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

Yihua Cheng, Ziyi Zhang, Hanchen Li, Anton Arapin, and Yue Zhang, The University of Chicago; Qizheng Zhang, Stanford University; Yuhan Liu, Kuntai Du, and Xu Zhang, The University of Chicago; Francis Y. Yan, Microsoft; Amrita Mazumdar, NVIDIA; Nick Feamster and Junchen Jiang, The University of Chicago

https://www.usenix.org/conference/nsdi24/presentation/cheng
GRACE: Loss-Resilient Real-Time Video through Neural Codecs

Yihua Cheng\textsuperscript{1}, Ziyi Zhang\textsuperscript{1}, Hanchen Li\textsuperscript{1}, Anton Arapin\textsuperscript{1}, Yue Zhang\textsuperscript{1}, Qizheng Zhang\textsuperscript{2}, Yuhan Liu\textsuperscript{1}, Kuntai Du\textsuperscript{1}, Xu Zhang\textsuperscript{1}, Francis Y. Yan\textsuperscript{3}, Amrita Mazumdar\textsuperscript{4}, Nick Feamster\textsuperscript{1}, Junchen Jiang\textsuperscript{1}
\textsuperscript{1}The University of Chicago, \textsuperscript{2}Stanford University, \textsuperscript{3}Microsoft, \textsuperscript{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\textsuperscript{1}. However, retransmitting lost packets across high-latency networks is not feasible due to stringent real-time latency requirements [57].

\textsuperscript{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].
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 de-packetizes 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):

2On-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.
Current estimation frame
Motion estimation & residual calculation
Video Encoder
Packetization
Rate adaptation
Congestion control

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 pertubations 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 undecodable.

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

$$\mathbb{E}_x [D(g_\theta(y), x) + \alpha \cdot S(f_\theta(x))] \quad \text{where} \quad y = f_\theta(x)$$

Here, $D(\hat{x}, x)$ is the pixel-level reconstruction error of the decoded frame $\hat{x}$ (by default, L2-norm\(^3\) of $\hat{x} - x$), and $S(y)$ is the entropy-coded data size of $y$ in bit-per-pixel (BPP). The parameter $\alpha$ governs the size-quality tradeoff: a higher $\alpha$ leads to a smaller frame size, $S(x)$, but higher distortion (i.e., poorer quality) of the reconstructed frame $\hat{x}$. As all the functions—$f_\theta$, $g_\theta$, $D$, and $S$ (approximated by a pre-trained NN [73])—are differentiable, the NN weights $\phi$ and $\theta$ 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, $f_\theta(x)$, as shown in Figure 4. The fraction of zeroed elements is dictated by a distribution, $P(y|f_\theta(x))$, which represents the probability distribution of the resulting tensor $y$ after random masking $f_\theta(x)$. For instance, with a 33% loss rate, $P(y|f_\theta(x))$ is the probability of $y$ arising from the random masking of 33% of elements in $f_\theta(x)$, as illustrated in Figure 4. Formally, GRACE jointly

\(^3\)Note that the L2-norm (or mean squared error) of $\hat{x}$ and $x$ is closely related to the PSNR of $\hat{x}$. To avoid this bias, our evaluation in §5 measures the quality improvement in SSIM and subjective user studies.
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 $j$th element to the $(i \cdot p - j) \mod n$th packet at the $\lceil i \cdot p - n \rceil$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. §4.1 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-
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.

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.

(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\%\} \footnote{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.}. 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 \(\alpha \left(2^{-7}\right)\) using Eq. 2. Subsequently, we perform fine-tuning with 11 \(\alpha\) 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. G\textsubscript{GCC} is on par with H.265/H.264 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\times, 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\times 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 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\times, 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\times 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).

**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-

<table>
<thead>
<tr>
<th>Dataset</th>
<th># of videos</th>
<th>Length (s)</th>
<th>Size</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kinetics</td>
<td>45</td>
<td>450</td>
<td>720p</td>
<td>360p</td>
</tr>
<tr>
<td>Gaming</td>
<td>5</td>
<td>100</td>
<td>720p</td>
<td></td>
</tr>
<tr>
<td>UVG</td>
<td>4</td>
<td>80</td>
<td>1080p</td>
<td></td>
</tr>
<tr>
<td>FVC</td>
<td>7</td>
<td>140</td>
<td>1080p</td>
<td></td>
</tr>
</tbody>
</table>

Total 61 770

Table 1: Dataset description.
we empirically choose the 64×64 block size [64, 74, 99], while a larger block size such as 256×256 hinders 
NN-based baselines and does not provide a public codebase for testing.

for a particular video type (talking head), but it lacks comparisons with any 
design choices, we evaluate

by the congestion control algorithm, i.e., no overshoots or 
buffering and improve the user’s QoE.

employs selective frame skipping to mitigate video re-
fected 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]

This idea makes these baselines perform
slightly better than they would under real-world conditions.

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.

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 −10log(1−SSIM) across all rendered frames.

Realtimeiness 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

Figure 8: Video quality achieved by different schemes under varying packet loss rates at the same encoded bitrate (6 Mbps).
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 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...
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, 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
Mean opinion score (MOS) for Grace, Tambur, WebRTC, and Salsify.

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.

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 decoding, 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.

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


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 bytestream 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 $P(y)$, we can apply the differentiation property of logarithms to get

$$\nabla_\theta P(y) = P(y) \log P(y)$$

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

$$\nabla_\theta \mathbb{E}_{y \sim P(y)}([D(g_\theta(y),x)]) = \mathbb{E}_{y \sim P(y)}([D(g_\theta(y),x)] \nabla_\theta \log P(y)) \tag{3}$$

which can be estimated using Monte-Carlo sampling $\approx \frac{1}{N} \sum_{i=1}^{N} D(g_\theta(y_i),x) \nabla_\theta \log P(y_i)$. 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 $D(g_\theta(y_i),x)$, where $P(y_i) = 1$.

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)
To summarize, the encoder side’s computational overhead 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 re-decodes the motion vector and residuals of every 10 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
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.
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

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.

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).

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

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

C.9 Encoding/decoding time on 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.

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.
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.

<table>
<thead>
<tr>
<th></th>
<th>SSIM (dB)</th>
<th>% of non rendered frames</th>
<th>Video stall ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>GRACE</td>
<td>15.53</td>
<td>0.21</td>
<td>0.0011</td>
</tr>
<tr>
<td>GRACE-Lite</td>
<td>15.01</td>
<td>0.22</td>
<td>0.0012</td>
</tr>
<tr>
<td>GRACE-D</td>
<td>13.91</td>
<td>0.24</td>
<td>0.0014</td>
</tr>
<tr>
<td>GRACE-P</td>
<td>12.53</td>
<td>0.33</td>
<td>0.0023</td>
</tr>
</tbody>
</table>

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.