the detailed designs here.

B.1 Optimization of the Redundancy Rate

We build an absorbing Markov chain to model the redundancy and calculate the deadline miss rate considering retransmission, as shown in Fig. 8. We first define the *state* in the Markov chain as (n,r), where n is the number of unacknowledged packets within the block, and r is the number of retransmission. For example, $(d_0,0)$ represents the initial transmission where all d_0 packets have not been received before (since it is the first time of transmission). $(3,2)$ denotes that there are still 3 data packets that need to be retransmitted for the second time.

For node (d,l), given redundancy rate β, its DMR follows:

$$DMR(d,l;\beta)=\sum_{d'=0}^{d} p((d,l)\to(d',l-1);\beta)\cdot DMR(d',l-1) \quad (6)$$

where $p((d,l)\to(d',l-1);\beta)$ is the transition probability from (d,l) to $(d',l-1)$ and could be calculated based on the current loss rate α and redundancy rate β. Similarly, the BWC could also be updated as:

$$BWC(d,l;\beta)=\beta\frac{d}{F}+\sum_{d'=0}^{d} p((d,l)\to(d',l-1);\beta)\cdot BWC(d',l-1) \quad (7)$$

where the latter term is the additional BWC introduced in this layer l.

We then calculate the transition probability between states in the Markov chain. For the transition between state (n_1,r) to $(n_2,r-1)$, we know that n_2 data packets are lost in the r-th transmission and need to be transmitted for the $(r+1)$ time. We first discuss the scenario of $n_2>0$. We denote the total number of packet losses (including data and redundancy) in the r-th transmission as $l(n_1,r)$. We denote the number of redundant packets in the r-th transmission as $k(n_1,r)$. Since the packet losses of all packets should not be less than the packet losses of data packets, we have $l(n_1,r)\geqslant n_2$. Meanwhile, since there are only $k(n_1,r)$ redundant packets in total, we have $l(n_1,r)\leqslant n_2+k(n_1,r)$. We also have $l(n_1,r)>k(n_1,r)$, otherwise the lost packets could be recovered with FEC. Therefore, the probability of n_2 data packet losses under the condition of $l(n_1,r-1)$ total packet losses follows the hypergeometric distribution:

$$\begin{aligned}&H(n_2;n_1+k(n_1,r-1),n_1,l(n_1,r))\\&\quad=\binom{n_1}{n_2}\binom{k(n_1,r)}{l(n_1,r)-n_2}\Big/\binom{n_1+k(n_1,r)}{l(n_1,r)}\end{aligned} \quad (8)$$

Thus, the transition probability from (n_1,r) to $(n_2,r-1)$ is:

$$\begin{aligned}&p((n_1,r)\to(n_2,r-1))=\\&\sum_{l(n_1,r)} H(n_2;n_1+k(n_1,r),n_1,l(n_1,r))\cdot P(l(n_1,r)\text{ losses})\end{aligned} \quad (9)$$

On the other hand, at the loss rate of α, losing $l(n_1,r)$ packets in all $n_1+k(n_1,r)$ packets follows the Binomial distribution:

$$\begin{aligned}&P(l(n_1,r)\text{ losses})=Bi(l(n_1,r);n_1+k(n_1,r),\alpha)=\\&\quad\binom{n_1+k(n_1,r)}{l(n_1,r)}\alpha^{l(n_1,r)}(1-\alpha)^{n_1+k(n_1,r)-l(n_1,r)}\end{aligned} \quad (10)$$

Therefore, by substituting Eq. 8 and 10 into Eq. 9, we can have the transition probability for $n_2>0$. Similarly, when state transits from (n_1,r) to $(0,r-1)$, then the number of lost packets in the r-th layer of Fig. 8 must be less than $k(n_1,r)$. Therefore, the transition probability satisfies:

$$p((n_1,r)\to(0,r-1))=\sum_{i=0}^{k(n_1,r)} Bi(i;n_1+k(n_1,r),\alpha) \quad (11)$$

B.2 Optimization of Block Size

In the following analysis, we are going to compare the utility of transmitting the whole frame for L chances, or splitting the frame into several blocks and some of them enjoying L+1 chances. With that, we assume that the dispersion is less than one RTT.

Therefore, when the block size is set to d, there are N_{L+1} blocks that could enjoy L+1 chances of transmission, and the remaining N_L blocks with L chances of transmission, where

$$\begin{aligned}N_{L+1}&=\left\lfloor\frac{DDL-(L+1)\cdot RTT}{d/\Theta}\right\rfloor\\N_L&=\left\lceil\frac{F}{d}\right\rceil-N_{L+1}\end{aligned} \quad (12)$$

Therefore, the on-time delivery of the frame requires the on-time delivery of each block. Since the deadline miss rate is equal to one minus the probability of on-time delivery, we have the frame

DMR (FDMR) given a certain block size d as:

$$\begin{aligned} 1-FDMR(d) &= (1-DMR(L+1,d))^{N_{L+1}} \cdot (1-DMR(L,d))^{N_L} \\ \Rightarrow FDMR(d) &= 1-(1-DMR(L+1,d))^{N_{L+1}} \cdot (1-DMR(L,d))^{N_L} \\ &= N_{L+1} \cdot DMR(L+1,d) + N_L \cdot DMR(L,d) \end{aligned} \tag{13}$$

where the last equation holds since $DMR(L,d) \ll 1$ and $(1-\alpha)^n = 1-n\alpha$ when $\alpha \ll 1$. As for the bandwidth cost, recalling Eq. 7, the number of extra packets of the frame is the sum of the number of extra packets for each block. Since the BWC of each block shares the same denominator (frame size S), the frame BWC is also the sum of BWC of each block:

$$FBWC(d) = N_{L+1} \cdot BWC(L+1,d) + N_L \cdot BWC(L,d) \tag{14}$$

Therefore, the optimal block size is:

$$d_{opt} = \underset{d}{\operatorname{argmax}}\ utility(FDMR(d), FBWC(d)) \tag{15}$$

In our implementation, we iterate the possible block size B from 1 to the frame size S, and store the optimal block size in each scenario in an offline lookup table. Since the $DMR(L,B)$ and $BWC(L,B)$ are accessible in the absorbing Markov chain constructed above, the construction of the table is time-efficient.

C Implementation Details

We are going to introduce the sending mechanism beneath Hairpin and the implementation of the redundancy optimization in Hairpin.

Acknowledgement aggregation. In wireless networks, researchers also propose to aggregate several acknowledgements at the client side to alleviate the uplink interference [52]. However, the delayed acknowledgement might also interfere with the measurements of RTT, delay the detection of packet losses and waste potential chances of retransmission. In our implementation, to eliminate the interference from acknowledgement mechanisms, we disable the aggregation of acknowledgements. The precise measurement of RTT in the scenario of aggregated acknowledgement could also be implemented with recent efforts such as TACK [52], which is out of our scope.

Note that this is different from the aggregation on wireless routers [19]. Such aggregations due to wireless channel competition should be reflected in our measurements of network RTT fluctuations in Fig. 4. In our simulation with online measurements and deployments in production, Hairpin behaves well even with the RTT fluctuations.

FEC codec. For the scenarios with a redundancy rate of $\leqslant$100%, we implement the FEC codec as RS-FEC, as suggested by many other related efforts [65]. We refer the readers to [65] for the details of RS-FEC. However, when implementing the redundancy rate of >100%, RS-FEC is not designed to reliably recover lost packets in all cases. For example, when there are 2 data packets and 4 FEC packets, RS-FEC cannot always recover 2 data packets when there are 4 packet losses due to the invertibility of the decoding matrix: it depends on whether two packets received at the client are linearly independent at the generation matrix.

Therefore, we implement a customized FEC codec. For example, for data packets a and b, when considering them as two numbers (with a length of up to 12kbits), we could calculate $a+b$, $a+2b$, $2a+b$, etc., and send them to the client. The only overhead is the additional bits that could overflow from the addition, which is much less than the data bits. Moreover, as shown in Fig. 25, in most cases the redundancy rate is less than 100%. Therefore, the overall decoding overhead is also acceptable. We leave the further adoption of advanced FEC codec when the redundancy rate is >100% as our future work.

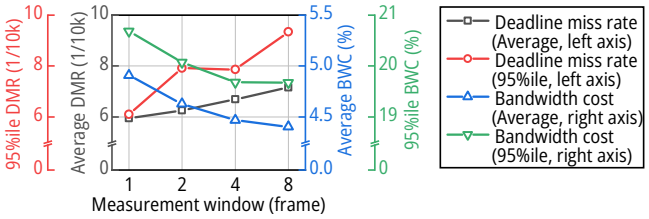


Figure 20: Sensitivity of the measurement window in §3.4.

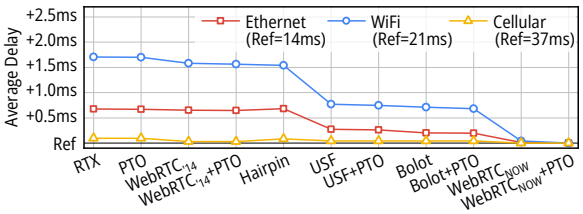


Figure 21: Average end-to-end delay of in the experiments in §4.3. We trim the lowest average delay in different traces for comparison.

D Supplementary Experiments

Measurement window. We also evaluate the performance of Hairpin by adjusting the measurement window of the network conditions that we discussed in §3.4. Since Hairpin optimizes the redundancy parameters based on real-time measurements of the network conditions, the size of the measurement window might affect the performance of Hairpin. We vary the measurement window from the last 1 to 8 frames and reconduct the experiments over WiFi traces. We measure the average and 95th percentile DMR and BWC, and present the results in Fig. 20. The DMR and BWC are quite robust: By varying the measurement window from 1 to 8, the average DMR and average BWC vary within 0.47%-0.49% and 6.94%-7.19%, which is subordinate to the improvements in §4.3 (Fig. 11(b)). In practice, operators can decide the measurement window based on the fluctuations of network conditions.

Per-frame latency of Hairpin. Besides, we also measure the average end-to-end delay for the successfully delivered frames in the experiments in §4.3 for Hairpin and different baselines. As shown in Fig. 21, the average end-to-end delay of Hairpin does increase compared to the baseline with the lowest average delay. However, the increase is only 0.1-1.5ms for all traces, which is negligible compared with the RTT (1%-7%), and considering

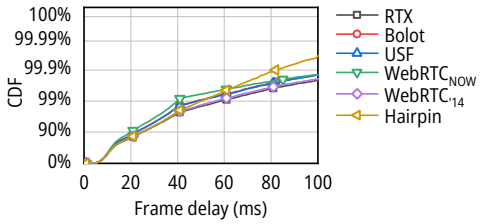
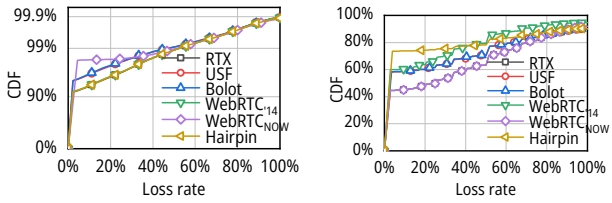


Figure 22: The distribution of the delivery time of each frame. Note that the y-axis is log-scaled.



(a) Initial transmission (b) The 3rd retransmission

Figure 23: Distribution of loss rates by frame in each round of transmission.

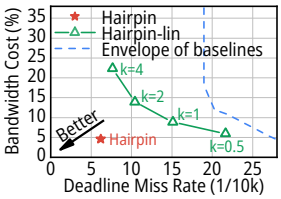


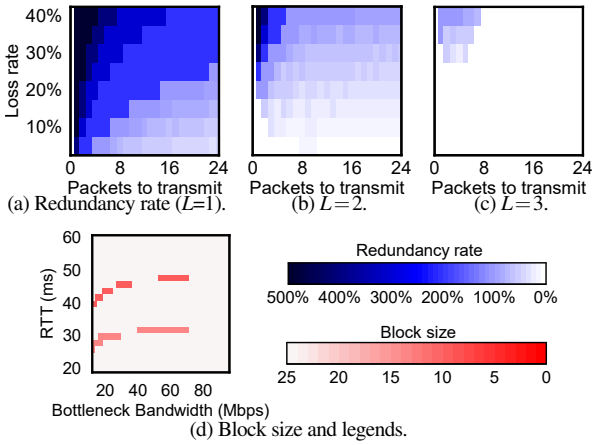
Figure 24: Heuristic-based Hairpin (Hairpin-lin). The envelope of baselines is from Figure 11(b).

the deadline effect we discussed in §2.1. Furthermore, operators could also adopt less aggressive mappings (e.g., increasing λ) to tradeoff between the tail delay and average delay.

We also present the distribution of the delay of each frame in Fig. 22. Similar to Fig. 21, the average (median) latency of frames of Hairpin is similar to other baselines. However, Hairpin could reduce the tail latency significantly. For example, Hairpin can reduce the 99.9th percentile frame latency to 80ms while all baselines of longer than 100ms. Looking at the vertical axis, Hairpin is also capable of reducing the ratio of higher than 100ms by more than a half, as shown in Fig. 11(b).

Loss rates in each round. We further present the distributions of loss rates of all frames in each round (specifically, initial transmission and the third retransmission) in Fig. 23. This expands the results in Fig. 15(b). We can tell from Fig. 23(a) that due to the conservative redundancy strategy of Hairpin, the loss rate of Hairpin is higher. However, when retransmission starts, Hairpin is able to maintain a low loss rate – which means a high success rate in delivering frames – compared to other baselines. This shows the strategy of Hairpin: conservatively adding FEC packets when deadline is far away, and aggressively adding FEC packets to retransmissions.

The improvements of using Markov chain. As we analyzed in §3.2, a strawman solution is good but not enough to fully utilize the design space of redundancy and retransmission. Thus, we also evaluate the heuristic baseline we present in §3.1 (denoted



(a) Redundancy rate (L=1). (b) L=2. (c) L=3.
(d) Block size and legends.

Figure 25: Optimization results by Hairpin. Fig. 25(a) to 25(c) present the redundancy rate with different transmission chances L.

as Hairpin-lin, with sweeping the coefficient k from 0.5 to 4, and present the results in Figure 24. As we can see, Hairpin-lin (green line) does improve the trade-off compared to existing baselines (dashed blue line). Yet, there is still a half gap between Hairpin-lin and the Markov chain-based Hairpin (the red star). Therefore, it is necessary to analytically formulate the problem with the Markov chain to further push the trade-off forward.

Understanding Hairpin's decisions. We further present the redundancy rate results of Hairpin to provide a deeper understanding of how Hairpin optimizes in different scenarios. For redundancy rate, since the optimization of the absorbing Markov chain (§3.2) relies on the remaining transmission chance L, loss rate, remaining data packets to transmit, and the frame size, we present the optimized redundancy rate over different parameters in Fig. 25(a) to 25(c). With more transmission chances, Hairpin would decrease the redundancy rate and rely on retransmissions for packet loss recovery. With fewer packets to retransmit, Hairpin also prefers a higher redundancy rate, as discussed in §3.2. Moreover, when the number of packets to transmit is small, the optimized redundancy rate is up to 500% in Fig. 25(a), demonstrating the effectiveness of a redundancy rate of >100%.

As for the optimization of block size, as we also discussed in §3.2, the optimal block size is the frame size (24 packets) in many cases. Nevertheless, as we discussed, at the decision boundary of remaining transmission chance, smaller block sizes do enjoy a slightly better performance by having additional transmission chances. As shown in Fig. 25(d), although the optimal block size is the frame size in most cases, when the RTT is around 33ms and 50ms (the dividing point between 1, 2, and 3 transmission chances), the optimal block size might be smaller than the frame size. For example, compared to setting the block size to the frame size, the DMR with the optimized block size of Hairpin could be further reduced by 1.78× around the RTT of 50ms and bottleneck bandwidth of 60Mbps. We optimize the block size for the last mile performance improvement.

E Limitations

Delay components in interactive streaming. Hairpin could have maximum benefits when the end-to-end network delay dominates the total delay from the video encoder to the decoder in Fig. 10. This is generally true in interactive streaming services. Related measurement studies also demonstrate that the network delay is still one of the bottlenecks of edge-based interactive streaming [43, 57]. Therefore, we focus on the optimization of streams between edge servers and clients. Our deployments in the wild demonstrate that optimizing the network latency could significantly improve the user's experience (note that DMR is measured end-to-end). Hairpin can also work with the optimization of other delay components (e.g., encoding, decoding, etc.) to further improve the performance.

Deployment efforts for applications. Another concern of deploying Hairpin is that both the server and the client need modification to support the redundancy and retransmissions. There are previous efforts implementing the FEC mechanism over TCP [16, 37], which needs to modify the TCP protocol stack at the client and are not suitable for products at scale. For scenarios where TCP is compulsory for transport, the deployment of Hairpin may depend on the ability to modify the reception mechanism of TCP packets at the client. However, most interactive streaming applications adopt UDP to reduce the network delay [23, 35, 55, 66], including our service. In this case, Hairpin could be implemented within the application at the server and the client, which is practical for most applications.