Prism: Handling Packet Loss for Ultra-low Latency Video.

Devdeep Ray
devdeepr@cs.cmu.edu
Carnegie Mellon University
Pittsburgh, PA, USA

Vicente Bobadilla Riquelme
riquelmev@carleton.edu
Carleton University
Northfield, MN, USA

Srinivasan Seshan
srini@cs.cmu.edu
Carnegie Mellon University
Pittsburgh, PA, USA

ABSTRACT

Real-time interactive video streaming applications like cloud-based video games, AR, and VR require high quality video streams and extremely low end-to-end interaction delays. These requirements cause the QoE to be extremely sensitive to packet losses. Due to the inter-dependency between compressed frames, packet losses stall the video decode pipeline until the lost packets are retransmitted (resulting in stutters and higher delays), or the decoder state is reset using IDR-frames (lower video quality for given bandwidth).

Prism is a hybrid predictive-reactive packet loss recovery scheme that uses a split-stream video coding technique to meet the needs of ultra-low latency video streaming applications. Prism's approach enables aggressive loss prediction, rapid loss recovery, and high video quality post-recovery, with zero overhead during normal operation - avoiding the pitfalls of existing approaches. Our evaluation on real video game footage shows that Prism reduces the penalty of using I-frames for recovery by 81%, while achieving 30% lower delay than pure retransmission-based recovery.

CCS CONCEPTS

• **Information systems** → **Multimedia streaming**.

KEYWORDS

Cloud Gaming, Loss Prediction, Loss Recovery, Video Streaming

ACM Reference Format:
Devdeep Ray, Vicente Bobadilla Riquelme, and Srinivasan Seshan. 2022. Prism: Handling Packet Loss for Ultra-low Latency Video.. In *Proceedings of the 30th ACM International Conference on Multimedia (MM '22), Oct. 10–14, 2022, Lisboa, Portugal.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3503161.3547856

1 INTRODUCTION

Steady improvements in networking algorithms, hardware, and video coding techniques have enabled the Internet to support a wide range of video applications (e.g. live conferencing, large-scale broadcasts, video-on-demand). Recently, a new class of video streaming applications has emerged: *ultra-low latency* interactive video. These applications offload compute and rendering to the cloud, and stream the view-port to the end user. Examples of these applications include virtual game consoles (Stadia [13], GeForce NOW [23], XCloud [35]), cloud AR [29], and virtual desktop (Chrome Remote Desktop [12], Azure Virtual Desktop [21]). These applications push the limits of existing streaming techniques and network infrastructure, and have extremely demanding QoE requirements (video stream quality and end-to-end frame delay).

The Internet's design fundamentally follows a best-effort philosophy - there are no guarantees for reliable delivery of packets. As a result, applications must deal with issues such as packet loss [6], which is especially harmful for low-latency video streaming applications due to the interdependence between frames in compressed video (e.g. P- and B-frames). Loss of video data (as a consequence of packet loss) stalls the video decoder pipeline, since past frames need to be decoded in order to decode a P-frame.

Packet loss mitigation strategies fall into four broad categories:

(1) **Preventing data loss:** FEC [19, 26] transmits redundant packets to avoid data loss even if some packets are lost.
(2) **Concealing lost data:** Partial P-slice decoding [5, 8], error concealment [36, 37], joint source-channel coding [16], and scalable coding [11, 28] (SVC) are able to continue decoding the video at a reduced quality when packet losses occur.
(3) **Recovering lost data:** Packet or frame retransmissions [3] can be used to recover lost data, and has the benefit zero bandwidth or performance overhead when no losses occur.
(4) **Recovering from data loss:** IDR-frames (independent data refresh) and slice-based intra-refresh [34] result in lower delays compared to (3) by eliminating the need for the video decoding pipeline to catch up to the current frame.

Techniques that prevent or conceal data loss (FEC, partial decoding, error concealment, scalable coding) enable packet loss mitigation without requiring receiver feedback, and thus, do not incur a round-trip penalty for loss recovery. On the other hand, these techniques break down when packet losses exceed a certain threshold, resulting in non-recoverable video data loss. Non-recoverable video data loss when using FEC has the same implications as a packet loss when not using FEC. When techniques like loss concealment are used, packet losses manifest as undesirable artifacts that significantly degrade the QoE for immersive applications like cloud-based gaming, augmented reality (AR), and virtual reality (VR).

There are two techniques used for recovering from video data loss: packet retransmissions, and IDR frames (Independent Data Refresh, see § 2 for background). These solutions force applications to make difficult tradeoffs between accepting significantly degraded picture quality, or significantly higher frame delay when affected by packet loss [8, 14]. For instance, during bursty loss events, transmitting IDR frames reduces frame delay since they can be independently decoded. On the other hand, since IDR frames do not leverage temporal redundancies in video data, they require more bits per frame compared to P-frames for equivalent picture quality. In the case of retransmission based recovery, high picture quality is maintained since the frames are still encoded as P-frames.

This material is based upon work supported by NSF Grant No. CNS-1956095.

MM '22, October 10–14, 2022, Lisboa, Portugal

ACM ISBN 978-1-4503-9203-7/22/10.
https://doi.org/10.1145/3503161.3547856

Unfortunately, since the receiver accumulates a backlog of undecoded frames until the lost data has been recovered, the end-to-end frame delay remains high until the decoder catches up.

In order to resume decoding after non-recoverable video data loss, the sender must first be notified, after which the sender can retransmit packets or send IDR frames to recover from the data loss event. This process incurs a minimum delay of 1 round-trip time (RTT) before the sender can trigger loss recovery. In order to reduce this delay, one could use speculative loss prediction mechanisms [10, 27], but this approach has the risk of false positives, which can affect the video quality significantly (eg. when using IDR frames).

Our system, called Prism, incorporates a novel frame transport protocol, optimizations at the video codec level, and deep-learning based packet loss prediction to significantly improve the QoE by achieving smoother video playback without significantly sacrificing video quality. Prism leverages the insights that (1) IDR-frames enable rapid recovery of a video stream after a loss event, since they are immediately decodable and reset the decoder state, and (2) A stream of P-frames can sustain high quality for some time even after a reduction in bitrate, and thus enable rapid recovery post-recovery. When Prism identifies a potential loss event, it splits the video stream into two substreams - a low-latency, unreliable IDR-frame stream, and a high quality, reliable P-frame stream. The IDR-frames enable the application to continue displaying frames with low delay, albeit at a lower quality. When the lost P-frame data is retransmitted and the pending frames are decoded, the application can quickly switch back to the higher quality P-frame stream. Prism carefully allocates the total available bandwidth between the two streams by using results from a novel video analysis and optimization pipeline that runs offline, and thus avoids real-time computational overhead.

Prism's approach reduces the impact of falsely triggering loss recovery, enabling the use of aggressive loss prediction for speculative loss recovery. This allows Prism to take action before potential losses occur, reducing the time to recovery. When Prism's recovery mechanism is falsely triggered, the receiver can simply decode and display the high quality P-frame stream. Prism eliminates the hard tradeoffs present in traditional approaches and provides a more flexible trade-off between frame delay and video quality.

Our key design contributions include:

(1) **Split Stream Video Coding:** During packet loss, Prism splits the available bandwidth across a low latency IDR-frame stream and a high quality P-frame stream, ensuring low frame delay during loss events, and high video quality post recovery. In addition, the impact of falsely triggering Prism's loss recovery is low; this enables Prism to use loss prediction mechanisms to respond early to potential packet loss.

(2) **Video Analysis Pipeline:** Prism analyzes video scenes offline to optimize the bandwidth allocation across the two video streams. Prism's novel black box encoder modeling technique provides fast approximations of the video quality for given bitrate configurations, speeding up the optimization pipeline by multiple orders of magnitude.

(3) **Deep-learning Based Loss Prediction:** Prism's evaluation uses a simple deep-learning based packet loss predictor in order to demonstrate the benefits of Prism when used in conjunction with aggressive loss prediction. Loss prediction allows Prism to react to losses earlier, mitigating the network round-trip penalty for packet loss recovery.

We evaluated a real-time implementation of Prism across a diverse set of video game footage and emulated network conditions. Prism reduces the quality penalty of using IDR-frames for loss recovery, while preserving the low delay benefits of using IDR-frames. Simulation of Prism's algorithm using discrete event simulation techniques on real world network traces from M-Labs [20] shows that Prism can outperform IDR-frame based recovery *and* P-frame retransmissions by handling packet loss better in the presence of significant delay and bandwidth variation, reducing the quality penalty of I-frames by 81 % on average. Additionally, aggressive loss prediction mechanisms that reduce delay by proactively triggering loss recovery work in conjunction with Prism's split-stream approach, achieving much higher video quality in comparison to IDR-frame based loss recovery with loss prediction.

2 VIDEO COMPRESSION BACKGROUND

Video frames are comprised of slices, where each slice can be one of three types: (1) Intra (or I), (2) Predicted (or P), and (3) Bidirectional (or B). An I-slice is encoded and decoded independently, whereas P-slices depend on past frames and B-slices may depend on both, past and future frames. P- and B-slices leverage inter-frame redundancy by using motion compensation [18] to approximate the slice by using localized motion vector references to past and future frames, significantly reducing the bitrate requirement to achieve the desired picture quality (e.g. Structural Similarity [32] (SSIM), which ranges from 0 (worst quality) to 1 (identical to source)) [1].

Low latency streaming applications rely heavily on the use of IDR-frames (independent data refresh) and P-frames. These applications encode video as P-frames during normal operation due to their compression efficiency - thus, a single missing packet stalls prevents the decoding of subsequent frames. IDR-frames are special I-frames (contains only I-slices, I- and IDR- used interchangeably henceforth) that are commonly used to recover from video data loss, since they reset the decoder state, which can then discard previous undecoded frames and decode the most recent frames with minimal delay.

Prism leverages three key properties of video compression - (1) The quality of an I-frame depends only on the bitrate and the content of a particular frame, (2) I- and P-frames demonstrate diminishing returns with respect to the video quality as the bitrate increases, and (3) In addition to motion, frame content, and the bitrate, the quality of a P-frame also depends on the quality of previous frames since they are approximated from previous frames using motion vectors.

3 LOSS DETECTION AND RECOVERY

The goal of this paper is to design a video data loss recovery mechanism that *simultaneously* achieves (1) low delay and smooth video playback, (2) high picture quality, (3) minimal quality impact under false packet loss triggers, enabling aggressive loss prediction without significant penalties, and (4) zero overhead under normal operation. While loss prevention techniques (e.g. error coding, FEC) can reduce the frequency of video data loss with some bandwidth

[1] Raw SSIM lies in [-1,1], many tools scale it to be in [0,1].

and compute overhead, they provide no guarantees. Transmitting IDR-frames on video data loss achieves (1) and (4), and retransmission of lost P-frame packets achieves (2), (3) and (4). In the following subsections, we discuss: (1) the impact of loss detection mechanisms, (2) the two reactive recovery mechanisms, and (3) an example in Section 3.3 to show how Prism's split stream approach can achieve all of the above properties simultaneously.

3.1 Loss Detection

The sender getting notified of video data loss is the first step in loss recovery, which then triggers the loss recovery mechanism. Since packet losses are often accompanied by periods where no ACKS are received, or an increase in delay (e.g. due to cross traffic running loss-based TCP), speculatively triggering loss recovery using aggressive loss prediction (e.g. sub-RTT timeouts, loss prediction using machine learning) can reduce video stuttering and frame delays [1, 15]. On the other hand, noisy networks, queue-building cross traffic (e.g. BBR [4]), packet reordering and ACK loss [17] may result in frequent false triggers. Excessive false triggering of recovery mechanisms (like IDR-frames) harms video quality (§ 2). Delaying recovery until packet loss can be verified results in significant video stutter, which is unacceptable for ultra-low latency video streaming applications. Prism drastically reduces the impact of false loss recovery triggers on video quality, and thus enables the use of aggressive loss prediction mechanisms in order to reduce video stutter under packet loss.

3.2 Reactive Loss Recovery

Packet Retransmissions: A video data loss in a P-frame stream can be recovered using packet retransmissions. This approach achieves high picture quality by leveraging the compression efficiency of P-frames due to motion compensation, but results in higher end-to-end frame delays and video stutter. Consider a video stream being sent over a link with an RTT of 60 ms. When a loss occurs, the sender is only notified 60 ms after having sent the original data. During this period, the sender keeps encoding frames as P-frames, and these frames get queued at the decoder without being displayed, until the lost packets are retransmitted. On receipt of the retransmissions, the decoder "catches up" to the latest frame, which can take a long time (based on decode performance). For example, for a 60FPS video where each frame takes 10 ms to decode, catching up with 60 ms of backlogged frames can take up to 100 ms [2]. When streaming in 4K to low power client devices like mobile phones and TV streaming sticks, the decode time can be much closer to the frame time, significantly increasing the recovery delay.

I-frames: When packet loss occurs, transmitting the latest frames as I-frames (IDR) resets the decoder state, and thus, the latest frames can be decoded and displayed immediately. Using I-frames in conjunction with aggressive loss detection can significantly reduce video stutter and improve video smoothness. Unfortunately, (1) IDR frames have much lower video quality compared to P-frames encoded at the same bitrate, and (2) the low quality of an IDR frame also affects the video quality of subsequent P-frames, since their quality depends on the video quality of past frames.

[2] Let x be the delay after the loss until P-frames recover. Frames decoded: $\frac{x}{10}$, frames sent: $\frac{60+x}{16}$. Solving, we get x=100

3.3 Proposed Hybrid Approach

The two loss recovery mechanisms discussed above have their own benefits and downsides: P-frame retransmissions achieve higher picture quality, but incur higher delays and stutter. On the other hand, IDR-frames improve video smoothness and reduce latency at the cost of video quality. In Prism, we propose a hybrid architecture that combines the two mechanisms to achieve a better trade-off between latency and video quality by leveraging two key properties (§ 2): (1) The diminishing returns of video quality with higher bitrate, and (2) the dependence of the quality of a P-frame on the quality of past frames. When packet loss occurs, the quality of a P-frame stream can be preserved for a short duration at a reduced bitrate, while the residual bandwidth can be used for transmitting I-frames in order to maintain low delay until the P-frame stream recovers using packet retransmissions.

In Figure 1b, we encode a video segment for "GTA-V" using different encoding configurations. The yellow line is the steady state P-frame SSIM at 20 Mbps (best case - no packet loss). The line "I-frame insertion" shows the instantaneous drop in picture quality, and the quality impact on subsequent P-frames when I-frames at 10 Mbps are used during loss recovery (between frames 5 and 10). During loss recovery, Prism splits the available bandwidth in order to continue the P-frame stream (at a lower bitrate), and additionally transmits low delay I-frames. For example, say we allocate 2 Mbps for the P-frame stream and 8 Mbps for the I-frames during loss. The I-frames reduce latency during the ongoing loss event, and the receiver switches back to the P-frame stream after frame 10. The key insight here is that when the bitrate is reduced, the quality of the P-frame stream is preserved for a short duration. The line marked as "Real Loss" denotes Prism's video quality when real loss occurs - while the quality during loss is slightly lower compared to I-frames (at 10 Mbps), the video quality post-recovery is much higher. The line marked as "Spurious Loss" denotes Prism's video quality when loss recovery is falsely triggered (i.e. the quality of the P-frame stream), and thus, Prism's approach works well with aggressive loss-prediction techniques as opposed to I-frame insertion.

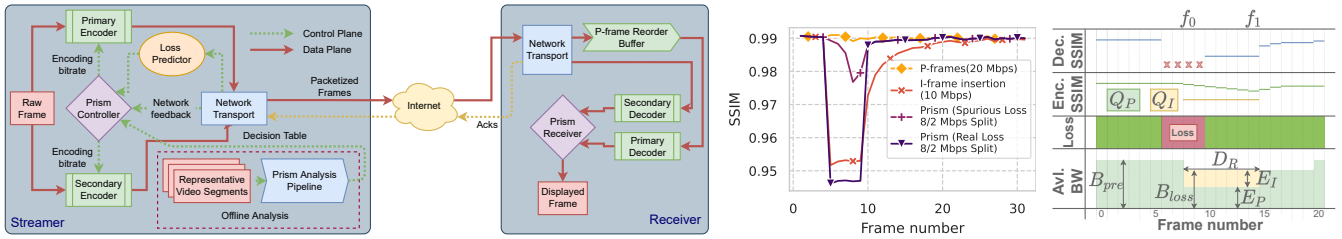
4 DESIGN

In this section, we discuss Prism's system design, and how it robustly mitigates video data loss for interactive video streaming applications like cloud gaming, and AR/VR streaming.

4.1 Overall Architecture

Prism's streamer and receiver architectures are shown in Figure 1a. The network transport layers at the streamer and the receiver are responsible for packetization and multiplexing of the two video streams over a single connection. In order to trigger loss recovery, the Prism controller combines signals from a deep-learning based loss predictor and loss signals from the transport layer. The controller also determines the allocation of the available bandwidth across the two video streams.

When no packet loss occurs, frames are encoded as P-frames using all of the available bandwidth. The receiver reassembles the packets, decodes and displays the frame. When the transport layer or the loss predictor indicates packet loss, each frame is encoded as a P-frame (by the primary encoder) and an IDR-frame (by the



(a) Prism streamer and receiver architecture.

(b) Impact of I-frames and bitrate changes on SSIM.

(c) Timeline of Prism when a packet loss occurs.

Figure 1: Prism architecture.

secondary encoder), and the Prism controller allocates the total bandwidth across the two streams. The P-frame is transmitted reliably using packet retransmissions, and the IDR-frames are transmitted as one shot (no retransmissions). The P-frames after a packet loss are stored in a buffer at the receiver while the transport layer retransmits the lost packets. Meanwhile, the IDR-frames that make it through are immediately decoded and displayed, maintaining low video latency while the P-frame stream recovers from the loss.

Prism needs to optimize the bandwidth allocation in order to maximize the benefit of this approach, and ensure that the quality is better than simply using I-frames. Prism's offline analysis pipeline uses prerecorded training video sequences to compute compact bandwidth allocation tables using a greedy heuristic in order to make bandwidth allocation decisions in real-time (§ 4.2). Prism uses a novel black box encoder modeling technique that enables fast and accurate estimation of the video quality for arbitrary video encoding schedules (bitrate, IDR-frame insertions) without actually encoding the video, which reduces the amount of computation required for generating the bandwidth allocation tables by multiple orders of magnitude (§ 4.3).

4.2 Optimizing Bandwidth Allocation

Consider the timeline shown in Figure 1c, with the X-axis denoting the frame number. In the example, no packet losses occur until frame 5, and frames 6 through 9 are affected by packet loss (shown in the loss timeline). The video bitrate of the P-frame stream before loss recovery is triggered is B_{pre} Mbps, and the available bandwidth during the period of loss recovery is B_{loss} (B_{pre} and B_{loss} are determined by the congestion control and rate control algorithms, assumed to be constant here for simplicity). During the recovery period, let the encoding bitrate of the I-frames and the P-frames be $E_I(f)$ and $E_P(f)$ respectively, where f is the frame number. This gives us the constraint

$$E_I(f) + E_P(f) = B_{loss} \; \forall \; f_0 \leq f \leq f_1 \tag{1}$$

Since the duration of loss is not known beforehand, we propose a greedy optimization strategy: when transmitting a frame during loss recovery, Prism splits the bandwidth assuming that it is the last frame in the loss recovery state. Thus, given the bandwidth split until frame f_i, Prism needs to determine the allocation for frame f_{i+1} that balances three things: (1) The video quality during loss (the quality of the I-frames), (2) the video quality after the loss event (quality of the last P-frame sent during loss recovery), and (3) the video quality when the loss is spurious (quality of the P-frames during loss recovery).

We first derive the optimization objective for a known video segment, where the loss recovery begins at f_0 and ends at f_1. The mappings between video bitrate and video quality are defined as:

- $Q_I(x, f)$: The quality of frame f when encoded at a bitrate x as an I-frame.
- $Q_P([x_1 ... x_f], f)$: The quality of frame f when frames $1...f$ are encoded as P-frames, and the bitrate of frame i is x_i.

For a frame f transmitted during loss recovery, we define two objective functions, O_1 and O_2, denoting the video quality during real loss, and when loss recovery was falsely triggered respectively. Under real loss, the video quality is determined by the quality of the I-frames (Q_I in Figure 1c) and the quality of the P-frames after recovery (Q_P after f_1 in Figure 1c). If the I-frame allocation is x,

$$O_1(x) = Q_I(x, f) + \sum_{i=f+1}^{f+K} Q_P([x_{f_0} ... x_{f-1}, B_{loss} - x, B_{loss} ... B_{loss}], i) \tag{2}$$

where K is a fixed horizon of future frames that are assumed to be encoded at B_{loss} to account for the quality convergence of the P-frames, and $x_{f_0} ... x_{f-1}$ are fixed according to the allocations for past frames during the current loss event (greedy approximation).

When spurious loss occurs, the video quality is determined solely by the quality of the P-frame stream. Thus,

$$O_2(x) = \sum_{i=f}^{f+K+1} Q_P([x_{f_0} ... x_{f-1}, B_{loss} - x, B_{loss} ... B_{loss}], i) \tag{3}$$

This allows us to define objective function for determining the bandwidth split for a frame f:

$$x_f = \arg\max_x w \cdot O_1 + (1 - w) \cdot O_2 \tag{4}$$

Here, w is the weight for real loss, and $1 - w$ is the weight for false loss recovery triggers. w is set based on the accuracy of the loss prediction mechanism (eg. a smaller value for w is better if false loss recovery triggers are frequent).

In order to find the best split, it is sufficient to sweep different values of x, and pick the value that maximizes the objective function. Unfortunately, this optimization cannot be solved in real-time since objective function requires the quality of future frames, which are not available for real-time applications. Prism's key insight is that while video properties can vary across individual frames, the

result of the optimization is "stable" for a given video style. In 4.3, we describe an offline approach to compute bandwidth allocation tables for a particular video style using prerecorded training segments, enabling Prism to run in real-time without any runtime computational overhead.

4.3 Offline Analysis Pipeline

Solving the optimization problem in real-time when a loss event occurs is impractical not only from a delay and overhead perspective, but also not possible since the objective function includes the quality of future frames. To address this issue, Prism analyzes pre-recorded video segments that are representative of particular scene types, and runs the optimization for a wide range of bandwidth values and loss durations, for various starting points in the pre-recorded video. The results are aggregated across the starting points in the video into a decision table that maps $(B_{pre}, B_{loss}) \rightarrow [x_f ... x_{f+D}]$, where D is a maximum limit on loss recovery duration before Prism falls back onto traditional techniques. Prism uses the appropriate decision tables for the particular video style in order to determine the bandwidth split in real-time during loss recovery.

Consider the optimization run for a particular section of a pre-recorded video segment. If we limit the maximum loss duration to 10 frames, and limit B_{pre}, B_{loss}, and x_f to $1 - 20$ Mbps in steps of 1 Mbps, the brute force approach requires $\approx 20 \times 20 \times 10 \times 10$, ie. 40000 different encoding schedules to be evaluated. In addition, for each evaluation, we must consider around 1 second of video before and after loss recovery for allowing the quality of the P-frames to converge. Thus, analyzing a single location in a training video requires an unacceptable 80000 seconds of video encoding (22 hours). This scales up linearly as we sample more locations in the training video in order to generalize the decision table for a particular video style.

4.3.1 ***Black box encoder modeling.*** In Prism, we propose a unique approach to reduce the complexity of the offline analysis step, speeding it up by multiple orders of magnitude. Prism analyzes data from a small set of carefully designed video encoding runs, which enables Prism to accurately predict video quality for the given video sample when (1) a frame is encoded as an I-frame, (2) the encoding bitrate of a P-frame stream is changed, or (3) an I-frame is inserted in a P-frame stream. This enables Prism to avoid encoding the actual video to compute the objective function when performing a brute force sweep of the search space for determining the optimal bandwidth splits. To our knowledge, this method is unique to Prism and has not been published in past work.

Suppose we are given a long sample video of duration N seconds. First, we encode the video only using I-frames for all bitrates ranging from $1 - 20$ Mbps ($20 \times N$ seconds of video encoded), enabling us to compute $Q_I(x, f)$. Second, we sample K segments (each segment starts at frame f, duration of 1 second) from the video, and encode each segment in the following manner for each bitrate E value between 1 Mbps and 20 Mbps:

- Encode the first frame at a very low bitrate, and the subsequent frames at the chosen bitrate as P-frames ($20 \times K$ seconds of video encoded). Let's denote this family of functions as $T_{(f,E)}$, shown as the yellow flow lines in Figure 2a.
- Encode the first frame at a very high bitrate, and the subsequent frames at the chosen bitrate as P-frames ($20 \times K$ seconds of video encoded). Let's denote this family of functions as $T'_{(f,E)}$, shown as the red flow lines in Figure 2a.

$T_{(f,E)}$ and $T'_{(f,E)}$ can be combined and interpolated as a vector field that denotes the quality convergence of P-frames for a given bitrate E, enabling us to compute the video quality of a P-frame encoded at E when the video quality of the previous frame is known.

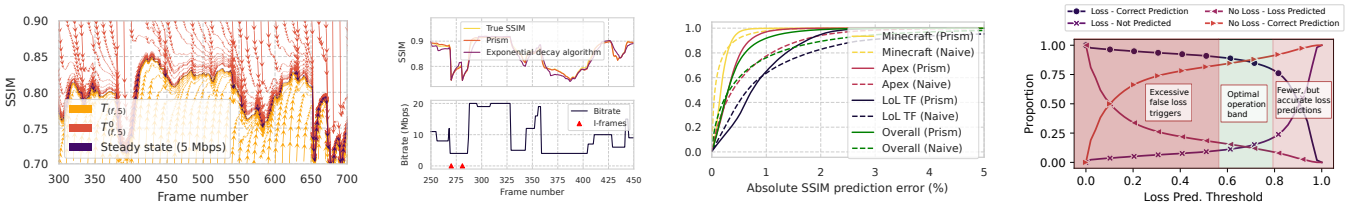
If we sample $K = 100$ segments from the original video of duration $N = 20$s, the total duration of video encoding required is 400s for I-frame qualities, and 4000s for the P-frame transition properties. This enables us to compute the SSIM of the 40000 encoding schedules for each starting point in the video discussed in § 4.3 without requiring any additional encoding. If the optimization aggregates across 100 different starting points in the video, this technique provides more than a three orders of magnitude speed up.

To measure the accuracy of our algorithm, we generated video encoding schedules with random changes in the video bitrate and with random I-frame insertions. Each video is encoded with 100 different random schedules. A zoomed in view of one such random schedule is shown in Figure 2b. The lower subplot shows the bitrate schedule and the I-frame insertions, while the true SSIM and the predicted SSIM are shown in the top subplot . We also compared the results of our algorithm discussed in Section 4.3 to a simpler algorithm where the SSIM of the video converges to the steady state P-frame SSIM using exponential decay (we choose the decay parameter that minimizes the total absolute error for each encoding schedule). The prediction error CDFs for three videos (best, worst and one in-between) are shown in Figure 2c, along with the overall prediction error CDF across all 13 videos (§ 5). Our algorithm for modeling the video SSIM is very accurate, with 90% of the predictions being within 1% of the true SSIM. The naïve approach using exponential decay fails to capture the SSIM variations across frames during transition periods.

4.4 Loss Prediction

Prism's design enables the use of aggressive loss prediction which in turn reduces video frame delay, since the decoder can utilize the higher quality P-frame stream if the loss prediction was a false trigger. We designed a simple convolutional neural network that uses network statistics such as bytes sent, mean, variance and linear fit for RTT, and the bytes lost for 200 ms windows to predict the occurrence of loss in the subsequent 50 ms. The neural network comprises of three convolutional layers and three dense layers, and the simplicity results in low computational overhead - predicting loss for a 50 ms window takes less than 1 ms on an Intel Core i7 CPU. We trained this model on 9 days of M-Labs NDT data [20], and tested the packet loss prediction performance for a tenth day. Additional details are provided in appendix F.

The neural network outputs a probability value that denotes its confidence in loss occurring. This design also enables Prism to choose a probability threshold for triggering loss recovery, which determines the trade-off between loss prediction accuracy and false triggering of loss recovery. This trade-off between accuracy and false loss recovery triggers is shown in Figure 2d. While a low threshold for loss prediction is able to identify many more packet



(a) $T_{(f,5)}$, $T'_{(f,5)}$ **for various starting points (**f**) (Tekken)**

(b) SSIM prediction for an encoding schedule.

(c) SSIM prediction error - Prism vs. naïve exponential decay.

(d) Prism neural-net loss prediction accuracy trade-off

Figure 2

loss events, thus enabling faster loss detection and recovery for lower delay and smoother video, it also results in a significant number of false packet loss triggers - Prism is able to adapt its bandwidth allocation to the accuracy of loss prediction since it includes a tuning parameter w (§ 4.2) that balances the quality between real losses and false loss triggers.

5 EVALUATION

Prism's evaluation has two broad themes: evaluation of the optimization framework, and end-to-end evaluation of the system.

Optimization Framework. Prism uses pre-computed bandwidth allocation tables for different video types, since optimizing the video quality in real-time is not feasible (§ 4.2). We evaluate Prism's optimization strategy using the CGVDS [2] dataset, which contains a diverse set of 13 video game captures that have very different video compression properties (screenshots in appendix A). This section of the evaluation provides a better understanding of Prism's efficacy in achieving better quality compared to recovery using I-frames. In § 5.1.1, we evaluate the impact of overall bandwidth availability on Prism's performance. In § 5.1.2, we quantify the impact of using a single decision table for an entire scene, as opposed to the "ideal" bandwidth split for a particular location in a given video. In § 5.1.3, we quantify the impact of loss duration on video quality.

End-to-End Evaluation. In the second part, we evaluate Prism's performance using two different approaches: (1) we evaluate Prism's implementation that runs in real-time and uses the generated decision tables for allocating bandwidth across the two streams, and (2) we evaluate Prism using a discrete event simulator that uses real-world network traces from M-Labs [20] and Prism's neural-network based loss prediction.

We compare Prism's end-to-end performance to the two key video data loss recovery techniques: IDR-frames, and packet retransmissions (§ 3.2). For comparing video quality, we use SQI-SSIM [7], a metric based on SSIM that imposes an exponential decay penalty when video frames are delayed (accounting for the length of video stalls), and penalizes the quality for some time after each stall event (accounting for frequency of stalls). We also compare the end-to-end frame delay (including transmission delay, propagation delay and decoding delay) for frames that end up getting displayed on the screen. SQI-SSIM and the frame delay together provide a holistic view of the QoE for low-latency streaming applications.

5.1 Optimization Framework

In our plots, we compare structural dissimilarity (DSSIM), defined as $\frac{1-rawSSIM}{2}$, which quantifies the amount of distortion from the reference image. For consistency, we use decision tables generated using $w = 0.5$ throughout (equal weights for real and spurious loss).

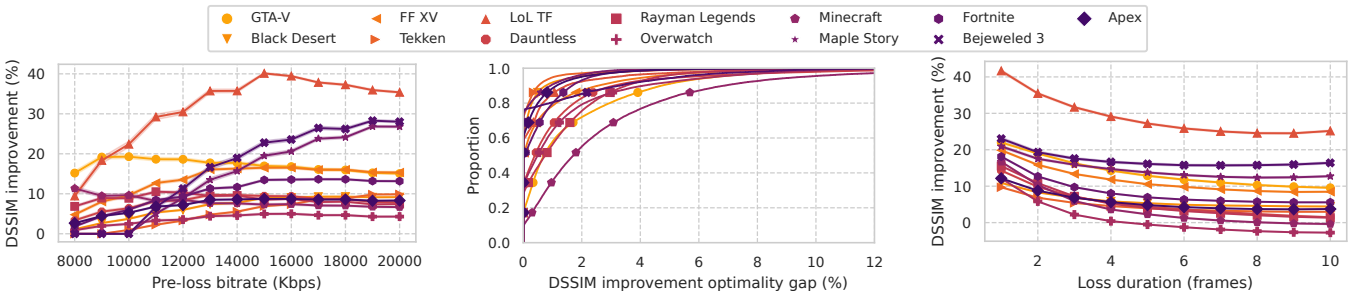
5.1.1 ***Bandwidth Sensitivity.*** Video quality gains achieved by Prism depend on the overall bandwidth availability. Transmitting two separate streams is inefficient when the available bandwidth is low, since a large portion of this bandwidth would be taken up by redundant data shared across the frames. On the other hand, when a lot of bandwidth is available, the impact of I-frames on video quality is insignificant. Figure 3a shows the improvement (reduction) in DSSIM vs. using I-frames, for different values of pre-loss bandwidth (shown on the X-axis). The data points on the Y-axis aggregate the improvements across all the values of loss bandwidth that are greater than half of the pre-loss bandwidth.

Prism's gains are higher for videos like "LoL TF", where P-frames are particularly efficient at encoding the video data. On the other hand, games like "Minecraft" and "Tekken" have smaller gains since intricate textures and complex motion reduce the efficiency of P-frames (see appendix A). Further, Prism's gains are lower at very high bitrates, since I-frames have sufficient video quality.

5.1.2 ***Scene Stability.*** Prism's optimization pipeline generates a single decision table for an entire scene by aggregating the objective function across multiple segments extracted from the sample scene. The underlying assumption behind this approximation is that while the SSIM of individual frames can vary, the transition properties and the relative quality between I-frames and P-frames are stable across a particular scene type. To verify this assumption, we compute the CDF of the percentage difference in the objective function when using the decision table, versus the optimal bandwidth split that is optimized for a specific video section. We generate data points using various pre-loss and loss bitrates, and multiple loss durations.

The results (shown in Figure 3b) show that the performance when using a single decision table is almost as good as using the optimal split for a specific segment. This approximation enables Prism to make bandwidth allocation decisions in real-time by simply performing a table lookup. In appendix B, we include additional analysis that shows the impact of using incorrect bandwidth allocations on the video quality, demonstrating that optimizing for each video type is necessary.

5.1.3 ***Loss duration sensitivity.*** Prism's benefits are realized due to two fundamental video coding properties: P-frames are more



(a) Impact of network bandwidth on Prism's performance.

(b) DSSIM gap when optimizing for an entire scene vs. a specific point in the video.

(c) Impact of loss duration on DSSIM improvement.

Figure 3: Performance of Prism's bandwidth split optimization algorithm.

efficient at encoding video data, and the quality of a P-frame stream exhibits a gradual transition when the bitrate is changed. The benefits of Prism can be realized even for very long periods of packet loss if the P-frames are extremely efficient (eg. "LoL TF"). When P-frames are not as efficient, Prism still demonstrates quality gains for shorter loss durations (e.g. "Overwatch"). This is shown in Figure 3c, which plots the improvement in video quality over simply using I-frames, as a function of the loss duration. Prism's optimization algorithm accounts for scenarios like "Overwatch", switching to sending I-frames only after running in split-stream mode for a few frames. For extended loss durations, this approach is only slightly worse than transmitting I-frames right from the beginning, and stems from Prism's greedy approach for allocating bandwidth across the two streams.

These results also suggest that game developers designing games for cloud platforms can improve streaming performance by making encoder-friendly choices for artistic styles and gameplay mechanics.

5.2 End-to-end Evaluation

We evaluated the end-to-end performance of Prism in two different ways: (1) A real-time implementation that streams video over emulated lossy links (§ 5.2.1), and (2) a discrete event simulator that evaluates Prism on real-world M-Labs [20] traces, integrating Prism's neural network based loss prediction mechanism (§ 5.2.2).

5.2.1 ***Real-time Streaming Experiments.*** We implemented a real-time video streaming test bed that enables the comparison of various loss recovery mechanisms and allows us to compare Prism's performance to baselines like I-frame based recovery and packet level retransmissions. This test bed uses the NvEnc [25] encoder and the H.264 [33] video codec[3], and implements a video streaming API that provides direct control over each frame's encoding configuration. For simplicity, we assume that the available bandwidth under normal operation is 20 Mbps, dropping down to 15 Mbps during periods of loss. Note that Prism can work with any congestion control algorithm, Prism just needs to know the instantaneous available bandwidth and needs a trigger for loss recovery. In these experiments, we use the MahiMahi [22] network emulator to emulate a 25 Mbps link with a 40 ms RTT. We evaluate two types of packet loss - (1) random packet loss, and (2) loss caused by queue-building flows (e.g. loading a web page with `wget` that uses TCP-Cubic).

In Figure 4, we show timelines comparing I-frame based recovery, packet retransmissions, and Prism under random loss (timelines for web page load provided in appendix D). Each loss event results in a large, sustained drop in video quality when using I-frames. While the quality of Prism's I-frames are slightly lower, the video quality recovers rapidly after the loss event. In the case of packet retransmissions, the delay spike that occurs after a loss event takes a significant amount of time to recover since the decoder needs to catch up to the latest frame after receiving the retransmissions for the lost packets. Secondary losses can compound this delay, resulting in very low QoE.

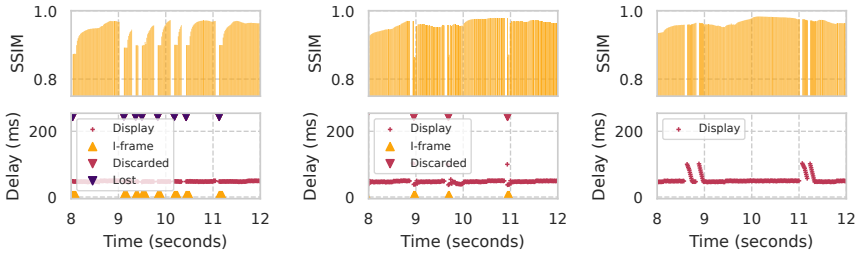
5.2.2 ***Trace-based Discrete Event Simulation.*** In order to evaluate Prism's performance under real-world network conditions, we designed a discrete event simulator that replicates Prism's protocol and integrates Prism's neural network based loss predictor. We evaluate Prism on ≈ 50 different network traces from M-Labs. These traces exhibit multiple loss events, and delay and bandwidth fluctuations, and were not used in the training of the loss predictor.

The bandwidth and delay of a sample trace is shown in Figure 5, along with the output of Prism's loss predictor. Aggressive loss prediction enables Prism to initiate recovery earlier, which reduces video stutter and delay. While Prism's loss predictor has quite a few false positives, recall that Prism's design significantly reduces the impact of falsely triggering loss recovery (§ 3.3, appendix E).

5.2.3 ***Results Summary.*** In Figure 6a, we compare the performance of Prism over the two emulated network conditions mentioned earlier (random loss, web-page load cross-traffic), and the network traces from M-Labs. The results are aggregated across 5 videos ("LoL TF", "Apex", "GTA-V", "Overwatch", and "FF XV"), and multiple runs for random packet loss and web-page load cross traffic (with staggered page load timing in each run), and across all the M-Labs traces. In our analysis, we use a simple clustering algorithm (see appendix C) to extract events of interest (e.g. I-frames, large stutter, frame loss) and compare the results from these periods (as opposed to the quality of an entire run, which may only have a few loss events) to highlight the performance of our system.

Across all the three scenarios, Prism achieves higher SQI-SSIM than a pure I-frame based approach for recovering from video data

[3]Any other codec that is supported by FFMPEG [31] and NvEnc [24] can be used.



(a) IDR-frames on packet loss. (b) Prism. (c) Packet retransmissions.

Figure 4: Prism vs. IDR-frames vs. Retransmissions (random loss)

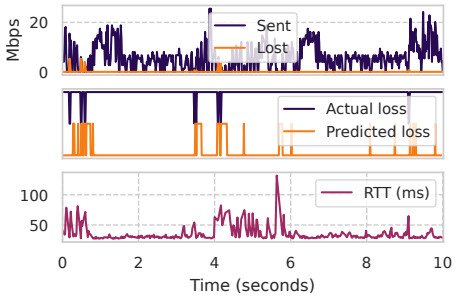
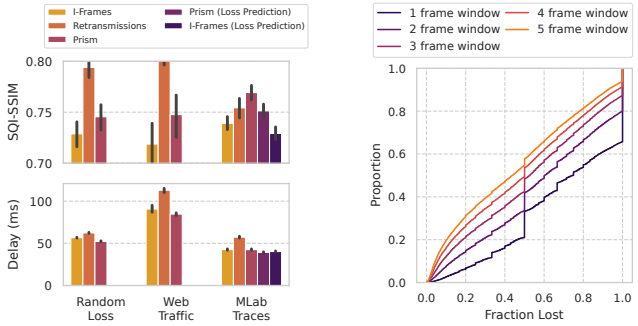


Figure 5: Prism loss prediction (MLabs).



(a) Prism's performance across various network types.

(b) Loss rate distribution for various timescales.

Figure 6

loss, since Prism's P-frame stream enables it to achieve higher video quality after recovery from loss (§ 1b). Prism's SQI-SSIM is also higher than retransmission based recovery on M-Labs traces - some of these traces have significant bandwidth and delay variations, and Prism's recovery stream is able to meet more playback deadlines due to its lower bitrate and since it is composed entirely of I-frames. In all cases, Prism achieves lower delay than I-frame based recovery and packet retransmissions. While it is clear why Prism's delay is lower than packet retransmissions (avoids delays from the decoder needing to catch up), the reasoning behind Prism's lower delay compared to I-frames is subtle - since the bandwidth of Prism's I-frames are lower, they have lower encoding, decoding, transmission and propagation delays.

This figure also shows the trade-offs of using loss prediction - Prism with loss prediction achieves lower end-to-end delay at the cost of lower video quality, but the video quality is much higher than I-frame based recovery with loss prediction since Prism's receiver simply ignores the recovery stream and uses the higher quality P-frame stream during a false loss recovery trigger.

6 RELATED WORK

A key body of related work is the field of error correcting codes (forward error correction). These techniques aim to prevent video data loss by transmitting redundant packets in addition to the base video data. A given FEC scheme makes the assumption that over a certain window of time, the loss rate is lower than a predetermined threshold. When packet loss exceeds this threshold, video data loss occurs and the only solution is a reactive loss recovery approach like Prism, I-frames or packet retransmissions. For handling 50% packet loss, the FEC code rate must be at least double of the video bitrate, and needs to be even higher for higher loss rates.

To understand the impact of Internet loss patterns on FEC designs, we plot the distribution of the fraction of data lost for different windows of time (Figure 6b). Even over durations as long as 5 frames (~80ms for 60FPS video), the 80th percentile loss rate is over 80%, which makes FEC impractical for low latency streaming applications [30]. FEC schemes that operate over larger windows add significant latency for recovery when losses occur, and also add a significant amount of computational overhead. Thus, systems that use FEC still require reactive loss recovery mechanisms to handle video data loss. Prism's design is complementary to FEC - Prism simply needs to know the effective available bandwidth after accounting for the overhead of using FEC. No other changes are required to Prism's core algorithm.

Salsify [9] proposes a joint video codec and network transport design that aims to avoid inducing losses by matching the size of the compressed frames to the estimated network capacity. While this approach is suitable for low bandwidth environments where the video traffic is itself likely to cause queuing and packet losses, Prism targets a different set of network conditions - cloud streaming applications require much higher bandwidth, and losses are often caused by external factors like queue-building cross traffic or noisy wireless links. We include results comparing NvEnc's and Salsify's rate control accuracy in appendix G.

7 CONCLUSION

In this paper, we presented the design of Prism, a novel approach for recovering from transient packet loss for ultra-low latency applications like cloud streaming. Prism uses a hybrid approach, splitting the available bandwidth between an I-frame stream and a P-frame stream to balance the video quality during loss recovery and video quality post-recovery while having zero overhead when network conditions are stable. Prism proposes a novel algorithm to model video SSIM which significantly reduces the computational requirements for optimizing the bandwidth allocation. Prism's approach also significantly reduces the impact of spurious losses on video quality, which enables aggressive loss prediction that can further reduce the end-to-end latency, resulting in overall improved QoE for low-latency interactive streaming applications like cloud gaming and AR/VR streaming.

REFERENCES

[1] Andy Backhouse and Irene YH Gu. 2004. A Bayesian framework-based end-to-end packet loss prediction in IP networks. In *IEEE Sixth International Symposium on Multimedia Software Engineering*. IEEE, 35–42.

[2] Nabajeet Barman, Saman Zadtootagha, Maria G Martini, and Sebastian Möller. 2021. https://www.lcevc.org/wp-content/uploads/Evaluation_of_MPEG-5_Part-2_LCEVC_for_Gaming_Video_Streaming_Applications_VQEG_CGI_2021_131.pdf Accessed on 07/11/2022.

[3] Callstats. 2015. Error resilience mechanisms for webrtc video communications. https://www.callstats.io/blog/2015/10/30/error-resilience-mechanisms-webrtc-video Accessed on 07/11/2022.

[4] Neal Cardwell, Yuchung Cheng, S Hassas Yeganeh, and Van Jacobson. 2017. BBR congestion control. *Working Draft, IETF Secretariat, Internet-Draft draft-cardwell-iccrg-bbr-congestion-control-00* (2017).

[5] Mark Claypool and Yali Zhu. 2003. Using interleaving to ameliorate the effects of packet loss in a video stream. In *23rd International Conference on Distributed Computing Systems Workshops, 2003. Proceedings.* IEEE, 508–513.

[6] Luca De Cicco, Saverio Mascolo, and Vittorio Palmisano. 2011. Feedback control for adaptive live video streaming. In *Proceedings of the second annual ACM conference on Multimedia systems*. 145–156.

[7] Zhengfang Duanmu, Kai Zeng, Kede Ma, Abdul Rehman, and Zhou Wang. 2016. A quality-of-experience index for streaming video. *IEEE Journal of Selected Topics in Signal Processing* 11, 1 (2016), 154–166.

[8] Nick Feamster and Hari Balakrishnan. 2002. Packet loss recovery for streaming video. In *12th International Packet Video Workshop*. PA: Pittsburgh, 9–16.

[9] Sadjad Fouladi, John Emmons, Emre Orbay, Catherine Wu, Riad S Wahby, and Keith Winstein. 2018. 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)*. 267–282.

[10] Anna Giannakou, Dipankar Dwivedi, and Sean Peisert. 2020. A machine learning approach for packet loss prediction in science flows. *Future Generation Computer Systems* 102 (2020), 190–197.

[11] Bernd Girod, Klaus Werner Stuhlmueller, M Link, and Uf Horn. 1998. Packet-loss-resilient Internet video streaming. In *Visual Communications and Image Processing'99*, Vol. 3653. International Society for Optics and Photonics, 833–844.

[12] Google. [n.d.]. Chrome Remote Desktop. https://remotedesktop.google.com/ Last accessed on 07/11/2022.

[13] Google. [n.d.]. Stadia. https://stadia.google.com/. Last accessed on 07/11/2022.

[14] Jason Greengrass, John Evans, and Ali C Begen. 2009. Not all packets are equal, part 2: The impact of network packet loss on video quality. *IEEE Internet Computing* 13, 2 (2009), 74–82.

[15] Roger Immich, Pedro Borges, Eduardo Cerqueira, and Marilia Curado. 2015. QoE-driven video delivery improvement using packet loss prediction. *International Journal of Parallel, Emergent and Distributed Systems* 30, 6 (2015), 478–493.

[16] Szymon Jakubczak and Dina Katabi. 2010. SoftCast: Clean-slate scalable wireless video. In *Proceedings of the 2010 ACM workshop on Wireless of the students, by the students, for the students*. 9–12.

[17] TV Lakshman, Upamanyu Madhow, and Bernhard Suter. 2000. TCP/IP performance with random loss and bidirectional congestion. *IEEE/ACM transactions on networking* 8, 5 (2000), 541–555.

[18] Didier J Le Gall. 1992. The MPEG video compression algorithm. *Signal Processing: Image Communication* 4, 2 (1992), 129–140.

[19] Yanlin Liu and Mark Claypool. 1999. Using redundancy to repair video damaged by network data loss. In *Multimedia Computing and Networking 2000*, Vol. 3969. International Society for Optics and Photonics, 73–84.

[20] Measurement Lab NDT 2021 Data. 2021. The M-Lab NDT Data Set. https://measurementlab.net/tests/ndt. Last accessed on 07/11/2022.

[21] Microsoft. [n.d.]. Deploy and scale your virtualized Windows desktops and apps on Azure. https://azure.microsoft.com/en-us/free/services/virtual-desktop

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

[23] GeForce Now. [n.d.]. GeForce NOW Gaming Anywhere |& Anytime. https://www.nvidia.com/en-us/geforce-now/. Last accessed on 07/11/2022.

[24] Nvidia. 2021. NvEnc Documentation. https://docs.nvidia.com/video-technologies/video-codec-sdk/nvenc-video-encoder-api-prog-guide/

[25] Abhijit Patait and Eric Young. 2016. High performance video encoding with NVIDIA GPUs. In *2016 GPU Technology Conference (https://goo. gl/Bdjdgm)*.

[26] Rohit Puri, Kannan Ramchandran, Kang-Won Lee, and Vaduvur Bharghavan. 2001. Forward error correction (FEC) codes based multiple description coding for Internet video streaming and multicast. *Signal Processing: Image Communication* 16, 8 (2001), 745–762.

[27] Lopamudra Roychoudhuri and Ehab S Al-Shaer. 2005. Real-time packet loss prediction based on end-to-end delay variation. *IEEE transactions on Network and Service Management* 2, 1 (2005), 29–38.

[28] Heiko Schwarz, Detlev Marpe, and Thomas Wiegand. 2007. Overview of the scalable video coding extension of the H. 264/AVC standard. *IEEE Transactions on circuits and systems for video technology* 17, 9 (2007), 1103–1120.

[29] Shailesh Shukla. 2021. Google Cloud Streams Augmented Reality. https://cloud.google.com/blog/products/networking/google-cloud-streams-augmented-reality Accessed on 07/11/2022.

[30] Ian Swett. 2016. Quic FEC V1. https://docs.google.com/document/d/1Hg1SaLEl6T4rEU9j-isovCo8VEjjnuCPTcLNJewj7Nk/edit#heading=h.vi0xq6pda8s3 Accessed on 07/11/2022.

[31] Suramya Tomar. 2006. Converting video formats with FFmpeg. *Linux Journal* 2006, 146 (2006), 10.

[32] Zhou Wang. 2003. The SSIM index for image quality assessment. *https://ece.uwaterloo. ca/˜ z70wang/research/ssim* (2003).

[33] Thomas Wiegand, Gary J Sullivan, Gisle Bjontegaard, and Ajay Luthra. 2003. Overview of the H. 264/AVC video coding standard. *IEEE Transactions on circuits and systems for video technology* 13, 7 (2003), 560–576.

[34] ST Worrall, AH Sadka, AM Kondoz, and P Sweeney. 2000. Motion adaptive intra refresh for MPEG-4. *Electronics Letters* 36, 23 (2000), 1924–1925.

[35] Microsoft xCloud. [n.d.]. Cloud gaming (Beta) with Xbox Game Pass: Xbox. https://www.xbox.com/en-US/xbox-game-pass/cloud-gaming/home. Last accessed on 07/11/2022.

[36] Yanling Xu and Yuanhua Zhou. 2004. H. 264 video communication based refined error concealment schemes. *IEEE Transactions on Consumer Electronics* 50, 4 (2004), 1135–1141.

[37] Bo Yan and Hamid Gharavi. 2008. Efficient error concealment for the whole-frame loss based on H. 264/AVC. In *2008 15th IEEE International Conference on Image Processing*. IEEE, 3064–3067.

A VIDEO DATASET

In our evaluation of Prism, we use videos from CGVDS [2]. This dataset includes a variety of gameplay footage with different gameplay and artistic styles. Screenshots from these videos are shown in Figure 7. Bejeweled, Lol-TF, MapleStory and Rayman Legends are examples where motion compensation is highly effective - these games have simple graphics with static backgrounds or simple translation motion of the background. Minecraft is an example where intra-compression is very effective, but motion compensation is less effective since the edges of the macro-pixels are hard to approximate by motion vectors alone. Overwatch, Dauntless, and Fortnite have simplified textures, and thus, efficient intra-compression is feasible. In addition, motion compensation is also effective, in contrast to Minecraft. Black Desert and FF-XV have fine textures, making intra-compensation less suitable. The footage from Apex Legends and GTA-V have a mix of fine textures (on ground surfaces) and large, flat gradients (like skies). In the case of Apex Legends, antialiasing is disabled, which adds complexity to the frame and temporal noise during movement, which makes motion compensation less effective. In the case of Tekken, intra-compression is less effective due to fine textures and complex arena design, but since the camera movement mostly involves panning, motion compensation is very effective.

B BANDWIDTH OPTIMIZATION

In Figure 8, we show the difference in the video quality improvement achieved by Prism's per-video optimization strategy and an alternate strategy that generates a single bandwidth allocation table across all of the videos. When using a single bandwidth allocation table, the improvement in video quality can be up to 15 percentage points lower at the 80th percentile, with a heavy tail. This demonstrates that optimizing per-video is necessary. Note that since our algorithm is greedy and not truly optimal, the single bandwidth allocation table gets lucky sometimes and beats the per-video optimized allocation table - hence, there are some data points on the negative side of the X-axis.

C EVENT CLUSTERING

In order to highlight Prism's benefits in the overall results shown in Figure 6a in the main paper, we use a simple filter that detects frame clusters that are of interest. The filter identifies frames in the video where there were large delay spikes, missing frames, I-frames and retransmissions, and expands these events into windows of interest that include additional frames after the event in order to account for the transition properties of the video quality of P-frames. In all the timeseries discussed in this section, the red boxes highlight the events automatically detected by our filtering algorithm.

D WEBPAGE LOAD TIMESERIES

We evaluated Prism's real-time implementation over an emulated bottleneck using MahiMahi [22], where we induce packet losses by introducing a competing Cubic flow that downloads a web page. In this experiment, we use a simple RTT threshold filter as a dummy loss prediction technique in order to demonstrate how loss prediction impacts the video quality and delay. The time series depicting the performance of I-frames (SSIM and frame delay) is shown in Figure 13. During each run, the web page is downloaded 3 times, and each page load lasts for a duration of about 1.5 seconds, which can be seen by the regions with increased delay. For each event, there is one false loss trigger at the beginning (due to the increased RTT as Cubic starts filling the buffer), and one true loss event. I-frames cause a significant drop in video quality during the false loss trigger and the actual loss event, and takes a while to recover. Figure 14 depicts the use of packet retransmissions for recovering from packet loss. There is no drop in the video quality, but the video stalls until the missing packet is retransmitted. This also causes high delays after recovery while the decoder catches up. Figure 15 shows the timeline for Prism. The three key benefits of Prism are highlighted here - (1) The drop in video quality when a loss is falsely triggered is minimal, (2) Once loss recovery is complete, the video quality recovers to the steady state P-frame quality quickly, and (3) the delay remains low throughout the recovery process.

E M-LABS TRACE TIMESERIES

Figure 16 shows the timeline for Prism for one trace from the M-Labs dataset without loss prediction, and Figures 17 and 18 show the timelines for Prism and I-frames respectively when using loss prediction. Loss prediction reduces the end-to-end frame delay, but false predictions in the case of I-frames significantly impact video quality. Prism avoids this drop in some cases when the loss recovery is falsely triggered (first two events), and thus achieves overall higher video quality than I-frame based recovery.

F LOSS PREDICTION

Prism's loss prediction neural network uses data from a 200 ms window in the past in order to predict if there will be a loss in the next 50 ms. The 200 ms window comprises of multiple overlapping 50 ms windows with a stride of 10 ms. This is shown in Figure 11. The architecture of Prism's loss predictor is shown in Figure 12. During training, we filtered the data to only include traces that contained at least 500 windows and had a maximum RTT less than 200 ms. Since the training data was heavily skewed towards examples with no loss (less than 20% of traces contained any loss, less than 3% of training windows contained loss). During training, we oversample the windows with loss in order to reduce training bias. Prism's loss predictor as presented in this paper is just used as an example of what can be achieved when such techniques are combined with Prism, and is a supplementary tool used in our evaluation.

G ENCODER RATE CONTROL

We ran some simple tests to evaluate the rate-control accuracy of NvEnc [24], and a design like Alfalfa(used in Salsify [9]). We encoded each video 5 times with a random bitrate for each frame. For Alfalfa, we used the REALTIME_QUALITY preset with two pass encoding, and for NvEnc we used recommended options by NVIDIA for real-time streaming [24]. Alfalfa can overshoot by 2X for bitrates lower than 10 Mbps at the 90th percentile, whereas we do not see any overshoots at the 90th percentile for NvEnc. Thus, rate control algorithms of hardware-based video codecs today work extremely well, and there is no need to use a slower software-based codecs like Alfalfa.

(a) Bejeweled 3 (b) Black Desert (c) Dauntless (d) FF XV (e) Fortnite (f) GTA V (g) LoL Teamfight

(h) Maple Story (i) Minecraft (j) Overwatch (k) Apex Legends (l) Rayman Legends (m) Tekken

Figure 7: Screenshots from the raw videos.

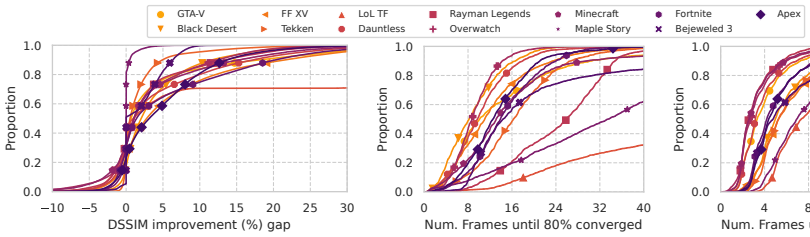


Figure 8: Comparison of per-video optimization vs. optimizing jointly for all the videos.

Figure 9: Convergence of P-frame SSIM when the bitrate is decreased.

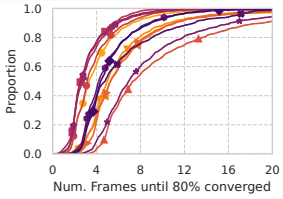


Figure 10: Convergence of P-frame SSIM when the bitrate is increased.

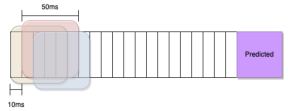


Figure 11: Prism's loss prediction input window and output window.

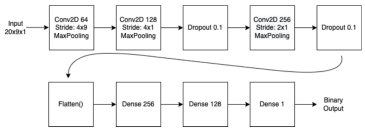


Figure 12: Prism's loss prediction input window and output window.

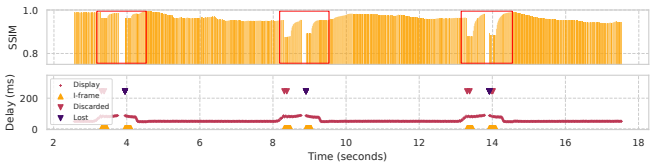


Figure 13: I-frame timeseries for loss caused by a competing flow performing a web page load.

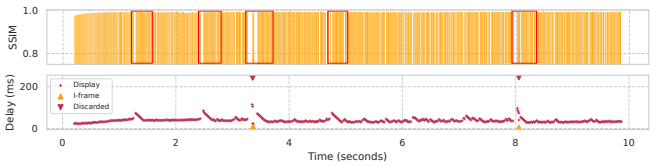


Figure 16: Prism timeseries for an MLabs trace (no loss prediction). Video quality quickly recovers after a loss event.

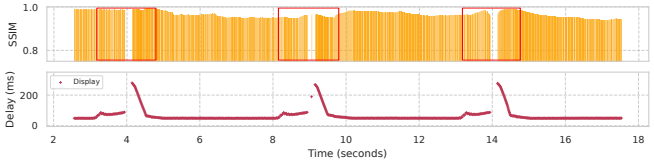


Figure 14: P-frame timeseries for loss caused by a competing flow performing a web page load.

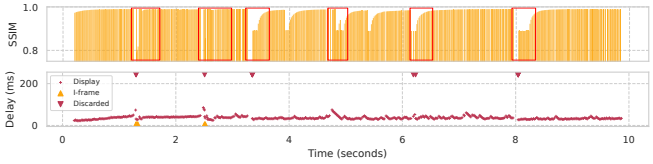


Figure 17: Prism timeseries for an MLabs trace (loss prediction). Some false triggers cause quality drops, but the delay reduced.

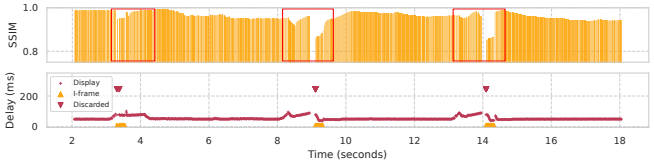


Figure 15: Prism timeseries for loss caused by a competing flow performing a web page load.

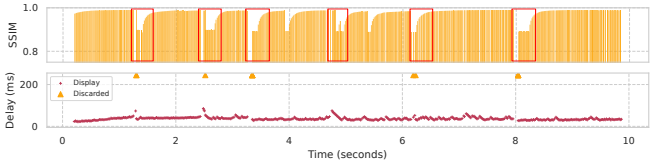


Figure 18: I-frame timeseries with loss prediction for an MLabs trace. Note the frequent drops in video quality for events where loss recovery is triggered without an actuall loss.