GRACE: Loss-Resilient Real-Time Video through Neural Codecs

Yihua Cheng[1], Ziyi Zhang[1], Hanchen Li[1], Anton Arapin[1], Yue Zhang[1], Qizheng Zhang[2], Yuhan Liu[1], Kuntai Du[1], Xu Zhang[1], Francis Y. Yan[3], Amrita Mazumdar[4], Nick Feamster[1], Junchen Jiang[1]
[1]*The University of Chicago,* [2]*Stanford University,* [3]*Microsoft,* [4]*NVIDIA*

Abstract

In real-time video communication, retransmitting lost packets over high-latency networks is not viable due to strict latency requirements. To counter packet losses without retransmission, two primary strategies are employed—encoder-based forward error correction (FEC) and decoder-based error concealment. The former encodes data with redundancy before transmission, yet determining the optimal redundancy level in advance proves challenging. The latter reconstructs video from partially received frames, but dividing a frame into independently coded partitions inherently compromises compression efficiency, and the lost information cannot be effectively recovered by the decoder without adapting the encoder.

We present a loss-resilient real-time video system called GRACE, which preserves the user's quality of experience (QoE) across a wide range of packet losses through a new neural video codec. Central to GRACE's enhanced loss resilience is its *joint training of the neural encoder and decoder under a spectrum of simulated packet losses*. In lossless scenarios, GRACE achieves video quality on par with conventional codecs (e.g., H.265). As the loss rate escalates, GRACE exhibits a more graceful, less pronounced decline in quality, consistently outperforming other loss-resilient schemes. Through extensive evaluation on various videos and real network traces, we demonstrate that GRACE reduces undecodable frames by 95% and stall duration by 90% compared with FEC, while markedly boosting video quality over error concealment methods. In a user study with 240 crowdsourced participants and 960 subjective ratings, GRACE registers a 38% higher mean opinion score (MOS) than other baselines. We make the source codes and models of GRACE public at https://uchi-jcl.github.io/grace.html.

1 Introduction

Real-time video communication has become an integral part of our daily lives [29], spanning online conferences [3, 20], cloud gaming [11, 17], interactive virtual reality [6, 18], and IoT applications [16, 19]. To ensure a high quality of experience (QoE) for users, real-time video applications must protect against packet losses[1]. However, retransmitting lost packets across high-latency networks is not feasible due to stringent real-time latency requirements [57].

[1]In this study, we use the term "packet loss" to refer to both packets dropped in transit and those not received by the decoding deadline. Under this definition, a video frame could experience a high packet loss rate (e.g., 50%) even if the actual network loss rate is low [86].

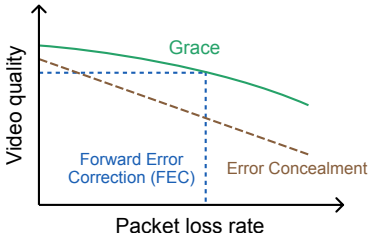


Figure 1: *Illustration of the video quality achieved by different loss-resilient schemes, operating under the same bandwidth budget, across varying packet loss rates. Actual experimental results are shown in Figure 8.*

Loss-resilient techniques generally fall into two categories. First is encoder-side forward error correction (FEC), such as Reed-Solomon codes [100], fountain codes [76, 77], and more recently, streaming codes [28, 86]. FEC incorporates redundancy into data prior to transmission. With a redundancy rate of $R\%$—the percentage of redundant data relative to the total data size—up to $R\%$ of lost data can be recovered. Beyond that, the video becomes undecodable, rendering a sharp collapse in video quality (Figure 1). Increasing R protects against higher losses but also entails a higher bandwidth overhead, which in turn reduces video quality. Thus, determining the optimal R in advance poses a practical challenge.

The second category is decoder-side error concealment, which reconstructs portions of a video frame affected by packet losses, through handcrafted heuristics [63, 97, 115] or neural networks [59, 67, 79, 87, 102]. Nevertheless, implementing error concealment requires partitioning a video frame into independently decodable units (e.g., slices [99] or tiles [64]) first, thus reducing compression efficiency. Moreover, since the encoder is not optimized for loss resilience, the lost information cannot be effectively recovered by the decoder alone. As a result, the video quality tends to deteriorate rapidly with increasing packet loss, as illustrated in Figure 1.

In this paper, we present GRACE, a loss-resilient real-time video system designed to maintain the user's quality of experience (QoE) across a wide range of packet losses. Our key insight is that *jointly optimizing the encoder and decoder under a spectrum of simulated packet losses* considerably strengthens loss resilience. To facilitate this joint optimization, GRACE strategically employs a neural video codec (NVC) [73], integrating neural networks into the core components of a conventional video encoder and decoder. In contrast to FEC, GRACE's NVC is trained to handle *diverse* packet losses, eliminating the need to predict a loss rate beforehand and preventing the undecodable video under exceedingly high

losses. Unlike decoder-side error concealment, GRACE *jointly* trains the encoder and decoder, so that the encoder learns to properly distribute each pixel's information across multiple output elements in anticipation of losses, facilitating the decoder's frame reconstruction when packets are actually lost. Consequently, GRACE displays a more graceful quality degradation amid varying losses, while consistently delivering higher video quality than previous solutions (Figure 1).

To materialize the above benefits of GRACE's codec, our design of GRACE addresses three system challenges.

First, to ensure loss tolerance, each packet must be independently decodable. Existing solutions achieve this by dividing the frame into independently decodable units. However, this introduces a size overhead because the data in each unit follows different distributions and thus cannot be compressed efficiently. In response to this challenge, we train GRACE's neural encoder to regularize the values in its output to conform to the same distribution, thereby reducing the partitioning overhead. We also utilize reversible random mapping [8] during such partitioning, making it more amenable to NVCs. While training GRACE under packet losses, simulating random partitioning and packet losses is inefficient and precludes differentiability. Hence, we apply random zeroing to the encoder's output directly, simulating packet losses without actually dropping packets (§3).

Second, packet losses can lead to discrepancies between the reference frames at the encoder and decoder side, resulting in sustained quality degradation in the decoded video stream if synchronization is not maintained. Traditional remedies, such as retransmission or sending a new keyframe, fall short of seamlessly rectifying this inconsistency. GRACE introduces an innovative protocol to adeptly realign the encoder and decoder states without hindering video decoding. In the event of packet loss, the decoder leverages the loss resilience of GRACE to decode partially received packets. Simultaneously, the decoder communicates the loss details to the encoder. This feedback mechanism enables the encoder to swiftly adjust its recent reference frames to match those at the decoder end, eliminating the need for additional data transmission (§4.2).

Third, GRACE must be efficient to encode and decode video in real-time across various devices, from laptops to mobile phones. Existing NVCs, however, often employ expensive neural networks, particularly for motion estimation and post-processing. We show that by downscaling the image input for motion estimation and simplifying post-processing, GRACE accelerates the encoding and decoding by 4× without a noticeable impact on loss resilience. Moreover, with hardware-specific runtimes such as OpenVINO and CoreML, GRACE attains over 25 fps on CPUs and iPhones (§4.3).

Comprehensive experiments (§5) on a diverse set of videos and real network traces show that with the same congestion control logic, GRACE reduces undecodable frames by 95% and stall duration by 90% compared with state-of-the-art FEC baselines. It also boosts the visual quality metric of SSIM by 3 dB over a recent neural error concealment scheme (§5.1). Our IRB-approved user study with 240 crowdsourced participants and a total of 96 subjective ratings demonstrates a 38% higher mean opinion score (MOS) for GRACE, further attesting to its effectiveness. Regarding computational efficiency, GRACE achieves more than 25 fps on popular mobile devices (e.g., iPhone 14 Pro), meeting the real-time requirement.

Our contributions are summarized as follows. *(i)* We present GRACE, which, to the best of our knowledge, represents **the first effort to *jointly* train a neural video encoder and decoder under a spectrum of packet losses, aiming to improve loss resilience in real-time video** (§3). Different from other recent ML-based real-time video systems [107, 108, 114] that use ML-based rate adaptation to minimize packet loss, GRACE uses ML to make the video codec itself resilient to packet loss. *(ii)* We build the end-to-end video system to address the practical challenges associated with integrating a new NVC, developing optimization techniques related to packetization, encoder/decoder state synchronization, and runtime efficiency (§4).

2 Background

2.1 Real-time video coding

To help explain GRACE's design, we first introduce some key concepts in real-time video coding and streaming.

The sender encodes video at a specific frame rate and bitrate, e.g., with 25 fps (frames per second) and 3 Mbps, the encoder generates a 15 KB frame on average every 40 ms. An encoded video is composed of groups of consecutive frames, called a group of pictures (GoP). Each GoP starts with an I-frame (or key-frame), followed by multiple P-frames (or inter-frames)[2]. I-frames are independently encoded without referencing other frames, while P-frames encode only the differences relative to previous reference frames. In real-time video, the majority of frames are P-frames to minimize frame sizes, so our discussion here focuses on P-frames. Figure 2 shows the workflow of P-frame encoding and decoding. Given a new frame and a reference frame, the encoder (1) calculates motion vectors (MVs) and residuals for each macroblock (MB), e.g., 16×16-pixel samples, (2) transforms and quantizes the MVs and residuals, (3) performs entropy encoding on the transformed data, (4) divides the entropy-encoded data into packets, and (5) transmits these packets with congestion control, such as GCC [31]. Correspondingly, the receiver depacketizes and decodes the received data to reconstruct each frame from the received packets.

To reduce frame delay, which denotes the time from frame encoding to rendering, real-time video clients (e.g., WebRTC) commonly make two choices that differentiate them from video streaming (e.g., Netflix, ESPN Live):

[2] On-demand video also uses B-frames (bidirectional predicted frames), which refer to both past and future P-frames. However, real-time video rarely uses B-frames in order to render frames as soon as possible.

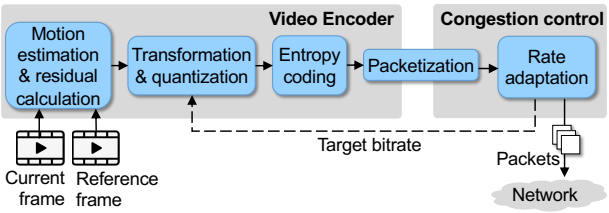


Figure 2: *A typical workflow of video frame encoding.*

- Real-time video employs notably *shorter* (tens of ms) buffers, as opposed to video streaming that utilizes several seconds of playback buffer for on-demand or live content. Thus, retransmission delay is difficult to conceal with such short buffers, especially in high-latency networks.
- To maintain a short buffer, real-time video sends each frame in a burst and decodes it as soon as its packets are received. As a result, *any* lost packets, whether due to drops or queuing, can affect frame decoding. In contrast, streaming video is transmitted in chunks (each with hundreds of frames) over the HTTP/TCP protocol.

Ideally, congestion control and bitrate adaptation (e.g., GCC [31], Salsify [45], and NADA [116]) are designed to handle bandwidth fluctuations, thereby avoiding congestion-induced packet losses. However, predicting bandwidth fluctuations in advance is challenging, making loss-resilient methods necessary when decoding frames under packet loss.

We define *packet loss per frame* as any packets not received before the receiver is expected to decode the frame. In other words, even if a packet is not dropped, it can still be counted as packet loss if it arrives too late. It is important to note that our notion of packet loss differs from network-level loss. Even with low network loss (which typically remains below 1%), real-time video may still encounter a high packet loss rate (e.g., over 50%) in certain frames, as corroborated by recent research in this space [35, 36, 86].

2.2 Related work

Various loss-resilient schemes have been studied.

Forward error coding (FEC) adds redundancy at the *encoder* before the data is sent to the network. This is also known as error-resilient channel coding. Examples include Reed-Solomon codes, LDPC [77], fountain and rateless codes [33, 76], streaming codes [28, 86], and recent ones based on DNNs [32, 50]. There are also hierarchical and multilevel FEC [94, 95], which organizes FEC into multiple layers and protects each layer with different redundancies. FEC is also used to protect frame metadata or the base layer in SVC (also known as UEP [25, 113]). However, in order to pick a suitable rate of redundancy, they need to estimate how many packets will be lost in advance. If the loss rate is underestimated, the redundancy will be insufficient to recover missing packets. On the other hand, adding excessive redundancy results in a higher bandwidth overhead and, in turn, a lower video quality.

Postprocessing error concealment reconstructs missing data in lost packets at the *decoder*. These methods generally consist of two components. First, the encoded packets should be decodable when only a subset of the packets is received. This is accomplished through INTRA-mode macroblock encoding [38], slice interleaving [56], or flexible macroblock ordering [66]. However, these approaches often compromise the encoder's ability to exploit redundancies across neighboring MBs, as adjacent MBs are either encoded in INTRA mode or split into different packets (in a checkerboard manner [64, 66] or based on ROI detection [93]). Therefore, these methods impair compression efficiency, causing the encoded frame size to inflate by 10%–50% [42, 64, 74, 99].

Then, the decoder reconstructs lost data based on the received packets, using classic heuristics (e.g., motion vectors interpolation [63, 97, 115] and intra-block refreshing [64] in H.264) or neural-network-based inpainting [59, 67, 79, 87, 102]. Recent work [67] use vision transformers [27, 43] to directly predict the missing bits in the lost packets before frame decoding. However, due to the encoder's lack of awareness of the decoder's postprocessing, each encoded packet contains limited redundancy and information that could aid in reconstructing missing motion vectors or residuals. As a result, the reconstruction process is forced to guess the missing data when a packet is lost. Even recent techniques still see a notable drop in video quality (e.g., PSNR drops from 38 dB to 25 dB at a 20% packet loss [81]).

GRACE takes a different approach than FEC and error concealment. Unlike error concealment that relies only on decoder-side postprocessing, GRACE jointly optimizes (via training) both the (neural) encoder and decoder. Unlike FEC that requires a pre-determined redundancy rate, GRACE's codec is optimized across a range of packet loss rates.

Other schemes: While there exist other techniques that might help mitigate the impact of packet loss, their primary goals are *not* loss resilience. Nevertheless, for the sake of completeness, we discuss some notable schemes here and also quantitatively compare GRACE against several of them in §5.

Scalable video coding (SVC) [40, 88, 89] and fine-granularity scalability (FGS) [68, 75] aim to optimize rate-distortion (RD) tradeoff—video quality achieved by a single encoded bitstream under different received bitrates. SVC encodes a video in multiple quality layers and sends data *layer by layer*. This is feasible for on-demand video [40, 72] but in real-time video, all packets of a frame are sent together to reduce frame delay (§2.1). When a packet loss occurs to a base layer, it will block the decoding of any higher layers. For this reason, SVC is rarely used to improve unicast real-time video (though it is used in multicast video to serve users with heterogeneous network capacities [88]).

There are a few alternatives to postprocessing error concealment. For instance, when loss occurs, Salsify [45] reverts to an older but reliably received frame—instead of the last frame—as the reference frame, so the decoder can safely skip a loss-affected frame without hurting subsequent frames. However, it needs more bits to encode the same quality than using the

last frame as the reference frame, e.g., the P-frames between every other frame are 40% greater in size than between two consecutive frames. Similar limitations apply to long-term reference frame (LTR) [112], which makes each P-frame individually decodable if the long-term reference is received, regardless of packet loss in between or not. Voxel [83] skips a loss-affected frame if the encoder indicates that skipping the frame does not affect video quality. It works well for on-demand video where B-frames can be safely skipped, and the impact of a skipped frame will stop at the next chunk within a few seconds. Unfortunately, neither applies to real-time video.

Recently, deep learning has been used in super resolution [61, 92, 106], SVC [40, 75], and postprocessing error concealment based on CNNs [59, 79, 87, 102] or transformers [44, 67]. Super-resolution can reduce packet losses by sending the video in a lower bitrate and enhancing the video quality on the receiver side. However, it still requires retransmissions to rectify frames impaired by packet loss. For SVC and postprocessing error concealment techniques, the aforementioned limitations inherent to these approaches remain, despite the use of deep learning.

Loss resilience has also been studied under specific assumptions, such as availability of multi-path [41, 80], early retransmission driven by router feedback [117], low-latency networks [85], and availability of video gaming states [52, 53, 101]. We do not make special assumptions in this work.

2.3 Neural video codec background

Our work is based on neural video codecs (NVCs), which use learned neural networks (NNs), instead of handcrafted logic, to encode and decode video frames [40, 55, 73]. Recent NVCs have demonstrated comparable or even better compression efficiency than traditional video codecs for two reasons:

- They leverage logical components commonly found in traditional video codecs, such as motion estimation, warping, and transformative compression (§2.1), replacing their handcrafted heuristics with NNs, which can learn more sophisticated algorithms from data.
- These NVCs exhibit remarkable generalization across a variety of video content because of training on a large corpus of videos (e.g., Vimeo-90K [103]). This capability to generalize is also observed in our evaluation (§5).

Despite their exceptional compression efficiency, NVCs have received little attention so far in the context of loss resilience. However, we believe NVCs have the potential to achieve greater loss resilience for the following reasons.

- First, unlike traditional codecs that map each pixel (or macroblock) to a distinct motion vector/residual, the highly parameterized NN of NVC's encoder can be trained to map the information of each pixel to multiple elements in output tensor, potentially making lost information recoverable.
- Second, the NVC's decoder, comprising convolutional NNs, can be trained to decode not only a direct encoder output but also tensors that resemble those with perturbations such as random noise or zeroing. In contrast, traditional codecs might fail to decode under similar circumstances.

Nevertheless, NVCs as is still lack tolerance to packet loss. Their standard training implicitly assumes that the encoder's output is identical to the decoder's input, so it does not prepare the NVC to handle data loss between the encoder and decoder. Meanwhile, entropy encoding used in conventional NVCs compresses the entire encoder output as a single bitstream, and thus any packet loss will render it undecodeable.

GRACE is an attempt at transforming NVCs to be resilient to different packet loss rates. Our work is related to an emerging line of work on deep joint source-channel coding [30, 37, 65], which trains an NVC to encode images in a representation robust to signal noises. GRACE differs with them on two key fronts. First, GRACE handles video frames, which cannot be treated separately as individual images because any error in one frame can propagate to future frames. Second, GRACE handles packet losses rather than physical-layer signal noises, which can be naturally modeled by differentiable linear transformations [30, 51].

In short, traditional error-resilient methods struggle to maintain video quality across a range of packet losses. Encoder-based forward error coding (FEC) optimizes quality only for a pre-determined maximum loss rate, whereas decoder-based postprocessing error concealment suffers from suboptimal quality especially at high loss rates. On the other hand, existing NVCs have the potential to tolerate data perturbations but are not explicitly designed to handle packet losses.

3 Training GRACE's neural video codec

This section outlines the training process of GRACE's neural video codec (NVC). At a high level, GRACE *jointly trains the neural encoder and decoder under a range of packet losses* to achieve enhanced loss resilience.

Basic NVC framework: Figure 3 depicts the workflow of GRACE's encoder and decoder (excluding entropy coding and packetization). The encoder follows a similar logical process as a traditional video encoder (Figure 2). It first employs a neural network (NN) to estimate motion vectors (MVs) and encodes them into a quantized tensor using an NN-based MV encoder. Subsequently, the tensor is decoded back into MVs to match those received by the decoder. Next, the encoder applies these MVs to the reference frame to generate a motion-compensated frame, and uses a frame smoothing NN to increase its similarity with the current frame before calculating the residual differences between them. Finally, an NN-based residual encoder encodes the residuals into another quantized tensor. When the encoded MV tensor and the encoded residual tensors are received by the decoder, they go through the NN-based MV decoder and residual decoder jointly trained with their respective encoders. Appendix A.1 provides more details of the tensors and NNs.

Although both the encoder and decoder of GRACE contain

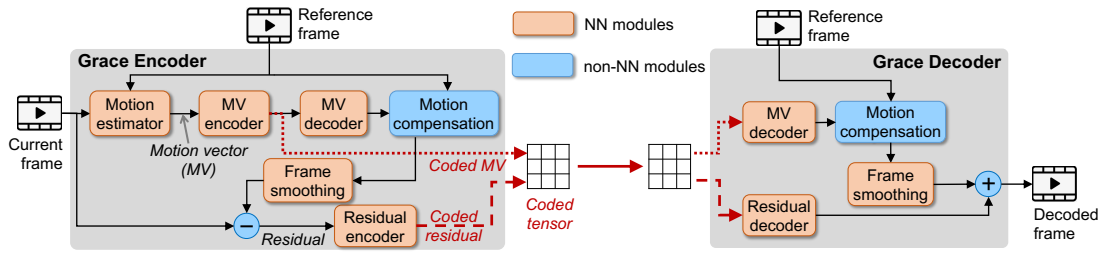


Figure 3: *Workflow of* GRACE*'s neural video codec.*

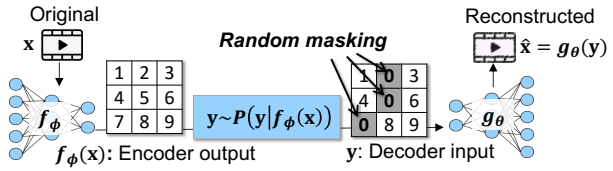


Figure 4: *Unlike traditional NVC training that assumes no data loss between the encoder and decoder,* GRACE *applies "random masking"—setting a fraction of randomly selected elements to zeros—to the encoder's output.*

multiple steps, they can be viewed as two differentiable models. We denote the encoder by f_ϕ (with its NN weights ϕ) and the decoder by g_θ (with its NN weights θ). The encoder encodes a frame $\mathbf{x}$ into a *coded tensor* $\mathbf{y} = f_\phi(\mathbf{x})$, and the decoder decodes $\mathbf{y}$ into a reconstructed frame $\hat{\mathbf{x}} = g_\theta(\mathbf{y})$. Traditionally, NVC seeks to minimize the following loss function:

$$\mathbb{E}_{\mathbf{x}}[\underbrace{D(g_\theta(\mathbf{y}),\mathbf{x})}_{\text{Pixel error}} + \alpha\cdot \underbrace{S(f_\phi(\mathbf{x}))}_{\text{Encoded size}}], \text{ where } \underbrace{\mathbf{y} = f_\phi(\mathbf{x})}_{\text{No data loss}} \quad (1)$$

Here, $D(\hat{\mathbf{x}},\mathbf{x})$ is the pixel-level reconstruction error of the decoded frame $\hat{\mathbf{x}}$ (by default, L2-norm[3] of $\hat{\mathbf{x}}-\mathbf{x}$), and $S(\mathbf{y})$ is the entropy-coded data size of $\mathbf{y}$ in bit-per-pixel (BPP). The parameter α governs the size-quality tradeoff: a higher α leads to a smaller frame size, $S(x)$, but higher distortion (i.e., poorer quality) of the reconstructed frame $\hat{\mathbf{x}}$. As all the functions—f_ϕ, g_θ, D, and S (approximated by a pre-trained NN [73])—are differentiable, the NN weights ϕ and θ can be trained jointly via gradient descent to minimize Eq. 1.

Simulating packet loss during training: We begin by pre-training an NVC using Eq. 1, which we refer to as GRACE-P, and then fine-tune it by introducing simulated packet losses in the following manner. GRACE simulates the impact of packet losses by randomly "masking"—zeroing selected elements—in the encoder's output, $f_\phi(\mathbf{x})$, as shown in Figure 4. The fraction of zeroed elements is dictated by a distribution, $P(\mathbf{y}|f_\phi(\mathbf{x}))$, which represents the probability distribution of the resulting tensor $\mathbf{y}$ after random masking $f_\phi(\mathbf{x})$. For instance, with a 33% loss rate, $P(\mathbf{y}|f_\phi(\mathbf{x}))$ is the probability of $\mathbf{y}$ arising from the random masking of 33% of elements in $f_\phi(\mathbf{x})$, as illustrated in Figure 4. Formally, GRACE jointly

[3]Note that the L2-norm (or mean squared error) of $\hat{\mathbf{x}}$ and $\mathbf{x}$ is closely related to the PSNR of $\hat{\mathbf{x}}$. To avoid this bias, our evaluation in §5 measures the quality improvement in SSIM and subjective user studies.

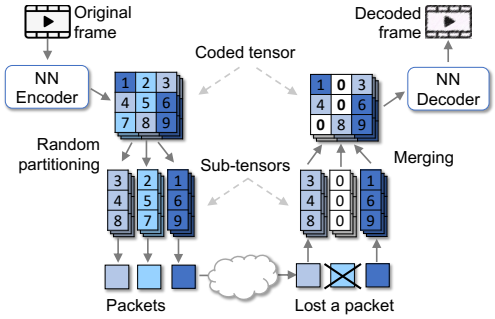


Figure 5: GRACE*'s reversible randomized packetization. The tensor elements mapped to a lost packet will be set to zeros.*

trains the encoder and decoder NNs to minimize:

$$\mathbb{E}_{\mathbf{x}}[D(g_\theta(\mathbf{y}),\mathbf{x}) + \alpha\cdot S(f_\phi(\mathbf{x}))], \text{ where } \underbrace{\mathbf{y} \sim P(\mathbf{y}|f_\phi(\mathbf{x}))}_{\text{Simulate packet loss}} \quad (2)$$

The key difference from the traditional objective in Eq. 1 is the distribution function P (highlighted in blue), which captures the distribution of decoder input under packet loss.

To train the weights of ϕ and θ under the random perturbations of P, we employ the REINFORCE trick [62] (commonly used in reinforcement learning [84, 109]) to approximate the gradient through Monte Carlo sampling. A more detailed mathematical formulation is included in Appendix A.2.

Choosing simulated packet loss rates: To prepare GRACE's NVC to handle a wide range of loss rates, it is essential to simulate such losses in training. One approach is to select loss rates uniformly at random from [0, 100%) and apply them to the encoder output $f_\phi(\mathbf{x})$. However, the resulting NVC turns out to perform poorly, especially when dealing with low loss rates. Notably, even when high loss rates (e.g., over 80%) are introduced only in a small fraction of training samples, we empirically observe a significant drop in video quality under low loss rates while the quality improvement under high loss rates is only marginal. This phenomenon could be attributed to the encoder's tendency to incorporate more redundant information to prepare for high loss rates, adversely affecting video quality at low loss rates. Therefore, a practically effective distribution should cover both low and high loss rates, with a slight emphasis on low losses. Our final choice of loss rate distribution is described in §4.4.

Packetization during inferencing: Recall that during training, we simulate packet loss by applying random masking

rather than replicating the actual packetization and packet dropping process. Therefore, it is important to ensure that the impact of actual packet loss during runtime mirrors the effects of random masking. To achieve this, GRACE employs reversible randomized packetization as shown in Figure 5. GRACE's sender first splits the encoded tensor of a frame (both the encoded MVs and encoded residual) into multiple subtensors using a uniform random mapping. We use a reversible pseudo-random function to generate the mapping so that the receiver can correctly recover the original tensor with the same random seed. Specifically, we map the ith element to the $j = (i \cdot p \mod n)^{\text{th}}$ packet at the $[(i \cdot p - j)/n]^{\text{th}}$ position, where n is the number of packets and p is a prime number. If a packet is lost, the decoder assigns zero to each element whose position is mapped to the lost packet. Consequently, an $x\%$ packet loss rate has the effect of randomly zeroing $x\%$ of the values in the encoder's output tensor. [4] §4.1 explains how each subtensor is losslessly compressed via entropy encoding into the bitstream of a packet, but this lossless compression is bypassed during training for efficiency purposes.

Why is GRACE more loss-resilient? Unlike decoder-side error concealment, the joint training ensures that the *encoder* is also aware of packet losses. Empirically, we observe that GRACE's encoder tends to produce more non-zero values in its output than an NVC pre-trained on the same dataset but without simulated packet loss. This increase in non-zero values can be viewed as more "redundancy," as the encoder embeds each pixel's information into multiple elements in its output tensor, assisting the decoder in discerning loss-affected elements (from intended zeros) and reconstructing video better under packet losses. §5.4 empirically shows that training only the decoder with simulated loss cannot reach the same level of loss resilience (Figures 20 and 29).

4 Real-time video framework

With the training techniques detailed in §3, GRACE's NVC acquires the ability to withstand simulated packet losses. This section describes the integration of this NVC into a real-time video delivery framework: GRACE entropy-encodes the neural encoder's output into packets (§4.1), streams frames under packet loss (§4.2), and accelerates encoding and decoding across various devices (§4.3).

4.1 Entropy encoding the encoder's output

As mentioned in §3, GRACE splits the encoder's output into subtensors using a reversible-random function, with each subtensor corresponding to an individual packet. Similar to classic codecs such as H.265 and VP9, each subtensor undergoes lossless compression into a bitstream through arithmetic (entropy) coding. An arithmetic encoder uses an underlying symbol distribution to compress the values in the tensor. Instead of relying on hand-tuned heuristics (e.g., CABAC [4] in H.265), we adopt the method described in [73], training a distribution estimator in conjunction with the neural encoder and decoder to better estimate the symbol distribution of each encoder output. Since GRACE decodes individual packets independently, the symbol distribution of a packet must be sent as part of the packet to the decoder, which implies that the size overhead of symbol distributions increases with more packets.

GRACE reduces this overhead by employing a simpler symbol distribution that requires fewer bits to store within each packet. Specifically, GRACE trains the neural encoder to regularize the distribution of values in each encoder's output channel (224 channels in total) to conform to a zero-mean Laplace distribution. In doing so, the symbol distribution only needs to store the variance for each channel while still effectively compressing the encoder's output tensor. As a result, the symbol distribution now requires only ~50 bytes per packet to store, a reduction from 40% of the packet size to 5%, without notably affecting the compression efficiency.

4.2 Streaming protocol

Basic protocol of GRACE: The encoder of GRACE encodes new frames at a fixed frame rate. When any packet for the next frame arrives, the decoder immediately attempts to decode the current frame. Unless all packets of the current frame are lost (which triggers a request for resending the frame), the decoder will decode the current frame using whatever packets have been received. We refer to a frame decoded using partially received packets as an *incomplete frame*. However, while GRACE can decode incomplete frames with decent quality, using these incomplete frames as reference images for decoding future frames causes the encoder's and decoder's states to be "out of sync," i.e., the next frame will be decoded based on a different reference image than the one used during encoding. This inconsistency causes error to propagate [96] to future frames even if all their packets arrive without loss.

One strawman solution to resolve error propagation is to synchronize the encoder and decoder on each frame. However, the encoding of each frame would be blocked until it knows which packets are used to decode the previous frame. This synchronization delay would render pipeline encoding, transmission, and real-time decoding infeasible.

Optimistic encoding with dynamic state resync: GRACE employs two strategies to prevent out-of-sync states from blocking the encoder or the decoder.

First, the encoder *optimistically* assumes all packets will be received and encodes frames accordingly, taking advantage of GRACE decoder's tolerance to packet losses for a small number of frames. For instance, §5.2 shows that GRACE is resilient against packet loss across 10 consecutive frames.

Second, when receiving an incomplete frame, the decoder, without stopping decoding new frames, requests the encoder to dynamically resynchronize the state in the following man-

[4] That said, such reversible random packetization requires a frame containing multiple packets. Therefore, GRACE's encoder controls the packet size such that each frame has at least 2 packets, since real-time video packets don't need to be as large as 1.5 KB [90] in practice.

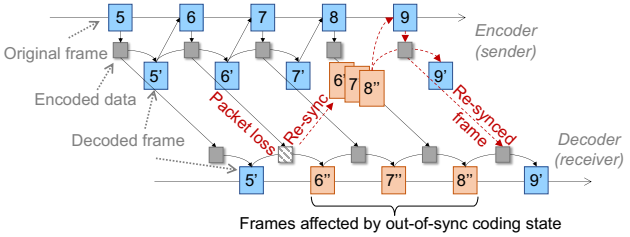


Figure 6: *Packet loss introduces discrepancies between the encoder's and the decoder's reference frames.* GRACE*'s state resync efficiently rectifies these discrepancies without causing interruptions for either the encoder or the decoder.*

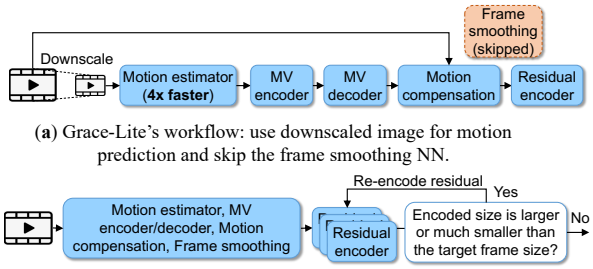
ner. Upon receiving a resync request, the encoder re-decodes the recent frames starting from the incomplete frame, using only the subset of packets received by the decoder (as indicated in the resync request), to compute the latest reference frame used by the decoder. As illustrated in Figure 6, if the encoder is about to encode the 9th frame and learns that the 6th frame has been decoded using partially received packets, it then quickly re-decodes frames from the 6th to the 8th. The 8th frame now aligns with the receiver's observation and thus is used as the reference frame for encoding the 9th frame.

A potential speed bottleneck is the re-decoding of frames during state resynchronization (e.g., the 6th to 8th frames in Figure 6). Fortunately, the encoder can re-decode these frames much faster than the regular decoding process by running only the motion decoder and the residual decoder. The insight is two-fold. First, motion estimation, motion encoding, and residual encoding can be skipped because these frames have already been decoded once at the encoder side, so the re-decoding only needs to estimate the incremental changes caused by the lost packets. Second, while skipping the frame smoothing NN may impact the compression efficiency of the last frame (e.g., the 9th frame in Figure 6), it only affects a single frame since the next frame will still be optimistically encoded. Appendix B.1 provides more details on the dynamics re-decoding, and §5.4 analyzes its runtime overhead.

GRACE's approach of optimistic encoding and dynamic state resynchronization capitalizes on a key advantage of GRACE's NVC—it does not need to skip or block the decoding processing for loss-affected frames; instead, it can decode them with decent quality *while* the encoder's and decoder's states are out-of-sync for a few frames, thus reducing frame delay. This approach differs from NACK (negative acknowledgement) in WebRTC [54], which requires blocking the decoding of loss-affected frames, and from Salsify's state synchronization [45], which skips all loss-affected frames.

4.3 Fast coding and bitrate control

Fast encoding and decoding: Using a standard GPU runtime with PyTorch JIT compiling [14], GRACE meets the latency requirement for real-time video communication on GPUs. As shown in §5.4, GRACE encodes and decodes 720p video at 31.2 and 51.2 fps respectively, on an NVIDIA A40 GPU



(**a**) Grace-Lite's workflow: use downscaled image for motion prediction and skip the frame smoothing NN.

(**b**) Accurate bitrate control: When the frame size is too large or small, only the residual requires re-encoding.

Figure 7: GRACE *adapts the NVC for efficient execution on CPUs and accurate bitrate control.*

(5× cheaper and 3× slower than A100). However, GRACE's NVC remains too heavy to run on laptops with CPUs and mobile phones. To address this, we develop GRACE-Lite, a lightweight version of GRACE that incorporates three optimizations (Figure 7a): *(i)* motion estimation NN operates on 2× downsampled frames, speeding up the motion estimation by 4×; *(ii)* frame smoothing NN is skipped; *(iii)* the floating point precision in NNs is reduced from 32 bits to 16 bits, making the inference 2× faster. These optimizations allow GRACE to encode and decode frames on an iPhone 14 Pro at 26.3 and 69.4 fps when compiled with the CoreML [5] library, while maintaining similar loss resiliency as GRACE (§5.4).

Accurate bitrate control: Video encoders are expected to encode frames to match the target frame sizes. Similar to Salsify [45], GRACE encodes a frame multiple times at different quality levels but in a faster way than encoding the frame from scratch each time (illustrated in Figure 7b). To achieve this, GRACE trains multiple neural encoders, each with a different α (in Eq. 2) to enable different quality-size tradeoffs. During the training phase, adjustments are confined to the residual encoders and decoders, leaving other NN weights fixed. Thus, once a frame is encoded, both the motion vector and residual are reusable, with the residual undergoing further encoding through different encoders, each producing a different frame size. This procedure, taking under 3 ms, can encode a frame multiple times using residual encoders with distinct α values. In practice, residual sizes can vary from 0.1× to 10× the MV size, allowing GRACE to cover a wide range of bitrates (§5.2).

4.4 Implementation of GRACE

GRACE is implemented in ∼2000 lines of Python code, including its NVC, packetization, bitrate adaptation, and state synchronization protocol.

Training: GRACE's NVC model architecture is based on a recent work called DVC [73]. We fine-tune GRACE from the pre-trained DVC model on the Vimeo-90K [103] dataset, under the following distribution of simulated per frame packet loss (§2.1): with an 80% probability, the loss rate is set to 0%; with a 20% probability, the loss rate is randomly selected

from {10%, 20%, 30%, 40%, 50%, 60%} [5]. By using this loss distribution, GRACE can be resilient to a wide range of loss rates without assuming the underlying network loss pattern. To achieve accurate bitrate control (§4.3), we first fine-tune an NVC with a default α (2^{-7}) using Eq. 2. Subsequently, we perform fine-tuning with 11 α values spanning from 2^{-8} to 2^{-15}, specifically to refine the residual encoder and decoder for bitrate adaptation. With a learning rate of 10^{-4}, each fine-tuning step takes about 1–2 hours on an Nvidia A40 GPU.

Delivery: We use *torchac* [13] for entropy encoding and decoding. GRACE utilizes PyTorch JIT compilation [14] when running on GPUs, while GRACE-Lite leverages CoreML [5] for inferencing on mobile devices. Both GRACE and GRACE-Lite operate using 16-bit floats at runtime. GRACE uses BPG [21] to encode and decode I-frames every 1000 frames, and can be integrated with any congestion control (CC) algorithms. Due to space limitations, we provide more details about I-frames and CCs in Appendix B.2 and B.3.

5 Evaluation

Our key findings are as follows:

- **Loss resilience:** GRACE's quality under no packet loss is on par with H.264/H.265 and gracefully declines with higher loss rates. Under 20–80% packet loss, GRACE improves the SSIM by 0.5–4 dB compared with other loss-resilient baselines across diverse videos.
- **Better video smoothness:** Under bandwidth fluctuations in real network traces, GRACE reduces the number of video freezes over 200 ms by up to 90%, tail frame delay by up to 2–5×, and non-rendered frames by up to 95%. Our user study also confirms a 38% rated score for GRACE.
- **Speed:** Our implementation of GRACE encodes/decodes 480p video at 65.8 fps/104.1 fps and 720p video at 33.6 fps/44.1 fps using Nvidia A40 GPU, 1.5–5× faster than recent neural video codecs [40, 73, 91, 105]. With the optimization detailed in §4.3, GRACE can encode/decode 720p video at 26.2 fps/69.4 fps on an iPhone 14 Pro with marginal quality degradation.

5.1 Setup

Testbed implementation: Our testbed 2 Nvidia A40 GPUs to run the video encoding and decoding with GRACE's NVC (each using one GPU). We use a packet-level network simulator to compare GRACE with baselines under various network conditions. The simulator uses a configurable drop tail queue to mimic congestion-induced packet losses and uses a token bucket scheme to simulate bandwidth variation every 0.1 seconds. Google Congestion Control (GCC) [31], a standard WebRTC algorithm widely used in real-time video applications, is used to determine the target bitrate of video codecs at each frame. It is worth noting that GCC is responsive to bandwidth drops and packet losses, as it tends to send data conservatively to avoid video delays and stalls caused by packet losses. The simulator includes encoding, packetization, rate adaptation, and decoding. We set the default frame rate at 25 fps (on par with typical RTC frame rates [78]), though GRACE can encode at a higher frame rate (§5.4). Instead of replaying stationary traffic/loss traces, the testbed can simulate dynamic packet loss rates under real-world bandwidth fluctuations. It records each decoded frame and its delay, including encoding, transmission, and decoding. We have confirmed our simulator's accuracy regarding frame delay via a real-world validation experiment in the appendix (§C.3).

[5] The packet loss rate should follow a uniform distribution covering a continuous range of losses (e.g., [0, 60%]). However, we empirically observe that using a discrete loss rate distribution makes the model converge faster without sacrificing the loss resilience.

Dataset	# of videos	Length (s)	Size	Description
Kinetics	45	450	720p 360p	Human actions and interaction with objects
Gaming	5	100	720p	PC game recordings
UVG	4	80	1080p	HD videos (human, nature, sports, etc.)
FVC	7	140	1080p	In/outdoor video calls
Total	**61**	**770**		

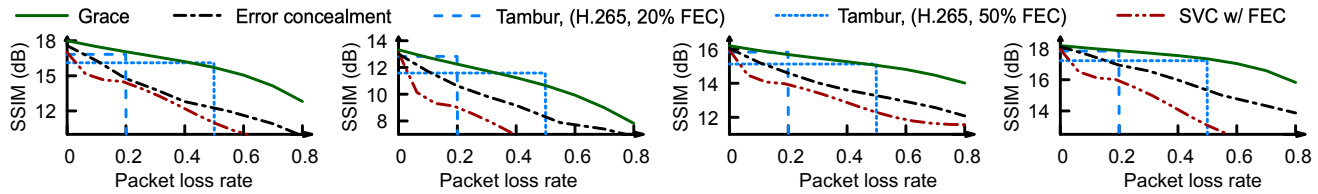
Table 1: *Dataset description.*

Test videos: Our evaluation uses 61 videos randomly sampled from four public datasets, summarized in Table 1. The total content length is 770 seconds where each video is 10–30 seconds long, matching the setup of similar works [39, 45, 86]. Importantly, these videos are obtained from entirely different sources than the training set, and they span a range of spatial complexity and temporal complexity (detailed in Appendix C.4), as well as multiple resolutions. This diversity allows us to assess GRACE's average performance across different contents and study how content affects its performance.

Network traces: We test GRACE and the baselines on 16 real bandwidth traces, eight of which are LTE traces from the Mahimahi network-emulation tool [9, 82], and the rest are broadband traces from FCC (July 2021) [10]. The traces are in the format of bandwidth timeseries. The bandwidth fluctuates between 0.2 Mbps to 8 Mbps in the traces. By default, we set the one-way propagation delay to 100 ms and the queue size to 25 packets. We also vary these values in §5.3.

Baselines: We employ H.265 (through FFmpeg v4.2.7) as the underlying video codec for all baselines (except for NVC-based ones) since H.265 is recognized with comparable or better compression efficiency than VP8/9 and H.264 [15, 49] (as confirmed in Appendix C.1). We compare GRACE against a range of loss-resilient baselines that cover various approaches outlined in §2.2 (more details in Appendix C.2).

- **Forward error correction**: We use Tambur [86], a state-of-the-art FEC scheme based on streaming codes [28]. Its redundancy rate dynamically adapts based on the mea-



(a) Kinetics dataset (b) Gaming dataset (c) UVG dataset (d) FVC dataset

Figure 8: *Video quality achieved by different schemes under varying packet loss rates at the same encoded bitrate (6 Mbps).*

sured packet loss in the preceding 2 seconds. Compared with regular FEC, streaming codes reduce the number of non-decodable frames when transmitting an equivalent amount of parity packets. We have also validated that Tambur outperforms WebRTC's default FEC scheme.

- **Decoder-side neural error concealment**: We use ECFVI [59], an NN-based error concealment method shown to outperform previous techniques relying on motion estimation recovery [87] or inpainting [34].[6] To ensure each packet is independently decodable, we apply flexible macroblock ordering (FMO) [42] to split the frame into 64×64-pixel[7] blocks and map them randomly to packets. This results in a 10% increase in the encoded frame size, in line with previous findings [64, 74, 99]. After decoding an incomplete frame, ECFVI uses NNs to estimate missing motion vectors and enhance the reconstructed frame through inpainting.
- **Scalable video coding (SVC)**: We implement an idealized SVC, designed so that when the first k layers arrive, it achieves the same quality as that of H.265 with the same number of received bytes. This idealized implementation surpasses the state-of-the-art NN-based SVC [40]. We also add 50% FEC to protect the base layer for SVC, following a common practice in real-time video applications [60].
- **Selective frame skipping**: Salsify [45] skips frames affected by loss at the decoder side after the encoder receives the packet loss indication and resends a new P-frame using the last fully received frame as a reference. Voxel [83] employs selective frame skipping to mitigate video rebuffering and improve the user's QoE.

We make another idealized assumption in favor of SVC, Salsify, and Voxel. We assume that their codec's output bitrate on every frame perfectly matches the target bitrate determined by the congestion control algorithm, i.e., no overshoots or undershoots. This idealization makes these baselines perform slightly better than they would under real-world conditions.

Variants of GRACE: To highlight the impact of different design choices, we evaluate **GRACE-P** and **GRACE-D**. They are trained the same way as GRACE, except that GRACE-P does not use simulated loss while GRACE-D freezes the encoder NN weights (i.e., fine-tuning only the decoder NN with simulated loss). They represent alternative ways to simulate packet losses during training. We also test **GRACE-Lite**, which incorporates the optimizations described in §4.3.

Furthermore, we present the quality improvement achieved by the state-of-the-art super-resolution (SR) model [70] when applied to GRACE and other baselines. It is important to note that SR can be applied to any decoded frames, making it *orthogonal* to GRACE's design space. Details of this experiment are provided in Appendix C.8.

Metrics: Following prior work on real-time video communication [26, 45, 47, 48, 86], we measure the performance of a video session across three aspects.

- *Visual quality* of a frame is measured by SSIM. Following recent work [45, 104], we express SSIM in dB, calculated as $-10\log(1-\text{SSIM})$ across all rendered frames.
- *Realtimeness* is measured by the 98th percentile (P98) of frame delay (time elapsed between the frame's encoding and decoding), and non-rendered frames (either undecodable due to insufficient FEC protection or exceeding 400 ms after the frame is encoded).
- *Smoothness* of the video is measured by video stall, defined as an inter-frame gap exceeding 200 ms, following the industry convention [78]. We report the average number of video stalls per second and the ratio of video stall time over the entire video length.

5.2 Compression efficiency and loss resilience

Loss resilience: In real world, packet loss per frame (defined in §2.1) can span a wide range from 0 to over 80% [86]. Figure 8 compares GRACE's video quality with the baselines under varying packet loss rates across different test video sets. For a fair comparison, we fix the encoded bitrate of all baselines at 6 Mbps (with actual differences under 5%) while ensuring that GRACE's encoded bitrate *never* exceeds that of the baselines. On average, the quality of GRACE drops by 0.5 dB to 2 dB in SSIM as the packet loss rate rises from 20% to 50%, and by up to 3.5 dB when the packet loss rate reaches 80%. These quality drops of GRACE are notably lower than the baselines, including FEC-based and neural error concealment schemes, at the same packet loss rates.

Figure 9 shows the average quality across all test videos

[6] A parallel effort, Reparo [67], demonstrates effective error concealment for a particular video type (talking head), but it lacks comparisons with any NN-based baselines and does not provide a public codebase for testing.

[7] A smaller block size such as 16×16 can greatly inflate the frame size [64, 74, 99], while a larger block size such as 256×256 hinders information recovery upon packet loss. We empirically choose the 64×64 block size to balance between frame size and quality.

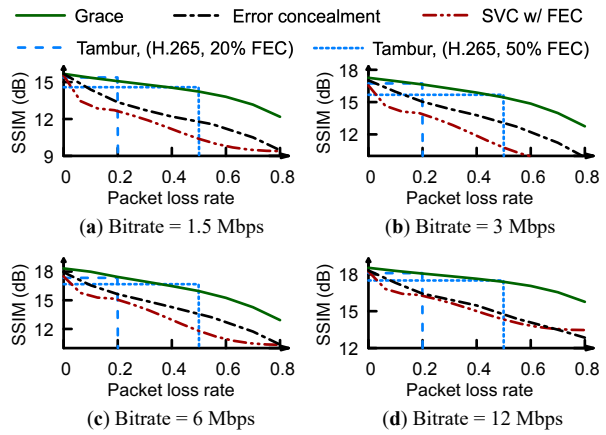


Figure 9: *Video quality of each scheme under different packet loss rates when videos are encoded at different bitrates.*

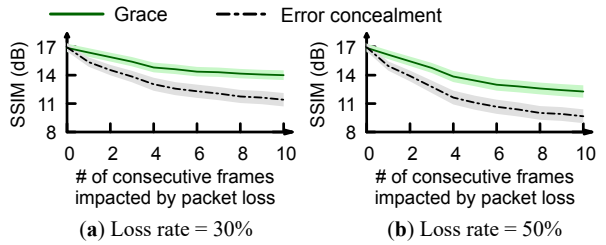


Figure 10: *Stress test of applying persistent packet loss on consecutive frames.*

when the encoded bitrates of all schemes are set to 1.5, 3, 6, and 12 Mbps. Compared with the baselines, GRACE achieves a more graceful and less pronounced quality decline as packet loss increases. Figure 10 further stress tests the loss resilience of GRACE against neural error concealment (the most competitive baseline), when a 30% or 50% packet loss is applied to 1 to 10 consecutive frames without the encoder and decoder synchronizing their states. Although the figure shows that both methods experience quality degradation, GRACE markedly surpasses the neural error concealment baseline in these extreme conditions. Figure 11 visualizes their decoded images after a 50% packet loss is applied to three consecutive frames, confirming that the image decoded by GRACE has less visual distortion.

Compression efficiency: We verify whether GRACE's compression efficiency under no packet loss is on par with H.264 and H.265, which are advanced video codecs designed for high compression efficiency rather than loss resilience. Figure 12 groups the test videos by resolution. On low bitrates, GRACE demonstrates similar compression efficiency as H.264 and marginally underperforms H.265 on both 720p and 1080p videos. On high bitrates (over 3 Mbps for 720p and 6 Mbps for 1080p), GRACE's compression efficiency matches or even surpasses H.265. Compared against Tambur with a persistent 50% FEC redundancy, GRACE achieves a better quality-bitrate tradeoff across the entire bitrate range.

Impact of video content on compression efficiency: To

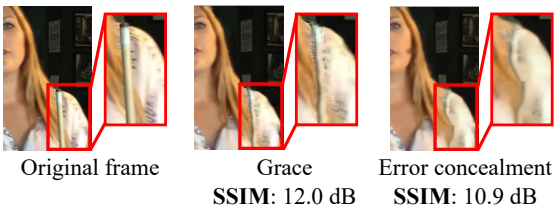


Figure 11: *Sample images decoded by* GRACE *and error concealment under a 50% packet loss on three consecutive frames.* GRACE *achieves less image distortion.*

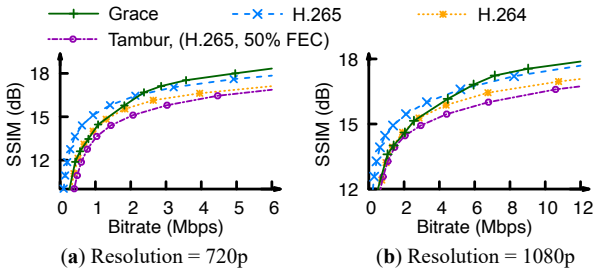


Figure 12: *Quality-size tradeoff of* GRACE *on videos with different resolution. Overall,* GRACE *is better than H.264 and slightly worse than H.265 in terms of compression efficiency.*

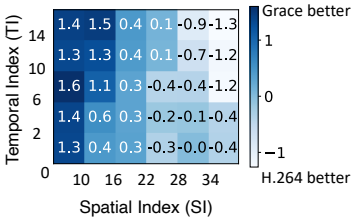
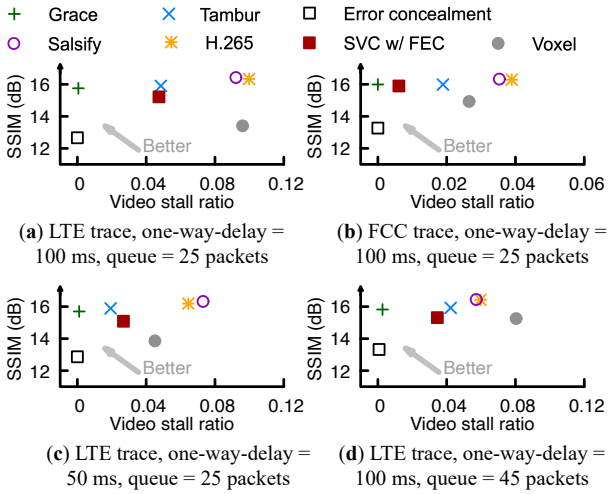


Figure 13: *Mean difference in SSIM (dB) between* GRACE *and H.264 on videos grouped by SI and TI. At the same bitrate (5 Mbps),* GRACE *achieves better video quality than H.264 on low-SI videos but lags behind H.264 on high-SI videos.*

understand the impact of video content on GRACE's compression efficiency, we group the video content based on spatial index (SI) and temporal index (TI), which are established metrics for assessing the spatiotemporal complexity of videos [58]. Figure 13 presents the average gain of GRACE over H.264 in terms of SSIM for videos in each SI-TI combination, encoded at a bitrate of 5 Mbps. The results indicate that GRACE's compression efficiency has a higher advantage over H.264 for videos with low spatial complexity, but this advantage diminishes as the spatial index increases. For a more thorough understanding of GRACE's behavior, Appendix C.5 also shows an example where GRACE performs poorly.

5.3 Video quality vs. realtimeness/smoothness

Figures 14a evaluates GRACE against baselines in terms of average quality (SSIM) and video stall ratio (a smoothness metric) using the network traces from the LTE dataset, under a one-way network delay of 100 ms and a drop-tail queue of 25 packets. Although the SSIM of GRACE is slightly lower than that of the baselines with the highest average SSIM, GRACE



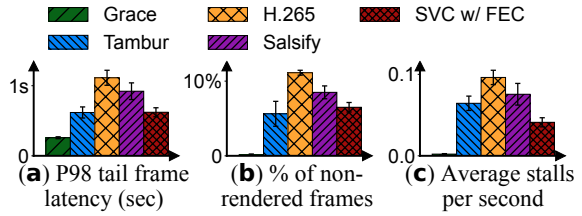
(a) LTE trace, one-way-delay = 100 ms, queue = 25 packets

(b) FCC trace, one-way-delay = 100 ms, queue = 25 packets

(c) LTE trace, one-way-delay = 50 ms, queue = 25 packets

(d) LTE trace, one-way-delay = 100 ms, queue = 45 packets

Figure 14: *End-to-end simulation results over different network traces, one-way delays and network queue lengths.*



(a) P98 tail frame latency (sec)

(b) % of non-rendered frames

(c) Average stalls per second

Figure 15: GRACE *outperforms other baselines on different metrics of realtimeness and smoothness. Updated this figure: added error bar and y-axis*

significantly reduces the video stall ratio.

We repeat the test on a different dataset (FCC) under the same network setup (Figures 14b), with a lower one-way network delay of 50 ms (Figure 14c), and with a longer queue length of 45 packets (Figure 14d). In all settings, GRACE maintains a video stall ratio below 0.5%, whereas the baselines have 4–32× more video stalls, except for the error concealment baseline, which yields a 3dB lower SSIM compared with GRACE. This is because when packet loss happens, GRACE can still decode the frame, while the baselines other than error concealment may experience video stalls due to either skipping frames (e.g., Salsify or Voxel) or waiting for retransmission packets (FEC and SVC).

Figure 15 compares GRACE with the baselines using other realtimeness and smoothness metrics, with the one-way delay set to 100 ms and the queue length set to 25 packets over the LTE traces. For clarity, we only include baselines with comparable average SSIMs in this figure (excluding Voxel and error concealment). While achieving similar video quality, GRACE reduces the 98th percentile frame delay by a factor of 2–5× and non-rendered frames by up to 95%. In §C.7, we also evaluate GRACE with a different congestion control algorithm—Salsify's CC [45].

Figure 16 provides a concrete example of GRACE's behavior. The bandwidth drops from 8 Mbps to 2 Mbps at 1.5 s,

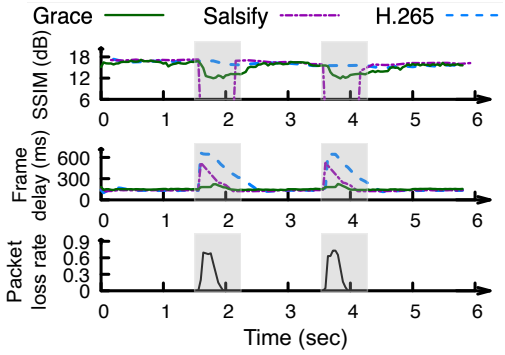


Figure 16: GRACE *achieves lower delay and maintains decent visual quality during sudden bandwidth drops: its delay is lower than both baselines while rendering more frames than Salsify without frame skipping or packet retransmission.*

lasting for 800 ms, before returning to 8 Mbps (another bandwidth drop occurs at 3.5 s and lasts for the same duration). During each drop, GRACE's delay does not experience a sharp increase as the baselines. Salsify is the second best owing to its frame skipping while H.265 must wait for retransmissions. In this experiment, both GRACE and Salsify use the same CC, leading to similar qualities on frames not skipped by Salsify. However, during congestion, GRACE's quality degrades only marginally without skipping any frames, limiting the drop of SSIM to less than 4 dB even when more than 10 consecutive frames encounter a packet loss of over 50%. With the assistance of state resync (§4.2), GRACE's quality resumes quickly (within 1 RTT) after packet losses.

User study: To validate GRACE's effectiveness, we conducted an IRB-approved user study, collecting 960 user ratings from 240 Amazon MTurk workers [1]. We first choose a few genres based on the real-time video streaming use cases, including cloud gaming, real-time sports events, daily human activities, and video conferencing. Then, we randomly selected 8 video clips from the UGC dataset [98]. These video clips were streamed using GRACE, Salsify codec, WebRTC with default FEC, and H.265 with Tambur. (A screenshot of each video clip is shown in Figure 26 in Appendix.) The sampled videos have a similar distribution of quality, realtimeness, and smoothness as seen in Figure 14. Following [22], when an MTurk user signs up for the user study, they are randomly assigned to rate their user experience on a scale of 1–5 for the videos delivered through different methods. Figure 17 displays the mean opinion score (MOS) for each video, confirming that the videos rendered by GRACE are consistently favored by real users.

5.4 Microbenchmarking

Encoding/decoding latency breakdown: Figure 18 shows a breakdown of the encoding and decoding delays of GRACE on an Nvidia A40 GPU (5× cheaper and 3× slower than Nvidia A100). GRACE encodes and decodes a 720p frame within 29.7 ms (33 fps) and 19.5 ms (51.2 fps), respectively. It can

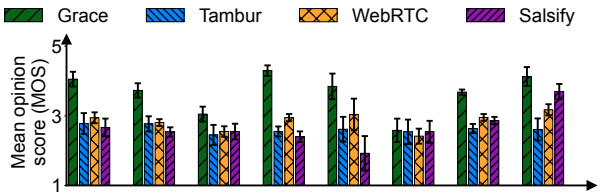


Figure 17: *User study experiment shows that videos streamed by* GRACE *are consistently favored by real users. The error bar shows the standard deviation of the mean.*

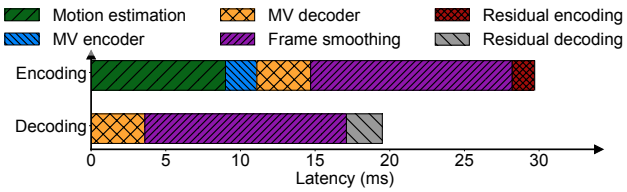


Figure 18: *Latency breakdown of GPU-based encoding and decoding of* GRACE *on a 720p frame.*

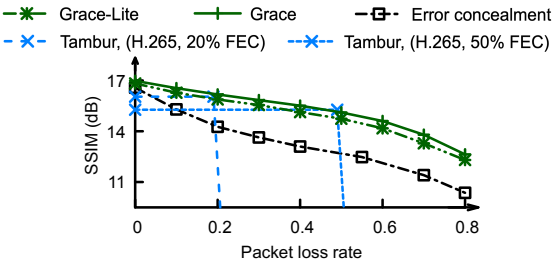


Figure 19: GRACE-*Lite realizes similar loss resilience to* GRACE *and outperforms other baselines.*

also encode/decode 480p video at 65.8 fps/104.1 fps.

This breakdown also carries several implications. First, the fast resync logic (§4.2) requires the encoder to run the MV decoder and residual decoder, which together only consume 6 ms on a 720p frame, allowing resync to complete with a minimal increase in encoding delay. Moreover, GRACE may need to encode a frame multiple times as explained in §4.3, but the extra overhead only involves residual encoding, which takes only 1.5 ms on a 720p frame.

Speed optimization in GRACE-Lite: With the optimizations described in §4.3, GRACE-Lite reduces the encoding delay of a 720 frame on iPhone 14 Pro from 314 ms to 38.1 ms, and the decoding delay from 239 ms to 14.4 ms. We also report GRACE-Lite's speed on CPUs with OpenVINO compilation in Appendix C.9. Figure 19 compares the loss resilience of GRACE-Lite and GRACE with the two most competitive baselines in §5.2—neural error concealment and Tambur. At the same packet loss, GRACE-Lite achieves slightly lower quality than GRACE, yet it still outperforms other baselines.

Impact of joint training: Figure 20 compares GRACE with its two variants: GRACE-P and GRACE-D, showing that both variants have lower levels of loss resilience than GRACE due to not jointly training the encoder and decoder. Appendix C.10 shows an example of frames decoded by the variants.

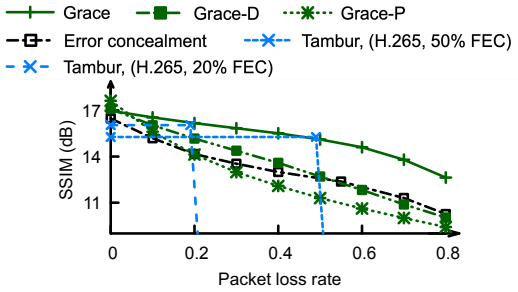


Figure 20: *Although* GRACE-*D and* GRACE-*P attain slightly better quality than* GRACE *in the absence of packet loss, they are much less resilient to loss than* GRACE.

6 Limitation

The current implementation of GRACE still has several limitations. First, it is not optimized enough to run at 30 fps on very resource-constrained devices that barely sustain a classic video codec. For instance, achieving real-time encoding and decoding on regular CPUs (e.g., Intel Xeon Silver 4216) still requires 32 cores (§C.9). Secondly, due to its use of NVC, GRACE may have lower compression efficiency than traditional handcrafted codecs on some video content that deviates a lot from the training data of NVC. For instance, its compression efficiency is worse than H.26x on videos with high spatial complexity (§5.2). In rare instances, GRACE is observed to fail to accurately reconstruct original frames under high packet losses. Third, our focus with GRACE is on unicast video communication rather than multiparty conferencing. We hope GRACE can inspire future work to address these limitations. Potential avenues include democratizing GRACE on more devices by embracing the recent advancements in hardware [23, 24], distilling more lightweight models suitable for less powerful devices. We acknowledge there is not a good solution to address GRACE's generalization issue, which is a problem not unique to GRACE but inherent in general NVCs. We hope that future measurement studies may shed light on the generalization of NVCs and contribute to their improvement.

7 Conclusion

This paper presents GRACE, a real-time video system designed for loss resilience, preserving quality of experience (QoE) for users across diverse packet losses. GRACE enhances loss resilience by jointly training a neural encoder and decoder under a spectrum of packet losses. It attains video quality on par with conventional codecs in the absence of packet loss, and exhibits a less pronounced quality degradation as packet loss escalates, outperforming existing loss-resilient methods.

8 Acknowledgement

We thank the anonymous reviewers and our shepherd Dongsu Han. This project is supported by NSF CNS 2146496, 2131826, 2313190, 1901466, and UChicago CERES Center.

References

[1] Amazon Mechanical Turk. https://www.mturk.com/.

[2] Aurora5 HEVC Test Results. https://www.visionular.com/en/putting-the-aurora5-hevc-encoder-to-the-test/.

[3] Bringing Zoom's end-to-end optimizations to WebRTC. https://blog.livekit.io/livekit-one-dot-zero/.

[4] Context-adaptive binary arithmetic coding. https://en.wikipedia.org/wiki/Context-adaptive_binary_arithmetic_coding.

[5] Core ML Documentation. https://developer.apple.com/documentation/coreml.

[6] Features of WebRTC VR Streaming. https://flashphoner.com/features-of-webrtc-vr-streaming/.

[7] FFmpeg streaming guide. http://trac.ffmpeg.org/wiki/StreamingGuide.

[8] Linear Congruential Generator. https://en.wikipedia.org/wiki/Linear_congruential_generator.

[9] Mamahi Cellular traces. https://github.com/ravinet/mahimahi/tree/master/traces.

[10] Measuring Broadband Raw Data Releases. https://www.fcc.gov/oet/mba/raw-data-releases.

[11] Open Source Cloud Gaming with WebRTC. https://webrtchacks.com/open-source-cloud-gaming-with-webrtc/.

[12] SI/TI calculation tools. https://github.com/VQEG/siti-tools.

[13] torchac: Fast Arithmetic Coding for PyTorch. https://github.com/fab-jul/torchac.

[14] Torch.compile tutorial . https://pytorch.org/tutorials/intermediate/torch_compile_tutorial.html.

[15] VP9 encoding/decoding performance vs. HEVC/H.264. https://blogs.gnome.org/rbultje/2015/09/28/vp9-encodingdecoding-performance-vs-hevch-264/.

[16] WebRTC and IoT Applications. https://rtcweb.in/webrtc-and-iot-applications/.

[17] WebRTC Cloud Gaming: Unboxing Stadia. https://webrtc.ventures/2021/02/webrtc-cloud-gaming-unboxing-stadia/.

[18] WebRTC: Enabling Collaboration Augmented Reality App. https://arvrjourney.com/webrtc-enabling-collaboration-cebdd4c9ce06?gi=e19b1c0f65c0.

[19] WebRTC in IoT: What is the Intersection Point? https://mobidev.biz/blog/webrtc-real-time-communication-for-the-internet-of-things.

[20] What powers Google Meet and Microsoft Teams? WebRTC Demystified. https://levelup.gitconnected.com/what-powers-google-meet-and-microsoft-teams-webrtc-demystified-step-by-step-tutorial-e0cb422010f7.

[21] Better Portable Graphics. https://bellard.org/bpg/, 2014.

[22] SENSEI: Aligning Video Streaming Quality with Dynamic User Sensitivity, author=Zhang, Xu and Ou, Yiyang and Sen, Siddhartha and Jiang, Junchen. In *18th USENIX Symposium on Networked Systems Design and Implementation (NSDI 21)*, pages 303–320, 2021.

[23] Deploying Transformers on the Apple Neural Engine. https://machinelearning.apple.com/research/neural-engine-transformers, 2022.

[24] Harnessing the NVIDIA Ada Architecture for Frame-Rate Up-Conversion in the NVIDIA Optical Flow SDK. https://developer.nvidia.com/blog/harnessing-the-nvidia-ada-architecture-for-frame-rate-up-conversion-in-the-nvidia-optical-flow-sdk/, 2023.

[25] Asma Ben Abdallah, Amin Zribi, Ali Dziri, Fethi Tlili, and Michel Terré. H.264/AVC video transmission over UWB AV PHY IEEE 802.15. 3c using UEP and adaptive modulation techniques. In *2019 International Conference on Advanced Communication Technologies and Networking (CommNet)*, pages 1–6. IEEE, 2019.

[26] Doreid Ammar, Katrien De Moor, Min Xie, Markus Fiedler, and Poul Heegaard. Video QoE killer and performance statistics in WebRTC-based video communication. In *2016 IEEE Sixth International Conference on Communications and Electronics (ICCE)*, pages 429–436. IEEE, 2016.

[27] Anurag Arnab, Mostafa Dehghani, Georg Heigold, Chen Sun, Mario Lučić, and Cordelia Schmid. Vivit: A video vision transformer. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 6836–6846, 2021.

[28] Ahmed Badr, Ashish Khisti, Wai-tian Tan, Xiaoqing Zhu, and John Apostolopoulos. FEC for VoIP using dual-delay streaming codes. In *IEEE INFOCOM 2017-IEEE Conference on Computer Communications*, pages 1–9. IEEE, 2017.

[29] Niklas Blum, Serge Lachapelle, and Harald Alvestrand. WebRTC-Realtime Communication for the Open Web Platform: What was once a way to bring audio and video to the web has expanded into more use cases we could ever imagine. *Queue*, 19(1):77–93, 2021.

[30] Eirina Bourtsoulatze, David Burth Kurka, and Deniz Gündüz. Deep joint source-channel coding for wireless image transmission. *IEEE Transactions on Cognitive Communications and Networking*, 5(3):567–579, 2019.

[31] Gaetano Carlucci, Luca De Cicco, Stefan Holmer, and Saverio Mascolo. Analysis and design of the google congestion control for web real-time communication (WebRTC). In *Proceedings of the 7th International Conference on Multimedia Systems*, pages 1–12, 2016.

[32] Fabrizio Carpi, Christian Häger, Marco Martalò, Riccardo Raheli, and Henry D. Pfister. Reinforcement Learning for Channel Coding: Learned Bit-Flipping Decoding. In *2019 57th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, pages 922–929, 2019.

[33] Jeff Castura and Yongyi Mao. Rateless coding over fading channels. *IEEE communications letters*, 10(1):46–48, 2006.

[34] Ya-Liang Chang, Zhe Yu Liu, Kuan-Ying Lee, and Winston Hsu. Free-form video inpainting with 3d gated convolution and temporal patchgan. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 9066–9075, 2019.

[35] Sheng Cheng, Han Hu, and Xinggong Zhang. ABRF: Adaptive BitRate-FEC Joint Control for Real-Time Video Streaming. *IEEE Transactions on Circuits and Systems for Video Technology*, 2023.

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

[37] Kristy Choi, Kedar Tatwawadi, Aditya Grover, Tsachy Weissman, and Stefano Ermon. Neural joint source-channel coding. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 1182–1192. PMLR, 09–15 Jun 2019.

[38] Wen-Jeng Chu and Jin-Jang Leou. Detection and concealment of transmission errors in H.261 images. *IEEE Transactions on Circuits and Systems for Video Technology*, 8(1):74–84, 1998.

[39] Mauro Conti, Simone Milani, Ehsan Nowroozi, and Gabriele Orazi. Do Not Deceive Your Employer with a Virtual Background: A Video Conferencing Manipulation-Detection System. *arXiv preprint arXiv:2106.15130*, 2021.

[40] Mallesham Dasari, Kumara Kahatapitiya, Samir R. Das, Aruna Balasubramanian, and Dimitris Samaras. Swift: Adaptive video streaming with layered neural codecs. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*, pages 103–118, Renton, WA, April 2022. USENIX Association.

[41] Sandesh Dhawaskar Sathyanarayana, Kyunghan Lee, Dirk Grunwald, and Sangtae Ha. Converge: QoE-driven Multipath Video Conferencing over WebRTC. In *Proceedings of the ACM SIGCOMM 2023 Conference*, pages 637–653, 2023.

[42] Yves Dhondt and Peter Lambert. Flexible Macroblock Ordering: an error resilience tool in H. 264/AVC. In *5th FTW PhD Symposium*. Ghent University. Faculty of Engineering, 2004.

[43] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

[44] Patrick Esser, Robin Rombach, and Bjorn Ommer. Taming transformers for high-resolution image synthesis. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 12873–12883, 2021.

[45] 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 *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*, pages 267–282, Renton, WA, April 2018. USENIX Association.

[46] Chen Gao, Ayush Saraf, Jia-Bin Huang, and Johannes Kopf. Flow-edge guided video completion. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XII 16*, pages 713–729. Springer, 2020.

[47] Boni García, Micael Gallego, Francisco Gortázar, and Antonia Bertolino. Understanding and estimating quality of experience in WebRTC applications. *Computing*, 101:1585–1607, 2019.

[48] Boni García, Francisco Gortázar, Micael Gallego, and Andrew Hines. Assessment of qoe for video and audio in webrtc applications using full-reference models. *Electronics*, 9(3):462, 2020.

[49] Dan Grois, Detlev Marpe, Amit Mulayoff, Benaya Itzhaky, and Ofer Hadar. Performance comparison of h. 265/mpeg-hevc, vp9, and h. 264/mpeg-avc encoders. In *2013 Picture Coding Symposium (PCS)*, pages 394–397. IEEE, 2013.

[50] Tobias Gruber, Sebastian Cammerer, Jakob Hoydis, and Stephan ten Brink. On deep learning-based channel decoding. In *2017 51st Annual Conference on Information Sciences and Systems (CISS)*, pages 1–6, 2017.

[51] Deniz Gündüz, Paul de Kerret, Nicholas D Sidiropoulos, David Gesbert, Chandra R Murthy, and Mihaela van der Schaar. Machine learning in the air. *IEEE Journal on Selected Areas in Communications*, 37(10):2184–2199, 2019.

[52] Zhaoyuan He, Yifan Yang, Shuozhe Li, Diyuan Dai, and Lili Qiu. Neural Video Recovery for Cloud Gaming. *arXiv preprint arXiv:2307.07847*, 2023.

[53] Zhaoyuan He, Yifan Yang, Lili Qiu, and Kyoungjun Park. Real-Time Neural Video Recovery and Enhancement on Mobile Devices. *arXiv preprint arXiv:2307.12152*, 2023.

[54] Stefan Holmer, Mikhal Shemer, and Marco Paniconi. Handling packet loss in WebRTC. In *2013 IEEE International Conference on Image Processing*, pages 1860–1864. IEEE, 2013.

[55] Zhihao Hu, Guo Lu, and Dong Xu. FVC: A new framework towards deep video compression in feature space. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1502–1511, 2021.

[56] Ismaeil Ismaeil, Shahram Shirani, Faouzi Kossentini, and Rabab Ward. An efficient, similarity-based error concealment method for block-based coded images. In *Proceedings 2000 International Conference on Image Processing (Cat. No. 00CH37101)*, volume 3, pages 388–391. IEEE, 2000.

[57] ITU-T. Recommendation G.114, one-way transmission time. *Series G: Transmission Systems and Media, Digital Systems and Networks, Telecommunication Standardization Sector of ITU*, 2003.

[58] P ITU-T RECOMMENDATION. Subjective video quality assessment methods for multimedia applications. 1999.

[59] Jaeyeon Kang, Seoung Wug Oh, and Seon Joo Kim. Error compensation framework for flow-guided video inpainting. In *Computer Vision–ECCV 2022: 17th European Conference, Tel Aviv, Israel, October 23–27, 2022, Proceedings, Part XV*, pages 375–390. Springer, 2022.

[60] Amin Abdel Khalek, Constantine Caramanis, and Robert W Heath. A cross-layer design for perceptual optimization of H. 264/SVC with unequal error protection. *IEEE Journal on selected areas in Communications*, 30(7):1157–1171, 2012.

[61] Jaehong Kim, Youngmok Jung, Hyunho Yeo, Juncheol Ye, and Dongsu Han. Neural-enhanced live streaming: Improving live video ingest via online learning. In *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication*, pages 107–125, 2020.

[62] Diederik P. Kingma and Max Welling. Auto-Encoding Variational Bayes. In *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*, 2014.

[63] Vineeth Shetty Kolkeri. *Error concealment techniques in H. 264/AVC, for video transmission over wireless networks*. PhD thesis, The University of Texas at Arlington, 2009.

[64] Sunil Kumar, Liyang Xu, Mrinal K Mandal, and Sethuraman Panchanathan. Error resiliency schemes in H. 264/AVC standard. *Journal of Visual Communication and Image Representation*, 17(2):425–450, 2006.

[65] David Burth Kurka and Deniz Gündüz. Deepjscc-f: Deep joint source-channel coding of images with feedback. *IEEE Journal on Selected Areas in Information Theory*, 1(1):178–193, 2020.

[66] Peter Lambert, Wesley De Neve, Yves Dhondt, and Rik Van de Walle. Flexible macroblock ordering in H. 264/AVC. *Journal of Visual Communication and Image Representation*, 17(2):358–375, 2006.

[67] Tianhong Li, Vibhaalakshmi Sivaraman, Lijie Fan, Mohammad Alizadeh, and Dina Katabi. Reparo: Loss-Resilient Generative Codec for Video Conferencing. *arXiv preprint arXiv:2305.14135*, 2023.

[68] Weiping Li. Overview of fine granularity scalability in MPEG-4 video standard. *IEEE Transactions on circuits and systems for video technology*, 11(3):301–317, 2001.

[69] Zhen Li, Cheng-Ze Lu, Jianhua Qin, Chun-Le Guo, and Ming-Ming Cheng. Towards an end-to-end framework for flow-guided video inpainting. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 17562–17571, 2022.

[70] Jingyun Liang, Jiezhang Cao, Guolei Sun, Kai Zhang, Luc Van Gool, and Radu Timofte. Swinir: Image restoration using swin transformer. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 1833–1844, 2021.

[71] Rui Liu, Hanming Deng, Yangyi Huang, Xiaoyu Shi, Lewei Lu, Wenxiu Sun, Xiaogang Wang, Jifeng Dai, and Hongsheng Li. FuseFormer: Fusing Fine-Grained Information in Transformers for Video Inpainting. In *ICCV*, 2021.

[72] Yunzhuo Liu, Bo Jiang, Tian Guo, Ramesh K. Sitaraman, Don Towsley, and Xinbing Wang. Grad: Learning for overhead-aware adaptive video streaming with scalable video coding. In *Proceedings of the 28th ACM International Conference on Multimedia*, MM '20, page 349–357, New York, NY, USA, 2020. Association for Computing Machinery.

[73] Guo Lu, Wanli Ouyang, Dong Xu, Xiaoyun Zhang, Chunlei Cai, and Zhiyong Gao. DVC: An end-to-end deep video compression framework. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 10998–11007, 2019.

[74] Rong Luo and Bin Chen. A Hierarchical Scheme of Flexible Macroblock Ordering for ROI based H.264/AVC Video Coding. In *2008 10th International Conference on Advanced Communication Technology*, volume 3, pages 1579–1582, 2008.

[75] Yi Ma, Yongqi Zhai, and Ronggang Wang. DeepFGS: Fine-Grained Scalable Coding for Learned Image Compression. *arXiv preprint arXiv:2201.01173*, 2022.

[76] David JC MacKay. Fountain codes. *IEE Proceedings-Communications*, 152(6):1062–1068, 2005.

[77] David JC MacKay and Radford M Neal. Near Shannon limit performance of low density parity check codes. *Electronics letters*, 33(6):457–458, 1997.

[78] Kyle MacMillan, Tarun Mangla, James Saxon, and Nick Feamster. Measuring the performance and network utilization of popular video conferencing applications. In *Proceedings of the 21st ACM Internet Measurement Conference*, pages 229–244, 2021.

[79] Michael Mathieu, Camille Couprie, and Yann LeCun. Deep multi-scale video prediction beyond mean square error. *arXiv preprint arXiv:1511.05440*, 2015.

[80] 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 *Proceedings of the ACM SIGCOMM 2022 Conference*, pages 193–206, 2022.

[81] Cholman Nam, Changgon Chu, Taeguk Kim, and Sokmin Han. A novel motion recovery using temporal and spatial correlation for a fast temporal error concealment over H. 264 video sequences. *Multimedia Tools and Applications*, 79:1221–1240, 2020.

[82] Ravi Netravali, Anirudh Sivaraman, Somak Das, Ameesh Goyal, Keith Winstein, James Mickens, and Hari Balakrishnan. Mahimahi: Accurate Record-and-Replay for HTTP. In *2015 USENIX Annual Technical Conference (USENIX ATC 15)*, pages 417–429, 2015.

[83] Mirko Palmer, Malte Appel, Kevin Spiteri, Balakrishnan Chandrasekaran, Anja Feldmann, and Ramesh K. Sitaraman. Voxel: Cross-layer optimization for video streaming with imperfect transmission. In *Proceedings of the 17th International Conference on Emerging Networking EXperiments and Technologies*, CoNEXT '21, page 359–374, New York, NY, USA, 2021. Association for Computing Machinery.

[84] Jan Peters and Stefan Schaal. Reinforcement learning of motor skills with policy gradients. *Neural networks*, 21(4):682–697, 2008.

[85] Devdeep Ray, Connor Smith, Teng Wei, David Chu, and Srinivasan Seshan. SQP: Congestion Control for Low-Latency Interactive Video Streaming. *arXiv preprint arXiv:2207.11857*, 2022.

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

[87] Arun Sankisa, Arjun Punjabi, and Aggelos K Katsaggelos. Video error concealment using deep neural networks. In *2018 25th IEEE International Conference on Image Processing (ICIP)*, pages 380–384. IEEE, 2018.

[88] Thomas Schierl, Thomas Stockhammer, and Thomas Wiegand. Mobile video transmission using scalable video coding. *IEEE transactions on circuits and systems for video technology*, 17(9):1204–1217, 2007.

[89] 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*, 17(9):1103–1120, 2007.

[90] Taveesh Sharma, Tarun Mangla, Arpit Gupta, Junchen Jiang, and Nick Feamster. Estimating WebRTC Video QoE Metrics Without Using Application Headers. *arXiv preprint arXiv:2306.01194*, 2023.

[91] Yibo Shi, Yunying Ge, Jing Wang, and Jue Mao. AlphaVC: High-Performance and Efficient Learned Video Compression. In *Computer Vision–ECCV 2022: 17th European Conference, Tel Aviv, Israel, October 23–27, 2022, Proceedings, Part XIX*, pages 616–631. Springer, 2022.

[92] Vibhaalakshmi Sivaraman, Pantea Karimi, Vedantha Venkatapathy, Mehrdad Khani, Sadjad Fouladi, Mohammad Alizadeh, Frédo Durand, and Vivienne Sze. Gemino: Practical and Robust Neural Compression for Video Conferencing. *arXiv preprint arXiv:2209.10507*, 2022.

[93] Keyu Tan and Alan Pearmain. A new error resilience scheme based on FMO and error concealment in H. 264/AVC. In *2011 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 1057–1060. IEEE, 2011.

[94] Wai-tian Tan and Avideh Zakhor. Multicast transmission of scalable video using receiver-driven hierarchical FEC. In *Packet Video Workshop*, volume 99, 1999.

[95] Wai-Tian Tan and Avideh Zakhor. Video multicast using layered fec and scalable compression. *IEEE Transactions on circuits and systems for video technology*, 11(3):373–386, 2001.

[96] Yao Wang and Qin-Fan Zhu. Error control and concealment for video communication: A review. *Proceedings of the IEEE*, 86(5):974–997, 1998.

[97] Yi Wang, Xiaoqiang Guo, Feng Ye, Aidong Men, and Bo Yang. A novel temporal error concealment framework in H. 264/AVC. In *2013 IEEE International Conference on Multimedia and Expo (ICME)*, pages 1–6. IEEE, 2013.

[98] Yilin Wang, Sasi Inguva, and Balu Adsumilli. YouTube UGC dataset for video compression research. In *2019 IEEE 21st International Workshop on Multimedia Signal Processing (MMSP)*, pages 1–5. IEEE, 2019.

[99] Stephan Wenger and Michael Horowitz. Scattered slices: a new error resilience tool for H. 26L. *JVT-B027*, 2, 2002.

[100] Stephen B. Wicker and Vijay K. Bhargava. *Reed-Solomon codes and their applications*. John Wiley & Sons, 1999.

[101] Jiangkai Wu, Yu Guan, Qi Mao, Yong Cui, Zongming Guo, and Xinggong Zhang. ZGaming: Zero-Latency 3D Cloud Gaming by Image Prediction. In *Proceedings of the ACM SIGCOMM 2023 Conference*, pages 710–723, 2023.

[102] Chongyang Xiang, Jiajun Xu, Chuan Yan, Qiang Peng, and Xiao Wu. Generative adversarial networks based error concealment for low resolution video. In *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 1827–1831. IEEE, 2019.

[103] Tianfan Xue, Baian Chen, Jiajun Wu, Donglai Wei, and William T Freeman. Video enhancement with task-oriented flow. *International Journal of Computer Vision*, 127(8):1106–1125, 2019.

[104] Francis Y. Yan, Hudson Ayers, Chenzhi Zhu, Sadjad Fouladi, James Hong, Keyi Zhang, Philip Levis, and Keith Winstein. Learning *in situ*: a randomized experiment in video streaming. In *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*, pages 495–511, Santa Clara, CA, February 2020. USENIX Association.

[105] Ren Yang, Fabian Mentzer, Luc Van Gool, and Radu Timofte. Learning for video compression with recurrent auto-encoder and recurrent probability model. *IEEE Journal of Selected Topics in Signal Processing*, 15(2):388–401, 2020.

[106] Hyunho Yeo, Hwijoon Lim, Jaehong Kim, Youngmok Jung, Juncheol Ye, and Dongsu Han. NeuroScaler: neural video enhancement at scale. In *Proceedings of the ACM SIGCOMM 2022 Conference*, pages 795–811, 2022.

[107] Huanhuan Zhang, Anfu Zhou, Yuhan Hu, Chaoyue Li, Guangping Wang, Xinyu Zhang, Huadong Ma, Leilei Wu, Aiyun Chen, and Changhui Wu. Loki: improving long tail performance of learning-based real-time video adaptation by fusing rule-based models. In *Proceedings of the 27th Annual International Conference on Mobile Computing and Networking*, MobiCom '21, page 775–788, New York, NY, USA, 2021. Association for Computing Machinery.

[108] Huanhuan Zhang, Anfu Zhou, Jiamin Lu, Ruoxuan Ma, Yuhan Hu, Cong Li, Xinyu Zhang, Huadong Ma, and Xiaojiang Chen. OnRL: improving mobile video telephony via online reinforcement learning. In *Proceedings of the 26th Annual International Conference*

on Mobile Computing and Networking, MobiCom '20, New York, NY, USA, 2020. Association for Computing Machinery.

[109] Junzi Zhang, Jongho Kim, Brendan O'Donoghue, and Stephen Boyd. Sample efficient reinforcement learning with REINFORCE. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 10887–10895, 2021.

[110] Kaidong Zhang, Jingjing Fu, and Dong Liu. Flow-guided transformer for video inpainting. In *European Conference on Computer Vision*, pages 74–90. Springer, 2022.

[111] Kaidong Zhang, Jingjing Fu, and Dong Liu. Inertia-guided flow completion and style fusion for video inpainting. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 5982–5991, 2022.

[112] Qing Zhang and Guizhong Liu. Error resilient coding of H. 264 using intact long-term reference frames. 2008.

[113] Zenghua Zhao and Shubing Long. RD-Based Adaptive UEP for H. 264 Video Transmission in Wireless Networks. In *2010 International Conference on Multimedia Information Networking and Security*, pages 72–76. IEEE, 2010.

[114] Anfu Zhou, Huanhuan Zhang, Guangyuan Su, Leilei Wu, Ruoxuan Ma, Zhen Meng, Xinyu Zhang, Xiufeng Xie, Huadong Ma, and Xiaojiang Chen. Learning to coordinate video codec with transport protocol for mobile video telephony. In *The 25th Annual International Conference on Mobile Computing and Networking*, pages 1–16, 2019.

[115] Jie Zhou, Bo Yan, and Hamid Gharavi. Efficient motion vector interpolation for error concealment of H. 264/AVC. *IEEE Transactions on Broadcasting*, 57(1):75–80, 2010.

[116] X Zhu, P Pan, M Ramalho, S Mena, P Jones, J Fu, S D'Aronco, and C Ganzhorn. Nada: A unified congestion control scheme for real-time media, draft-ietf-rmcat-nada-02. *Internet Engineering Task Force, IETF*, 2016.

[117] Xutong Zuo, Yong Cui, Xin Wang, and Jiayu Yang. Deadline-aware Multipath Transmission for Streaming Blocks. In *IEEE INFOCOM 2022-IEEE Conference on Computer Communications*, pages 2178–2187. IEEE, 2022.

A Details of NVC architecture and training

A.1 More details on GRACE's NVC model

Grace uses the exact same model architecture as the original DVC model [73]. With an RGB input image of size $C \times H \times W$, where H, W are the height and width of the image, and $C = 3$ is number of channels in RGB images, the encoder neural network will encode the image into a compressed motion vector of size $128 \times (H/16) \times (W/16)$ and a compressed residual of size $96 \times (H/16) \times (W/16)$. Then those two compressed features will be quantized and converted into bytesteam using entropy encoding.

When we finetune the DVC model to get our GRACE's loss resilient model, we train on the 90k Vimeo Dataset, with batch size of 4, learning rate of 10^{-4} and learning rate decay of 0.1, and an Adam optimizer.

A.2 Making GRACE trainable

Since P is a non-differentiable random function, the gradient of the expectation of D in Eq. 2 cannot be directly calculated. To address this issue, we use the REINFORCE trick [62] for reparameterization. First, given the packet loss distribution $P(\mathbf{y})$, we can apply the differentiation property of logarithms to get

$$\nabla_\phi P(\mathbf{y}) = P(\mathbf{y}) \nabla_\phi \log P(\mathbf{y})$$

Therefore, our gradient of the expectation of $D(g_\theta(\mathbf{y}), \mathbf{x})$ becomes

$$\begin{aligned} &\nabla_\phi \mathbb{E}_{\mathbf{y} \sim P(\mathbf{y})}([D(g_\theta(\mathbf{y}), \mathbf{x})]) \\ &\qquad = \mathbb{E}_{\mathbf{y} \sim P(\mathbf{y})}([D(g_\theta(\mathbf{y}), \mathbf{x}) \nabla_\phi \log P(\mathbf{y})]) \end{aligned} \quad (3)$$

which can be estimated using Monte-Carlo sampling $\approx \frac{1}{N} \sum_{i=1}^{N} D(g_\theta(\mathbf{y}_i), \mathbf{x}) \nabla_\phi \log P(\mathbf{y}_i)$. Since in our application, the loss is an independent and identically distributed random variable, the gradient evaluates to either 0 or 1, hence we propagate the gradients for the encoder only for $D(g_\theta(\mathbf{y}_i), \mathbf{x})$ where $P(\mathbf{y}_i) = 1$.

B Realtime video framework for GRACE

B.1 Fast re-encoding and re-decoding under loss

In GRACE's NVC, the most time-consuming components are motion estimation NN and frame smoothing NN, taking 28% and 42% of the total encoding time respectively. Fortunately, we do not need to use them during resync (§4.2). When the packet loss feedback arrives at the encoder, it takes the following steps to generate a new reference frame to re-sync with the decoder (Assuming the loss feedback is for 6th frame and the encoder is about to encode 10th frame)

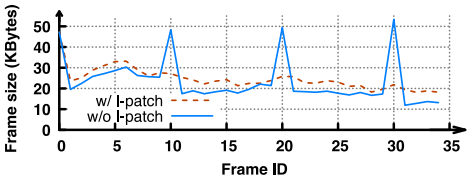


Figure 21: *Encoding each P-frame with a small I-patch leads to smoother frame sizes than naively inserting I-frames.*

- First, GRACE re-decodes the motion vector and residuals based on the packet loss feedback for 6th frame. This step needs to run the motion decoder NN and residual decoder NN, which only takes around 18% of the encoding time.
- Second, GRACE apply the cached motion vector and residuals of 7th frame on the "reconstructed" 6th frame to generate the "reconstructed" 7th frame. It applies the same logic on 8th and 9th frame and finally gets the "reconstructed" 9th frame. We do not run frame smoothing NN since the quality of the reference frame does not have a significant impact on compression efficiency. Therefore, this step does not involve any NN inference. It only needs to apply the motion and add the residuals, which takes 1% of the encoding time.
- Finally, GRACE uses the "reconstructed" 9th frame as the reference frame to encode the 10th frame. It is the same as encoding a frame when there are no packet losses. It will add an extra tag to the frame so that the receiver knows which reference frame to use.

To summarize, the encoder side's computational overhead is usually less than 10%. The logic requires the encoder to cache the motion vectors and residuals, but the cached value of frame x can be dropped after receiving the packet loss feedback of that frame.

At the receiver side, when receiving the frame with the extra tag, it will follow the same process as the second step above to generate the same "reconstructed" reference frame as the encoder. Again, the overhead is negligible since it does not require NN inference.

B.2 How GRACE handles I-frames

GRACE uses BPG [21] (also used in H.265) to encode and decode I-frames every 1000 frames. That said, in many NVCs (including DVC), the quality of P-frames will gradually degrade after an I-frame. By simply adding frequent I-frames (e.g., every 10 frames), we can achieve similar average compression efficiency with H.264 and H.265 when they use an optimal I-frame interval. However, since I-frames are larger than P-frames, adding too many I-frames causes frequent spikes in frame size. Instead, GRACE uses an *extra* small square-sized patch as a tiny I-frame, called *I-patch*, on *every P frame*. We split each frame into k patches, and for a window of k frames, each frame is sent with an extra I-patch at a different location, so I-patch "scan through" the whole frame every k frame. By default, $k = 30$ though we empirically found any value between 10 and 30 works well. With I-patch, GRACE does not need to send any I-frames (except the first frame). We use BPG [21] to encode/decode the I-patch. Figure 21 shows that when $k = 10$, I-patch mitigates the sudden size increase caused by I-frames.

It is worth noting that though I-patch encoding can also use a loss-resilient NVC, we do not protect their packet loss to simplify the system design. This is because if each patch will see an I-patch every k frames, so even if one patch is lost, its impact is confined to the next k frames, and empirically, even this impact is marginal since P-frames are still delivered.

B.3 Working with congestion control

GRACE can be integrated with any existing congestion control (CC) algorithms. When combined with GRACE, CC does *not* need to retransmit packets, unless no packets of a frame are received. CC determines the sending rate of packets and the target size of the next frame, while GRACE decides the content in each packet. Therefore, GRACE would not change the properties of the CC, such as fast convergence, oscillation avoidance, and TCP friendliness. In real-time video communication, traditional CC algorithms like GCC [31] typically mitigate packet losses by reducing bandwidth use, due to the non-loss-tolerant nature of conventional video codecs. These codecs necessitate retransmissions when packet loss happens, causing frame delays and video stalls. Conversely, GRACE is designed to handle packet losses by decoding the partially received frames with graceful quality. This capability allows GRACE to employ a more aggressive congestion control strategy, which, while resulting in occasional packet losses, enhances bandwidth utilization. An illustration of this approach can be found in Appendix C.7, where GRACE works with Salsify's congestion control (Sal-CC) [45] that yields a higher average sending rate albeit with increased packet loss.

B.4 Integration in WebRTC

GRACE is implemented with 3K lines of code, in both Python (mostly for NVC NNs) and C++ (for frame delivery and WebRTC integration). The code and trained model of GRACE will be made public upon the publication of this paper. The integration with WebRTC is logically straightforward since GRACE (including I-frame and P-frame encodings) exposes similar interface as the default codec in WebRTC.

We substitute the libvpx VP8 Encoder/Decoder in WebRTC with our GRACE implementation. When the sender encodes a frame, it parses the image data from the `VideoFrame` data structure (YUV format) into `torch.Tensor` (RGB format) and feed it into our GRACE encoder, which will return the encoded result as a byte array. Then the encoded bytes are stored into an `EncodedImage` (class in WebRTC) and sent through the network to the receiver as RTP packets. We modify the

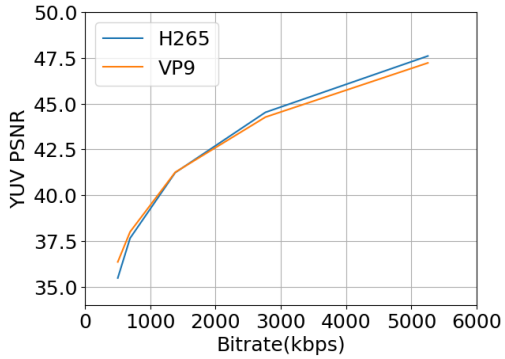


Figure 22: *H265 vs VP9 Encoding Efficiency on Kinetics*

built-in `RtpVideoStreamReceiver` (class in WebRTC) so that the receiver could flexibly decode the received packets even when not all the packets are received. When the receiver decides to decode the frame, it depacketizes the received packets into encoded data. Then it will use the GRACE decoder to decode the image into RGB format and then convert it back to YUV for displaying on the receiver side.

C Supportive details for GRACE's evaluation experiments

C.1 VP9 and H265 Comparison

In our paper we mainly compared with codecs in the H26x family. Since many prior work used VPx codec, we ran a simple experiment to show they have similar efficiency. We randomly chose 12 videos with resolution 1280x720 from the Kinetics dataset we used and compared encoding efficiency between VP9 and H265. We configured VP9 to use speed/quality tradeoff level 8 and set H265 to very-fast, zero-latency, and no B-frame. We confirm that they have similar performance as shown in Fig 22.

C.2 Baseline and testbed implementation details

We provide the extra implementation details of our baselines here:

- **Tambur**: To match the implementation in Tambur's paper [86], we force the codec to not encode any I-frames. Following recent work in real-time video coding [2, 7], we use the `zerolatency` option (no B-frames) and the `fast` preset of H.265. The command line we used to encode a video is `ffmpeg -y -i Video.y4m -c:v libx265 -preset fast -tune zerolatency -x265-params "crf=Q:keyint=3000" output.mp4` where Q controls the quality of the frame.
- **Error concealment**: We employ ECFVI [59], an NN-based error concealment pipeline, to mitigate errors from packet losses with H.265 encoding/decoding. When an incomplete frame is received, it starts a 3-step process to compensate for the errors. First, it uses a neural network to estimate the motion vector of the missing part from the previous N frames. Next, the missing pixel values are propagated from the reference frame using the estimated motion vector. Finally, an inpainting neural network is applied to enhance frame quality and minimize error propagation. We set $N = 5$ during our evaluation.

  ECFVI operates under the assumption that packet loss only corrupts portions of a frame, leaving the rest part (corresponding to the arrived packets) decodable. However, as discussed in §4.1, a single packet loss typically renders an entire frame undecodable in H.264/H.265. To reconcile this, we use flexible macroblock ordering (FMO) technique within the underlying H.265 video codec. This allows different parts of a frame to be encoded and packetized independently into distinct packets. In our baseline implementation, the frame is partitioned into 64×64-pixel blocks and randomly mapped to various packets during packetization. This method introduces a size overhead, as the codec cannot eliminate redundancy among packets. Based on prior works [64, 74, 99], we account for an additional 10% size overhead to ensure that each packet is individually decodable.

  ECFVI is chosen as the baseline for error concealment for two main reasons: *(i)* Its 3-step method is recognized as state-of-the-art within the computer vision research area. It surpasses the prior works that only do motion estimation [87] or inpainting [34]. *(ii)* Similar methods have been adopted by various recent works such as [46], [69], and [111], while ECFVI ranking as the most proficient among them. *(iii)* ECFVI's performance is also on par with or better than other recent error concealment techniques, including those utilizing transformers [71, 110].
- **Voxel** (selective frame skipping): We sort the video frames by the SSIM drop caused by skipping the frame (in real-time video communication, we usually cannot get the quality drop caused by skipping frames in advance. Thus, we are making an idealized assumption that improves the baseline). For 25% frames with the lowest SSIM drop, we use the default error concealment method in H.264/AVC [115] without any packet retransmission, and for the remaining frames (which cause more SSIM drops when skipped), we retransmit all the lost packets. We use a GoP (chunk length) of 4 seconds, which is also used by Voxel.
- **Salsify** (functional codec): We implement the Salsify codec based H.265 with the following two key features: firstly, the encoded frame size never surpasses the target bitrate determined by the underlying congestion control algorithm; secondly, upon packet loss, the encoder can dynamically select a reference frame, enabling subsequent frames to be decoded without resending any packets.

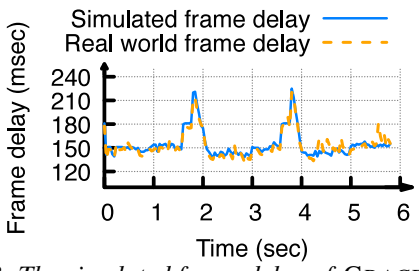


Figure 23: *The simulated frame delay of* GRACE *is close to the real world measured frame delay*

C.3 Simulator validation

Our simulator runs on an Ubuntu 18.04 server with 2 Intel Xeon 4210R CPU, and 256GB memory, with 2 Nvidia A40 GPUS. To validate that the frame delay measured in simulation matches the real-world numbers, we run a real-world emulation using GRACE. Being the same as simulation, we use 2 Nvidia A40 GPUs, one for encoding and one for decoding. The encoder process encodes the video using GRACE's encoder and send the encoded packets through an emulated network. The decoder process decodes the frame using the same logic as mentioned in §4. We compute the real-world frame delay by calculating the difference between the encoding time and the decoding time of a frame. Figure 23 compares the simulated frame delay and real-world measured frame. We use the bandwidth trace same as Figure 16. The result validates that our simulated frame delay is accurate. It is worth noting that we are running real encoding and decoding process in the simulation, hence the calculated frame quality should also be the same as using GRACE in the real world.

C.4 Distribution of video content complexity

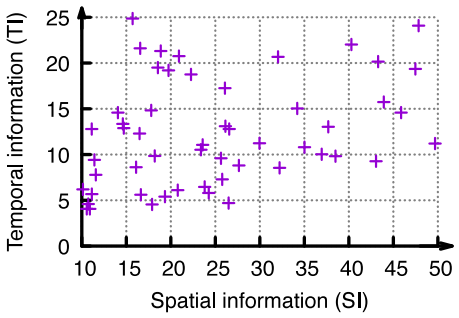


Figure 24: Spatial information (SI) and temporal information (TI) of test videos

To validate the test videos that we use cover different content complexities and movements, we calculate the spatiotemporal complexity of the video. We use Spatial Information (SI) and Temporal Information (TI) [58], which are frequently-used metrics to measure the spatiotemporal complexity and a larger SI/TI means that the video has a higher spatial/temporal complexity. The metrics are calculated by the tool [12] provided by Video Quality Experts Group (VQEG) and the result is shown in Figure 24.

The result validates that *(i)* the spatiotemporal complexity of the videos we used covers a wide range: SI is ranging from 15 to 85 and TI is ranging from 3 to 25. *(ii)* Our test videos covers all the following types: high spatial complexity and high temporal complexity, high spatial complexity but low temporal complexity, low spatial complexity but high temporal complexity, and low spatial complexity and low temporal complexity.

C.5 Illustration example where GRACE performs poorly

In some rare cases, GRACE may suffer from poor quality. Figure 25 visualizes an example of four consecutive frames when GRACE performs poorly. As shown in the yellow box, the frame decoded by GRACE has some notable artifacts around the moving object, which degrades the SSIM.

C.6 Screenshot of videos we used for user study

Figure 26 shows the screenshot of the videos we used for the user study (in §5.3)

C.7 Working with other congestion control

GRACE can also work with the congestion control algorithm proposed in Salsify (Sal-CC) [45], which is more aggressive than GCC. Sal-CC has a higher average sending rate, while paying the cost of potentially having more packet losses. Figure 27 show that changing from GCC to Sal-CC increases the average SSIM of 0.7-1.1dB for GRACE with a negligible increase in video stall ratio. In contrast, the video stall ratio for Salsify codec will increase a lot when using Sal-CC, because Salsify codec needs to keep skipping frames for more than one RTT when packet loss happens, which leads to frequent video stalls.

C.8 Working with super resolution

In line with the discussion in §2.2, Super-Resolution (SR) can supplement the receiver-side video quality. We employed SwinIR [70], a leading SR model, in our simulation to confirm that GRACE, like baselines, can also leverage SR benefits. Our experiments demonstrated that SR boosts receiver-side quality for all codecs, irrespective of the specific codec employed. For more details, refer to Appendix C.8.

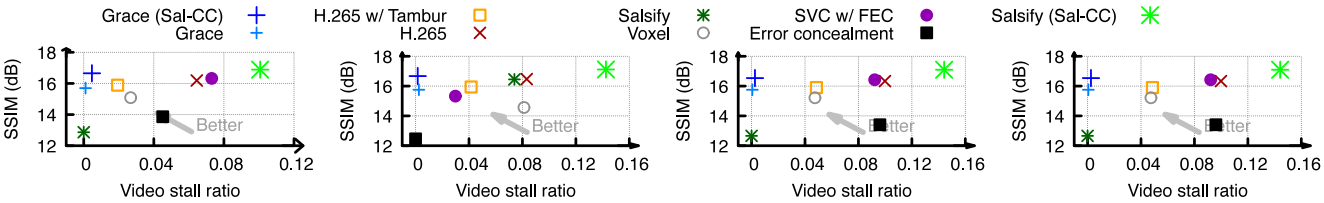
Figure 28 shows the tradeoff between quality and video stall ratio when using SR to enhance the quality at the receiver side. We run the simulation using LTE traces with a 100ms one-way delay and a 25-packet queue and then use a state-of-the-art SR model, SwinIR [70], to improve the quality of the decoded videos. When using SR, GRACE can still have on-par SSIM with Salsify codec and H.265 w/ Tambur: the

Figure 25: An example where GRACE performs poorly. It shows four consecutive decoded frames where the pink brush moves down quickly. Some artifacts in The yellow box degrade the frame quality and impact the SSIM.

(a) Sports: Football (b) Sports: Soccer (c) Gaming: Fortnite (d) Gaming: Genshin Impact

(e) Daily movement: Taekwondo (f) Daily movement: Baby in crib (g) Talking heads: indoor (h) Talking heads: outdoor

Figure 26: Summary of videos used in our user study. They span four categories: sports (a, b), gaming (c, d), daily movement (e, f), and talking heads (g, h).



(a) *One-way delay = 50ms* (b) *One-way delay = 75ms* (c) *One-way delay = 100ms* (d) *One-way delay = 150ms*

Figure 27: *End-to-end simulation result under different one-way delay. Network queue length = 25 packets*

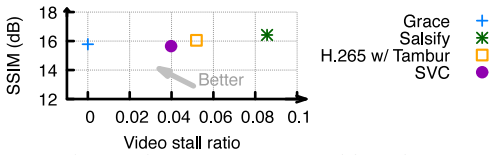


Figure 28: *The quality of* GRACE *and baselines after super-resolution*

	Encoding (ms)		Decoding (ms)	
	720p	480p	720p	480p
GRACE-Lite	35.1	17.2	40.9	21.6

Table 2: *Encoding/decoding time per frame for* GRACE*-Lite on Intel CPU*

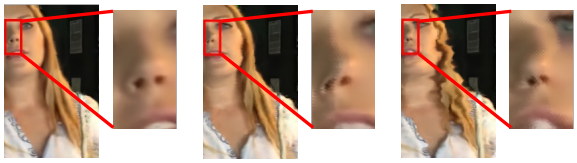
SSIMs are 15.8 dB, 16.4 dB, and 16.0 dB respectively. The SSIM of SVC (15.4 dB) is still lower than GRACE even with super-resolution. This is because packet loss can make higher layers of SVC undecodable, resulting in lower quality. This shows SR technique is complementary to our work, as it can improve the quality for any codecs at the receiver side.

C.9 Encoding/decoding time on CPU

We use OpenVINO library to run GRACE on a 32-core Intel(R) Xeon(R) Silver 4210R CPU. Table 2 shows the encoding/decoding time of a 720p/480p frame respectively. It can encode/decode a 720p frame at 28.5 fps and 24.4 fps respectively.

	SSIM (dB)	% of non rendered frames	Video stall ratio
GRACE	15.53	0.21	0.0011
GRACE-Lite	15.01	0.22	0.0012
GRACE-D	13.91	0.24	0.0014
GRACE-P	12.53	0.33	0.0023

Table 3: *End-to-end simulation shows* GRACE-*Lite has the same benefits in video realtimeness/smoothness compared to* GRACE *with marginal quality drop. Although* GRACE-*D and* GRACE-*P have similar video realtimeness/smoothness as* GRACE*, they suffer from low video quality.*



Grace **SSIM**: 12.0 dB | Grace-D **SSIM**: 10.7 dB | Grace-P **SSIM**: 8.8 dB

Figure 29: *Comparing reconstructed image when the same packet loss is applied to the pre-trained NVC (*GRACE-*P), a variant with only decoder fine-tuned with loss (*GRACE-*D), and* GRACE *(both encoder and decoder jointly fine-tuned).*

C.10 Simulation results and visualization examples for GRACE-Lite, GRACE-P and GRACE-D

Table 3 shows the end-to-end simulation results comparing GRACE, GRACE-Lite, GRACE-D, and GRACE-P. We use the LTE traces, and set the one-way-delay to 100 ms and the network queue length to 25 packets. GRACE-Lite has both similar quality and realtimeness/smoothness as GRACE. Without jointly training the encoder and decoder with loss, GRACE-P and GRACE-D fail to achieve similar quality as GRACE.

Figure 29 visualizes the reconstructed frame of GRACE, GRACE-P, and GRACE-D when the same 50% packet loss is applied to the encoded tensor of the same image, demonstrating that by jointly training both the encoder and decoder under various packet losses, GRACE delivers the best reconstruction quality without any prominent artifacts, and achieves a high SSIM.