STREAMING LOW-BANDWIDTH REAL-TIME VIDEO USING VIDEO SUPER-RESOLUTION

BY

SANCHIT VOHRA

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Electrical and Computer Engineering
in the Graduate College of the
University of Illinois Urbana-Champaign, 2022

Urbana, Illinois

Adviser:

Professor Sanjay Patel

ABSTRACT

Jalapeño is a real-time video streaming platform designed primarily for tele-operations. The platform uses a traditional video compression approach (H264) and pairs that with unique networking optimizations and artificial super-scaling to increase reliability and decrease bandwidth consumption. Jalapeño assumes the vehicle (client) is running the platform on a device with low computational capabilities. Conversely, the operator is controlling the vehicle on a system with very high resources. Additionally, the client network is assumed to be transmitting over a lossy wireless channel. Real-time streaming also has tighter constraints than traditional live-streaming, hence some networking algorithms that rely on introducing artificial stream delays are infeasible. Jalapeño must alleviate the issues of video streaming under heavy loss conditions with a low-powered client under the challenging real-time constraints. The networking layer closely monitors the quality of the stream and adjusts the parameters of the compression and transmission dynamically. We leverage the temporal scalable video codec capabilities of the open-source OpenH264 codec by Cisco to design a reconstruction algorithm that works under real-time constraints to mitigate packet loss. The prime feature is video super-resolution, a GAN based generator that is able to upscale a low-resolution stream from the client to high-definition frames while maintaining temporal consistency. This thesis also outlines an approach to build a large video dataset for unsupervised video learning and discusses compression-aware super-resolution to advance the framework to the next level.

To my parents, brother, and girlfriend, for their love and support.

ACKNOWLEDGMENTS

Thank you to the members of the Jalapeño group for all your hard work and contributions. To Ananmay Jain, for all the discussions about the design and vision for the platform. To Chirag Kikkeri, for taking the initiative and not being afraid to attempt something new. To Evan Chen, for being the rock of the team. To Alan Andrade, for always giving it your all. Finally, to Prof. Sanjay Patel, thank you for pushing us and guiding us on this journey. On many occasions, your excitement about our progress revitalized the team. Your trust in us kept us striving forward in the toughest of times.

TABLE OF CONTENTS

LIST OF ABBREVIATIONS

AVC	Advanced Video Coding
DCT	Discrete Cosine Transform
DON	Decoding Order Number
FEC	Forward Error Correction
GOP	Group of Pictures
ICE	Interactive Connectivity Establishment
MTU	Maximum Transmission Unit
NALU	Network Abstraction Layer Unit
NAT	Network Address Translation
RGB	Red Green Blue
RTP	Real-time Transport Protocol
SRT	Secure Reliable Transport
STAP	Single Time Aggregation Packets
SVC	Scalable Video Coding
TID	Temporal ID
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
UUID	Universally Unique Identifiers

CHAPTER 1

INTRODUCTION

With the rapid development of autonomous vehicles comes the need for teleoperations to remotely operate or closely monitor vehicles in areas where autonomy is uncertain or challenging. Jalapeño explores building a video streaming platform designed for teleoperations. Real-time video streaming has tighter constraints than traditional live-streaming applications. For the human operator to maneuver the vehicle, minimizing the stream latency becomes essential. Additionally, as the platform scales to support multiple clients, the bandwidth cost must not become a bottleneck. Jalapeño explores how to tackle these problems.

The system model for teleoperations is outlined in this paragraph. The vehicle, or client, capabilities are computationally limited. The only onboard hardware acceleration assumed is for video encoding. Conversely, the operator has access to a system with very high compute and a dedicated GPU to accelerate operations. The client(s) are transmitting over a lossy mobile network. Frequent bursty packet drops are expected and must be handled by the system.

Chapter 2 discusses the video compression technique used by the platform. Jalapeño uses the H264 codec for compressing video. H264 is the preferred codec because of the wide availability of hardware accelerators on embedded platforms. The compressed video packets are transmitted P2P from the client to the operator using the RTP protocol. Jalapeño constantly monitors the network to estimate stream quality and adapts the H264 stream to fit dynamic network conditions. We also explore using temporal SVC to divide the video stream into layers.

Chapter 3 discusses the networking stack used to transmit the compressed

video stream. We discuss how the network topology of the system is laid out and how we establish connections between all the devices. We use the RTP protocol to transmit H264 packets. Additionally, we explore using error correction techniques and implement a novel reconstruction algorithm that leverages the temporal SVC feature of H264.

Chapter 4 explores the system model and concurrency design used to engineer the platform. We used an asynchronous runtime developed using the Boost.Asio library. This environment allows the platform to scale while using system resources efficiently.

Chapter 5 explores using video super-resolution models to convert a transmitted low-resolution stream to high-definition frames in real-time. Jalapeño uses the TecoGAN and EGVSR model as the starting point for its video super-resolution approach. These models achieve 30fps 270p to 1080p conversion on the GPU used in the experiments. The models are augmented with a better training objective and trained on a large-scale unsupervised video dataset. We also explore compression-aware super-resolution in which the model is trained to additionally alleviate packet loss and compression artifacts.

Finally, Chapter 6 summarizes the thesis and outlines future work.

CHAPTER 2

VIDEO COMPRESSION

Transferring high-quality video streams is the centerpiece of Jalapeño's teleoperations platform. Jalapeño employs the H264 video compression standard (also called MPEG-4 AVC, Advanced Video Coding) for video compression. Video data consists of a sequence of images called frames. Each image frame is a two-dimensional array of pixels. Each pixel has three digitized color components: red, green, and blue (RGB). Before encoding, the frames are converted into the YUV color space. In the YUV space, the Y component is the 'luma' which is the grayscale intensity signal of the image. The UV components of the image impart the color signals. Then, the UV channels are downsampled to half their original resolution (also known as the YUV 4:2:0 format).

H264 uses a block-based encoding approach. Each macroblock is a subsection of the frame (a commonly used configuration is 16x16 Y and 8x8 UV pixels). For each macroblock in the image, H264 computes a prediction macroblock. The difference between the macroblock in the current frame and the computed prediction macroblock, i.e., the residual, is easier to compress than encoding the entire macroblock itself. The residual is first transformed into the frequency domain using the discrete cosine transform (DCT). From there, quantization is applied to retain low-frequency information.[1] Finally, H264 uses entropy encoding to store the data in efficient variable-length codes.

H264 uses two prediction methods for real-time video encoding: intra-frame prediction and inter-frame prediction. Computing the prediction macroblock involves exploiting spatial or temporal redundancy in the video stream. For intra-frame prediction, H264 calculates the prediction macroblock using neigh-

[1]Human vision is more sensitive to low-frequency signals and less sensitive to high-frequency ones.

boring blocks in the image (spatial redundancy). For inter-frame prediction, H264 calculates the prediction macroblock using motion estimation. H264 computes the motion estimation from the previous frame (temporal redundancy). Figure 2.1 shows the procedure to encode a frame in pseudocode.

```
procedure encode_a_frame (f_t, mode)

    for I = 1, N          //** N: #rows of MBs per frame
        for J = 1, M      //** M: #columns of MBs per frame
            Curr_MB = MB(f_t, I, J);
            case (mode)
                I: Pred_MB = Intra_Pred (f'_t, I, J);
                P: Pred_MB = ME (f'_t-1, I, J);
                B: Pred_MB = ME (f'_t-1, f'_t+1, I, J);

            Res_MB = Curr_MB – Pred_MB;
            Res_Coef = Quant(Transform(Res_MB));
            Output(Entropy_code(Res_Coef));

            Reconst_res = ITransform(IQuant(Res_Coef)) ;
            Reconst_MB = Reconst_res + Pred_MB;
            Insert(Reconst_MB, f'_t) ;

end encode_a_frame;
```

Figure 2.1: Pseudocode for the H264 encoding process from [1].

The H264 video compression standard is ideal for a teleoperations platform for many reasons. The design of the H264 standard accounts for the use of lossy network transmission. The H264 specification describes a network abstraction layer (NAL) which packetizes the compressed data for transmission. While other video compression standards like H265 (HEVC), VP8, and VP9 also provide network abstraction design in their specifications, H264 has faster encoding (albeit with lower quality). The faster encoding of H264 makes it possible to use the codec in real-time video-compression applications on low-powered devices such as drones, small autonomous robots, etc. Additionally, common embedded devices such as the Nvidia Jetson series and the TI Jacinto line provide hardware acceleration support which further reduces the latency and power consumption. H264 also has open-source software

implementations (OpenH264 by Cisco and x264) which make development easier. However, H264 (and HEVC) are proprietary codecs; a licensing fee must be paid for using the H264 compression standard unlike the open-source VP8 and VP9 codecs.

2.1 OpenH264 by Cisco

Jalapeño uses the OpenH264 codec by Cisco [2] to implement the H264 codec. OpenH264 is an open-source software (CPU) implementation of the H264 standard written in C++. It supports encoding video in the YUV 4:2:0 format and provides numerous options to packetize the compressed data for transmission. Additionally, the OpenH264 encoder has the ability to dynamically change the compression settings, supports four temporal scalability layers (SVC), and can insert IDR frames on demand. These settings of the codec are leveraged to optimize Jalapeño's approach for real-time video transmission scenarios such as teleoperations. OpenH264 allows users to define the bitrate at which the video stream will be compressed.[2] Jalapeño can use this setting to adjust the stream based on the available bandwidth of the network.

2.2 I-frames, P-frames, Packetization, and GoP

H264 can output two types of output frames based on macroblock composition: I-frames and P-frames. I-frames consist of only intra-frame prediction macroblocks. I-frames can be encoded and decoded completely independently of other frames (since they contain no inter frame dependencies). However (as seen in Figure 2.2), they are typically larger in size than P-frames. I-frames are also known as reference frames or key frames. P-frames can be composed of inter-frame prediction and intra-frame prediction macroblocks. P-frames are typically smaller than I-frames but depend on previous frames. P-frames can be thought of as encoding the change in the video from the previous frame(s). For this reason, P-frames are also known as delta frames.

[2]OpenH264 will sometimes skip frames to ensure the bitrate requirement is being met.

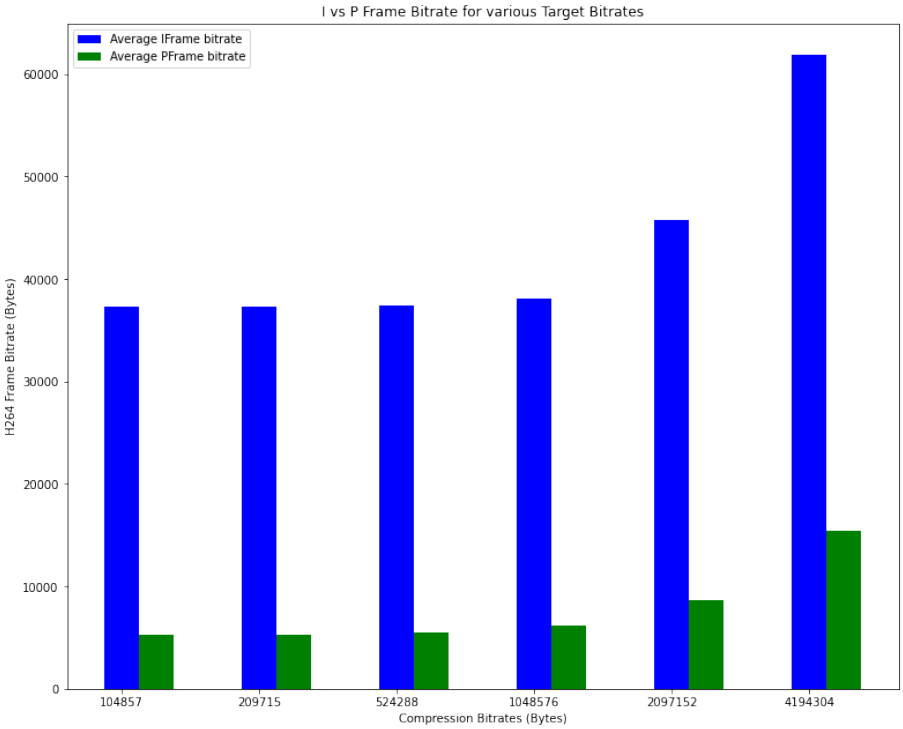

Figure 2.2: Bitrate of I-frames vs P-frames for different compression settings in OpenH264. The typical I-frame requires much larger bandwidth than the typical P-frame for all encoder settings.

Each H264 frame is outputted as NALU (network abstraction layer unit) packets. Each NALU is a chunk of the frame byte-stream. NALUs of the same frame can be passed to the decoder in arbitrary order. This allows the network to rearrange the video stream packets without affecting the stream decoding. In Jalapeño, the size of the NALUs is limited to 1500 bytes (MTU). Further study on how the change in NALU size impacts the video stream during network transmission is required.

Real-time video transmission occurs over lossy channels. OpenH264 provides error concealment algorithms to handle missing NALU during decoding. As seen from Figures 2.3 and 2.4, increasing packet loss negatively impacts the reconstruction quality of the decoded H264 bitstream despite error concealment.

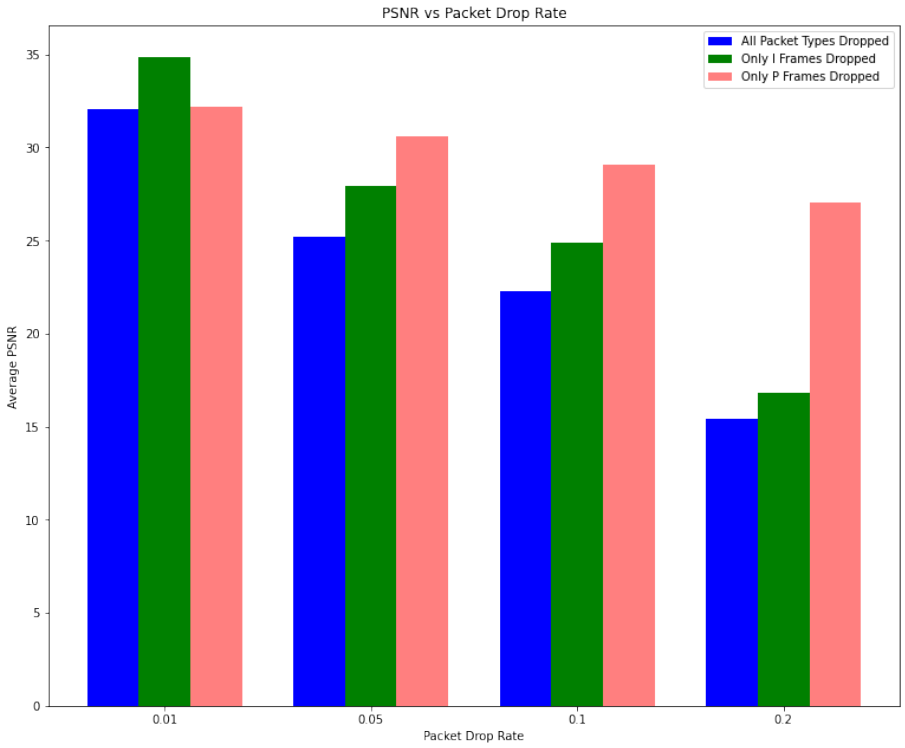

Figure 2.3: PSNR of original vs. decoded frames vs. packet drop rate. Higher drop rate results in lower stream quality. Dropping I-frames only results in a much worse quality stream than dropping just P-frames. This indicates that more essential reconstruction information is encoded in the I-frames. Intuitively that makes sense as P-frames encode the (typically small) delta between video frames while I-frames encode the pixel data directly.

While transmitting real-time video, the H264 encoder is configured to output a group of pictures (GoP) consisting of a single I-frame followed by a series of P-frames (as seen in Figure 2.5). If the GoP size is kept large (more P-frames), then the video stream requires less bandwidth for transmission (however P-frame bandwidth dominates as seen from Figure 2.6). However, in a lossy channel, this video stream is more prone to error artifacts due to lost P-frame packets propagating delta errors. A smaller GoP will produce a higher bandwidth video stream but will be more resilient to dropped packets. Note that dropped I-frame packets result in more significant frame errors than dropped P-frame packets. Hence, sending more frequent I-frame packets in smaller GoP streams is another way to combat packet loss.

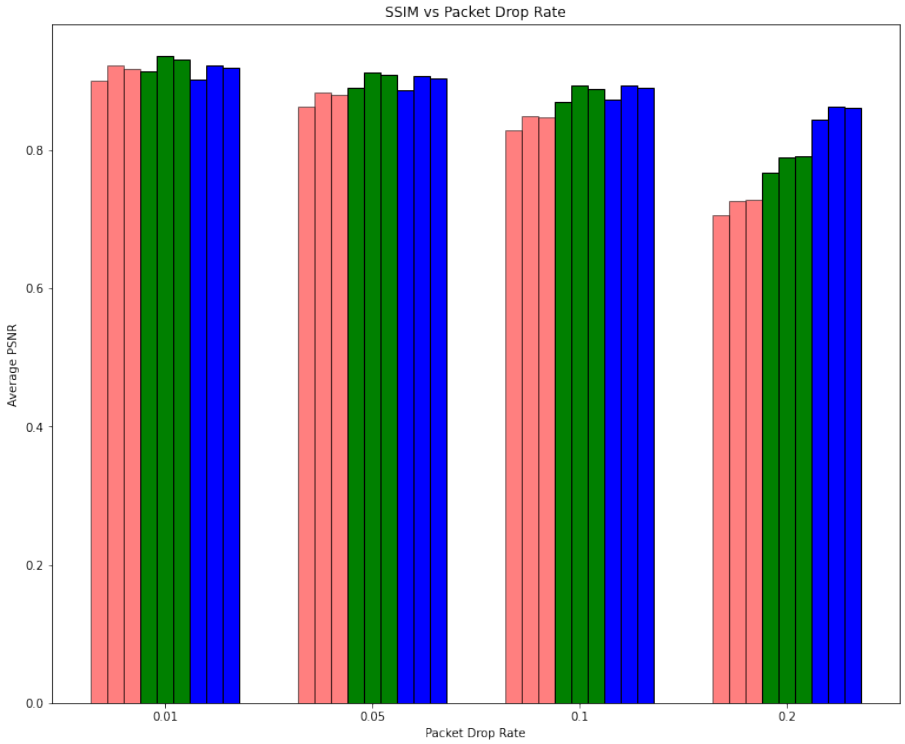

Figure 2.4: Per channel MSSIM of original vs. decoded frames vs. packet drop rate. SSIM score computes the reconstruction score per channel in blocks and is another way of interpreting reconstruction quality.

Jalapeño supports variable GoP video streams. The GoP size is dynamically adjusted with the estimated network parameters (packet drop rate in particular). High packet loss occurs as a result of video stream bandwidth exceeding stream capacity. Therefore, decreasing GoP to combat packet loss seems like a counterintuitive idea. Hence, Jalapeño couples the GoP adjustment with dynamic stream resolution, video super-resolution, forward error correction, and temporal reconstruction.

2.3 Temporal SVC

Another feature in the OpenH264 library is SVC (scalable video coding). SVC is an extension to the H264 compression standard. In SVC, the compressed video stream is split into multiple layers. The lowest layer (base layer) functions exactly like a regular compressed video stream. Higher lay-

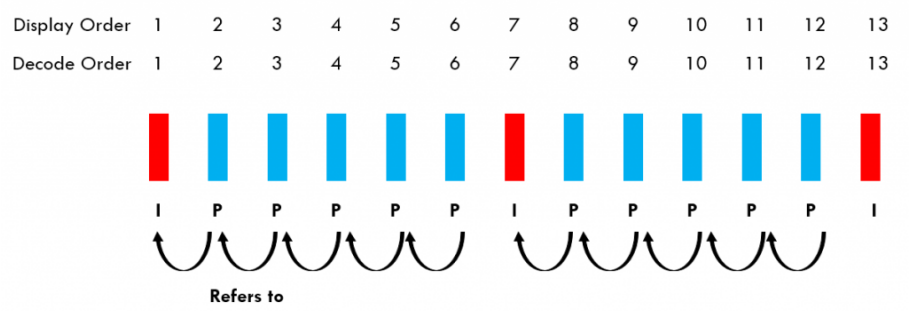

Figure 2.5: Visualization of a GoP. The I-frames are interleaved with a sequence of dependent P-frames. The next I-frame, and the start of the new GoP, is independent of any previous frames.

ers are dependent on the base layer and lower layers. Decoding the higher layer bitstream imparts more information into the base layer video stream. For example, the base layer can be a compressed low-resolution 360p video stream. A higher order layer (along with the base layer) will decode a higher resolution 1080p video stream. H264 supports SVC layers for scalable frame-rate (temporal layers), resolution (spatial layers), and quality (quality layers). These SVC layers can be combined. For example, a H264 SVC video stream can have higher order spatial, temporal, and quality layers. Figure 2.7 shows the packet layers with dependencies in a temporal SVC stream.

OpenH264 supports SVC with upto four temporal layers (including the base layer). The frame rate of the encoded video stream doubles every temporal layer.[3] Jalapeño leverages the temporal scalability in the compressed video stream to be more resilient to packet loss and dynamic network conditions. When the available bandwidth of the network is less than the combined bandwidth requirement of the temporal layers, Jalapeño can either adjust the bitrate requirement of higher temporal layers or omit transmitting the higher order temporal layers entirely. Additionally, Jalapeño protects the tid=0 (lowest temporal layer) packets in the networking layer (temporal reconstruction) as tid=0 packets account for most of the reconstruction quality (see Figures 2.8 and 2.9). The temporal reconstruction leverages the fact that there is a large delay between tid=0 frames (around 120 ms) enabling the

[3]OpenH264 also supports simulcasting video streams of different resolutions. In simulcasted streams, the higher spatial resolution bitstream is not dependent on the lower resolution bitstreams. Hence, this feature is not useful for real-time video encoding.

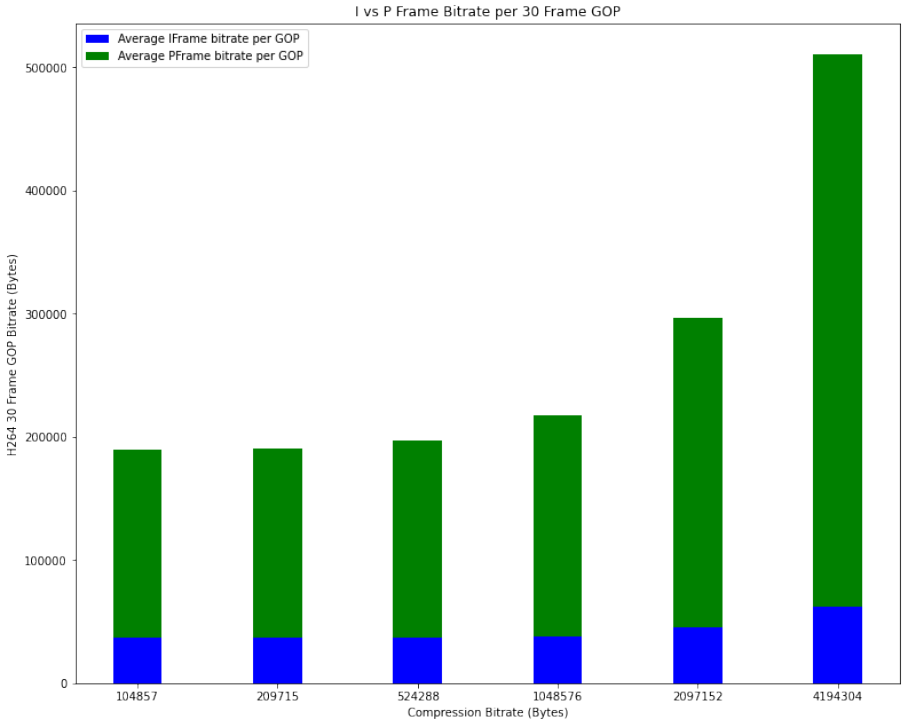

Figure 2.6: I-frame vs P-frame bandwidth requirement per 30 frame GoP in OpenH264. Even though I-frames are much larger (typically) than P-frames, a GoP consists of multiple P-frames after each I-frame.

use of buffer reconstruction algorithms which normally would be infeasible for real-time 30 fps video streams.

2.4 Dynamic Stream Control

In addition to I-frames and P-frames, H264 outputs stream parameter information packets (also known as SPS/PPS). These packets are output at the beginning of every I-frame and consist of essential stream parameters (frame format, picture resolution, SVC settings). The SPS/PPS bitstream must be decoded before the I-frame bitstream. If the SPS/PPS packets are dropped, the OpenH264 decoder is unable to decode the remaining bitstream until the next I-frame parameters. This presents a challenge for real-time streaming because a loss in the SPS/PPS packets would result in the loss of the stream for an entire GoP. In order to combat this issue, Jalapeño employs a short handshake protocol via TCP before starting transmission. The potential stream settings are predefined (different resolution, temporal layers, etc.).

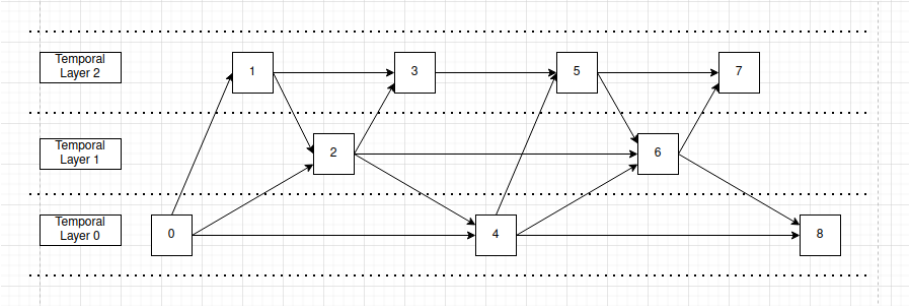

Figure 2.7: Representation of a H264 stream with three temporal layers. The blocks represent frames and the arrows represent dependence relations while decoding.

Each stream setting is mapped to an 8-bit identifier (256 possible stream settings). The SPS/PPS packets for each stream configuration are transmitted during the handshake and mapped in the remote decoder (operator). With the handshake complete, the encoder can attach the 8-bit configuration identification number with every I-frame packet. The transmission of just one I-frame packet enables the decoder to retrieve the necessary SPS/PPS packets for successful decoding.[4]

[4]The CONSTANT SPS/PPS setting must be enabled in OpenH64 for this strategy to work. This ensures that each successive I-frame in the same configuration can be decoded using the same parameter packets.

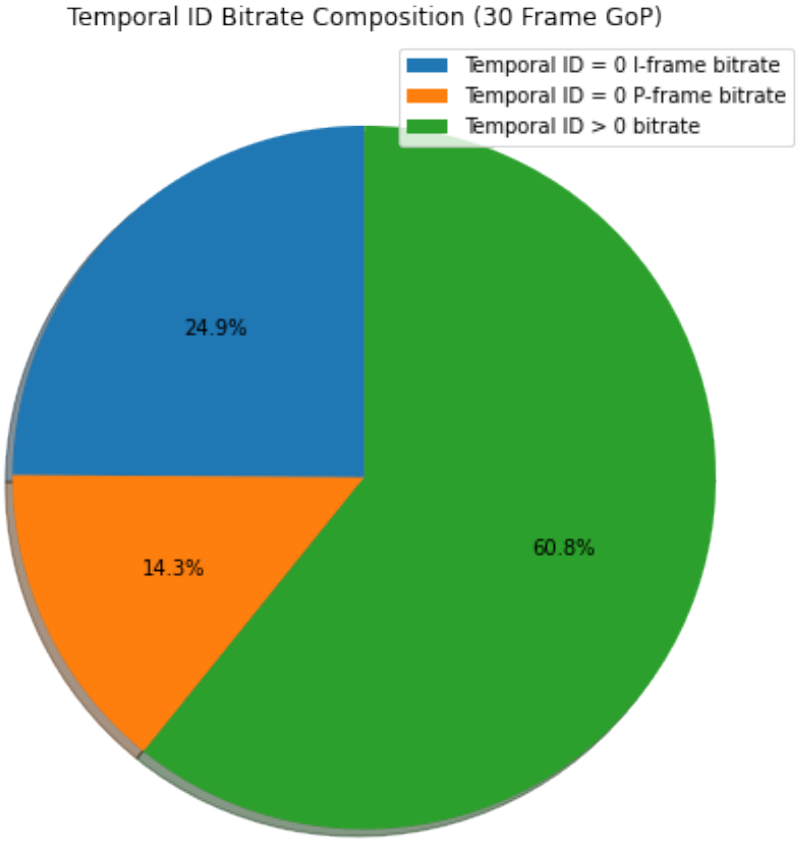

Figure 2.8: Composition of bitrate for video stream using temporal SVC in OpenH264 using four temporal layers for a 30 frame GOP. The lowest temporal layer tid = 0 contains I-frame and P-frame data and represents a 3.25 FPS stream. The higher order layers tid = 1, 2, 3 impart information for a 7.5, 15, 30 fps stream respectively. Tid = 0 can be independently reconstructed and hence contains a significant piece of information despite consisting of fewer frames.

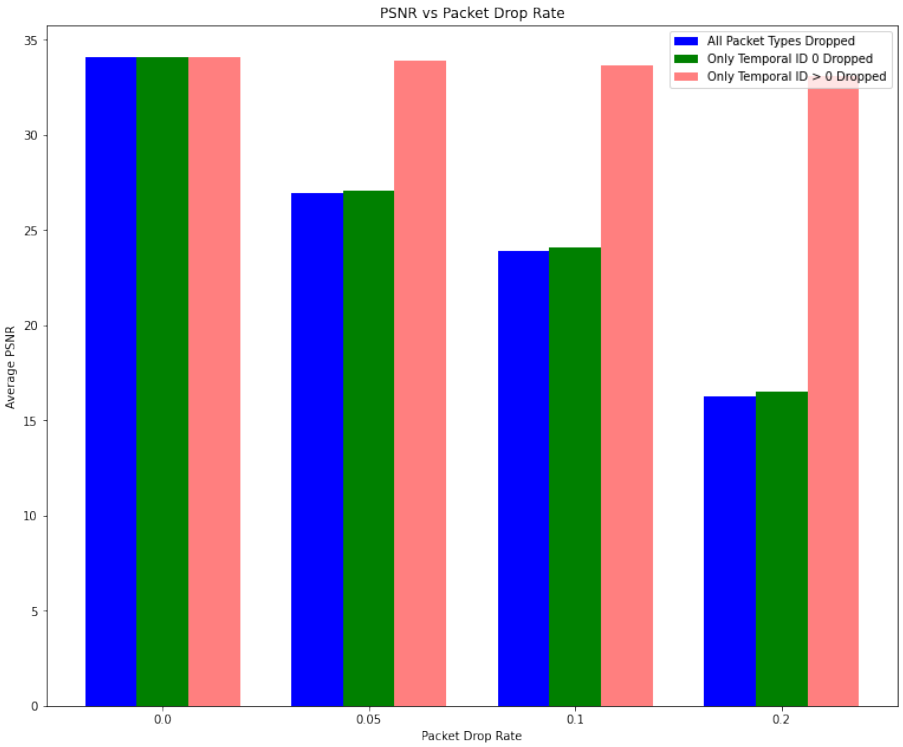

Figure 2.9: The effect of packet loss in an H264 SVC stream for tid = 0 vs. tid > 0. As seen from the PSNR graphs, the tid=0 information accounts for most of the reconstruction quality.

CHAPTER 3

NETWORKING

The networking stack is a critical component of the Jalapeño platform. In contrast to live-streaming applications (such as Twitch) that can afford to introduce artificial delays into the stream, the real-time constraints of tele-operations prioritizes timeliness over reliability. However, leveraging the temporal SVC feature of OpenH264, Jalapeño borrows ideas from buffer reconstruction algorithms to improve stream quality while maintaining real-time constraints. Additionally, Jalapeño measures the parameters (available bandwidth, latency, jitter) of the network and uses a feedback protocol to dynamically adjust the H264 stream settings.

3.1 Jalapeño Network Topology

The network topology of the Jalapeño platform (seen in Figure 3.1) consists of many clients (drones, cars, etc.), a public central server (hosted on AWS), and operators (fewer than the number of clients). The clients and operators connect to the server via TCP. The server registers the clients and operators via a handshake mechanism. In this handshake mechanism, each client and operator is assigned a 128 bit unique UUID. After the handshake, each client is assigned to one operator. The server ensures that the load among the operators is balanced. The server relays the client ID to the operator and vice-versa. Henceforth, the server can act as a relay to transmit messages between the client and the assigned operator.[1]

Transmitting video packets via a relay server is slow; additional delay is being added to the transmission via the relay, and expensive, additional bandwidth is required for the server to retransmit the packet. Thus, the server sets up

[1]The message must contain the destination ID that was communicated earlier.

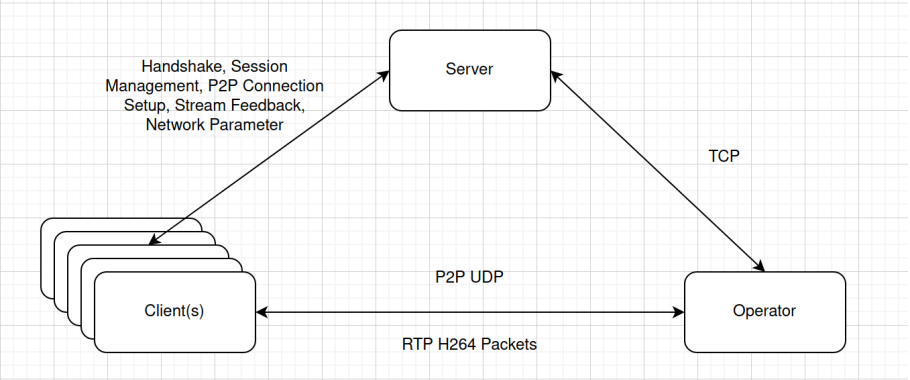

Figure 3.1: A depiction of Jalapeño's network topology and the corresponding connections.

a P2P connection between the client and the operator. However, unlike the public server, the client and operator are (likely) behind NAT. To bypass the NAT (hole punching), Jalapeño implements the ICE (Interactive Connectivity Establishment) protocol to establish the P2P connection. The candidate addresses of the client and operator are relayed via the server and the NAT is bypassed by creating the appropriate mappings in the translation tables of the routers.[2]

3.2 Real-time Transport Protocol (RTP)

Jalapeño uses the RTP [4] (Real-time Transport Protocol) to transmit H264 packets (NALUs). RTP defines a packet format that consists of an RTP header followed by a payload (See Figure 3.2). The H264 extension for RTP [5] details the payload format when transmitting video streams. Because OpenH264 can decode out-of-order NALUs of the same frame, Jalapeño uses the STAP-A extension (see Figure 3.3) format to aggregate NALUs into a single packet without needing a DON (decoding order number).

Jalapeño transmits RTP video packets over the UDP network layer. UDP prioritizes timeliness over reliability; packets may be dropped or reordered but will be delivered with minimum delay (no overhead like TCP).[3]

[2]Jalapeño uses Libjuice [3], a lightweight ICE implementation written in C to establish the P2P connections.

[3]Recall that video streams can handle packet loss because of error concealment. The error concealment strategies are built into the OpenH264 decoder.

Offsets	Octet	0								1								2								3							
Octet	Bit [a]	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	0	Version			P	X	CC			M	PT							Sequence number															
4	32	Timestamp																															
8	64	SSRC identifier																															
12	96	CSRC identifiers ...																															
12+4×CC	96+32×CC	Profile-specific extension header ID																Extension header length															
16+4×CC	128+32×CC	Extension header ...																															

Figure 3.2: RTP header format. The important fields are the sequence number (which is used for network parameter estimation, feedback, and buffer reconstruction algorithms) and the user-defined payload type (PT). The RTP fields in red are optional.

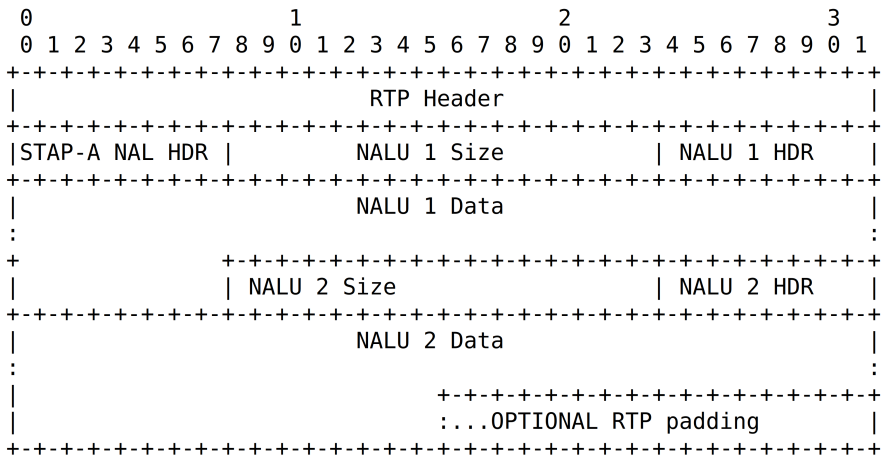

Figure 3.3: An example of an RTP packet including an STAP-A payload containing two NALUs of the same frame.

3.3 Stream Feedback

To deliver efficient real-time teleoperations performance, Jalapeño dynamically adjusts the video stream to fit the network. A network parameter estimation protocol is run in parallel to RTP to compute the estimated available bandwidth, latency, and jitter. This protocol takes inspiration from the SRT [6] protocol's network estimation. The network parameter estimation protocol periodically sends probe messages from the client to the operator via the P2P channel. The client probe message p_c contains a unique identifier $p_c.id$ generated for each probe run as well as the send timestamp $p_c.send$. On receiving the client probe message p_c, the operator replies with two successive replies p_{o1} and p_o2 with the same probe identifier, i.e. $p_c.id = p_{o1}.id = p_{o2}.id$. The operator also includes the send timestamp of the reply $p_{o1}.send = p_{o2}.send$. When the client receives the probe reply messages from the

operator, it can compute the estimated network parameters. The latency and jitter are computed when the first reply is received while bandwidth is computed after the second reply in the run is received. For simplicity, we assume that p_{o1} arrives before p_{o2} at the client.

Latency Estimation: The latency is computed as the round-trip time of the probe message / 2 i.e. $(p_{o1}.recv - p_c.send)/2$.

Jitter Estimation: The jitter is computed as the standard deviation of the latency measurements of the last N runs where N is a configurable window.

Bandwidth Estimation: The bandwidth is estimated by dividing the size of the second probe reply with the difference in the time to receive both the timestamps, i.e $size(p_{o2})/(p_{o2}.recv - p_{o1}.recv)$.

An additional layer of reliability is added by median filtering all the parameters over a specified window M. The median of the parameter over the last M runs is calculated. Any data point outside $M/F < x < M * F$ is filtered away and the average of the remaining points is computed. This makes the estimates resilient to outliers and prevents large fluctuations between runs.

The client uses the estimated bandwidth to change the bitrate of the encoder dynamically. The encoder bitrate is set to the available bandwidth at the start of every GoP. Additionally, the client may change the spatial resolution of the stream. For example, if the available bandwidth of the network drops below certain thresholds, encoding a lower spatial resolution with higher quality will yield better results than encoding a higher spatial resolution.[4]

The operator uses the network parameter estimates to detect dropped video stream packets. On receiving the first packet for a frame f at time t, the operator NACKs all packets not received by $t + r * j$ where j is the estimated jitter and r is a configurable ratio (set between 1.5 and 2 in our case).

[4]The number of skipped frames will become intolerably large if the bitrate drops too low for a particular resolution. At this point, it becomes better to encode the video at a lower spatial resolution.

Consequently, the client can estimate the health of the video stream at the operator. A dropped I-frame packet and temporal layer 0 packet is more significant for stream quality. A fresh IDR frame (GoP restart) is forced at the client if the estimated stream degradation crosses certain thresholds. Additionally, the client can omit sending higher temporal layers completely (for the current GoP) if too many packets are dropped from those layers. For example, if the client detects that bursty packet loss has resulted in significant losses in multiple tid=2 and tid=3 packets, it can omit sending packets from these layers so the operator may reconstruct tid=0 and tid=1 packets at higher quality albeit at a lower fps.

3.4 Forward Error Correction

Recall that H264 video information is concentrated in I-frames and temporal layer 0 packets. Forward error correction is a method to protect packets by transmitting redundant data in anticipation of packet loss.[5] The n data packets are augmented with k FEC packets. The bitstream consisting of the $n + k$ packets is able to withstand loss (of certain packets) by reconstructing lost packets using the FEC data. Jalapeño has the option to enable the simplest FEC scheme. The I-frame or TID=0 packets are duplicated (see Figure 3.4). Hence, if one copy of the packet is lost, the other video packet can be decoded instead. Since network loss tends to be bursty, we send the FEC packets after the entire sequence of the original packets.

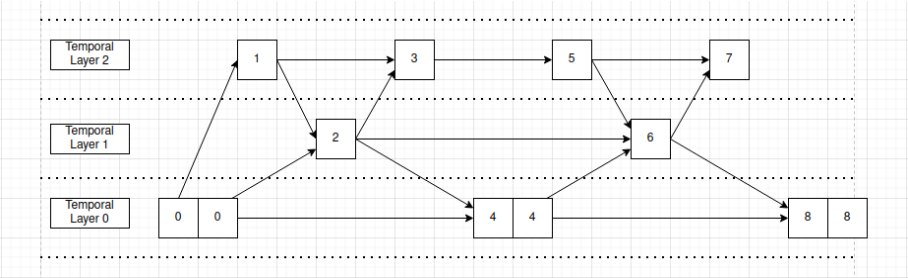

Figure 3.4: Visualization of H264 stream with the lowest (TID=0) layer being protected by FEC. The entire frame contents are duplicated.

[5]Correcting for bit-errors is not essential since the UDP checksum will detect bit-errors.

3.5 Buffer Reconstruction

Temporal layer 0 packets encompass the highest information in the video stream. Successive temporal layer 0 packets are 120 ms apart. The large delay between tid=0 packets enables the use of reconstruction algorithms that retransmit dropped packets to improve stream quality. Since the stream is real-time, the retransmitted packet is likely arriving at the decoder when the frame is already outdated. To make use of the retransmitted packet from a previous frame, Jalapeño uses a secondary OpenH264 decoder. The secondary decoder decodes all packets (including received retransmitted packets) from the previous TID=0 frame. Under successful reconstruction scenarios, the state of the decoder is now more updated than the original decoder as it contains information from the lost packet(s). The secondary decoder is decoding packets right before the current TID=0 frame to save computation.

The encoder chooses to retransmit a dropped packet when it receives the NACK for that packet and if the estimated packet arrival time of the retransmitted packet is before the next TID=0 frame. This is to ensure the decoder has enough time to decode all packets between two successive TID=0 frames.[6] See Figure 3.5 for a visualization of the buffer reconstruction algorithm.

[6]One can have a variation of the algorithm that allows retransmission with more leeway of estimated arrival time. However, the H264 decoder must be quick enough to decode all frames from the frame of the reconstructed packet to the frame of the next packet and still maintain real-time.

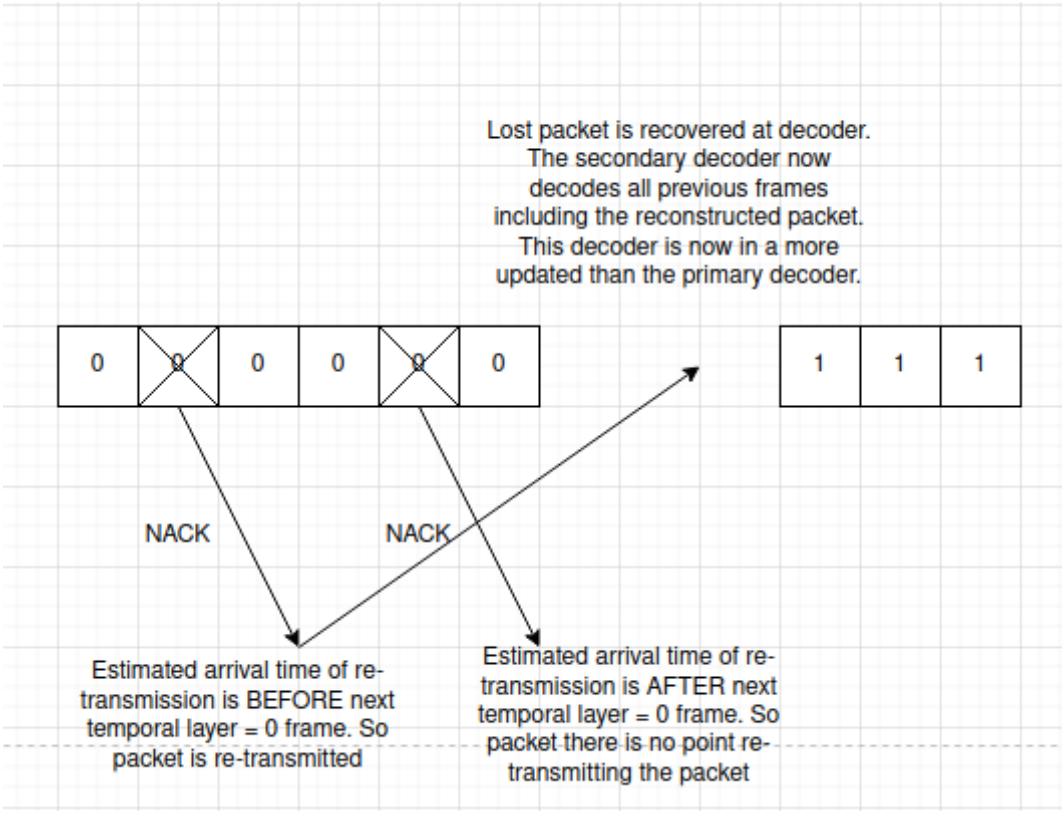

Figure 3.5: Visualization of Jalapeño's temporal buffer reconstruction algorithm. The crossed out packets denote lost TID=0 packets. The client, after receiving NACKs, decides whether to retransmit packets. The secondary decoder can now be updated to a better state after reconstruction.

CHAPTER 4

SYSTEM ARCHITECTURE

Minimizing system delays is crucial in I/O bound applications like Jalapeño's real-time streaming platform. As the platform scales, the computational resources required should not bottleneck the application. Jalapeño's runtime is built using Boost.Asio [7], a C++ library that offers the tools for efficient asynchronous operations. Naively spawning threads for each operation type will quickly exhaust system resources. Instead, Jalapeño's runtime preemptively spawns a fixed number of threads on startup that maximize the concurrency of the system.[1] The Boost.Asio runtime facilitates sending work operations, such as asynchronous networking operations or concurrent user functions, to the thread pool. The fixed thread pool always makes progress on the growing work queue (little idle time).[2]

Under the hood, Boost.Asio is using the operating system's event polling system calls (epoll on Linux and Kqueue on Windows) to efficiently poll sockets for completion of networking operations. Boost.Asio provides interfaces for TCP and UDP asynchronous networking calls with completion handlers being posted as work functions to the thread pool.

Boost.Asio also provides abstractions to simplify synchronization. Strands are used to send operations to the thread pool that will execute serially.[3] Strands effectively remove the need for explicit locks in the Jalapeño code-

[1]The number of threads spawned is a small multiple of the number of CPU cores available in the system.

[2]In order to remain efficient, the operations posted on the thread pool must be short for fairness. Additionally, the operations must nonblocking, otherwise the executing thread will become idle. Perhaps the biggest drawback of this approach is that the developer must be aware of the negative effect of long-running handlers and blocking operations on runtime performance.

[3]Operations on the same strand can be executed on different threads, but never concurrently.

base (each atomic operation can be posted as a separate handler in the thread pool on the same strand).

CHAPTER 5

VIDEO SUPER-RESOLUTION

Video super-resolution addresses the problem of converting low-resolution video frames into high-resolution ones. This process is directly analogous to typical, single image super-resolution, but offers the extra task of preserving temporal and motion consistency across the frames. It combines the challenges of a number of different generative applications including image generation, time-series generation, and super-resolution. As such, it offers many of the same motivations of these individual applications, especially within fields such as video surveillance, medical imaging, and more. For teleoperations, video super-resolution can aggressively reduce the bandwidth required for video transmission. Instead of transmitting the original video, the platform can stream a video of much lower resolution and upscale the bitstream at the decoder.

For video super-resolution to be viable in teleoperations, it must be performed in real-time on the decoder. Otherwise, the streamed video will need to be buffered before upscaling, and will no longer be a real-time representation of the roadway. The real-time generative constraint limits the complexity of the model that can be used.[1]

5.1 TecoGAN and EGVSR

Jalapeño uses the EGVSR [8] video super-resolution framework. The main points of the TecoGAN [9] and the closely related EGVSR model are summarized in this section. The TecoGAN generator produces a super-resolution output $g_t \in R^{3\times sh\times sw}$ from the input low-resolution frame $a_t \in R^{3\times h\times w}$ where

[1] All super-resolution experiments were performed on RTX 3080 graphics cards. For the purposes of this thesis, real-time performance implies a 30fps generative speed on the aforementioned GPU.

s is the scale-factor. The generator is a two-part CNN network composed of the FlowNet and the frame-recurrent generator. The FlowNet is a classic encoder-decoder CNN model used to predict the dense optical flow between two consecutive temporal frames $v_t = F(a_{t-1}, a_t)$. Next, the frame-recurrent generator takes the current low-resolution frame a_t and the previously generated high-resolution frame g_{t-1} warped forward in time using the computed optical flow to produce the current super-resolution frame $g_t = G(a_t, W(g_{t-1}, v_t))$. See Figure 5.1 for a visualization of the generator and discriminator of the model.

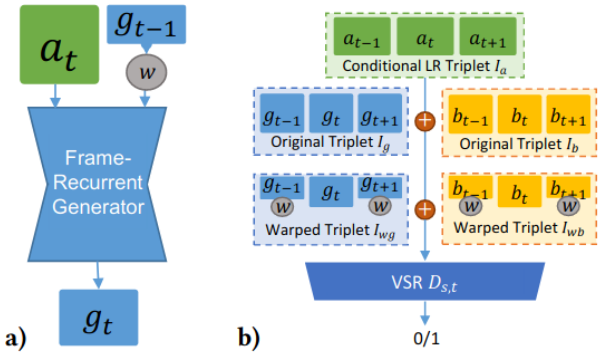

Fig. 4. a) The frame-recurrent VSR Generator. b) Conditional VSR $D_{s,t}$.

Figure 5.1: The generator and discriminator of EGVSR/TecoGAN.

The warped super-resolution input to the frame-recurrent generator $W(g_{t-1}, v_t)$ is reshaped $R^{3\times sh\times sw} \rightarrow R^{3s^2\times h\times w}$ and concatenated with the low-resolution input for a total input shape of $R^{3s^2+3\times h\times w}$. First, the input is passed through an initial convolutional layer with output feature f to produce $R^{f\times h\times w}$. From here, the generator employs a series of residual blocks. Finally, an upscaling block transforms the input into $R^{f/s^2\times sh\times sw}$ and a final convolutional layer compresses the feature map to produce $R^{3\times sh\times sw}$. The main difference between TecoGAN and EGVSR is the upscaling block used in the generator. TecoGAN uses two stacks of transposed convolutions for upscaling while EGVSR upscales via PixelShuffle layers which are computationally cheaper (as can be seen in Table 5.1) but slightly less accurate.

Table 5.1: Upsample runtime and performance comparison. (B) is ConvTranspose2d and (C) is Conv2d followed by PixelShuffle.

Up-sample Method	Total Param#	Train		Test		CPU time (ms)	GPU time (ms)
		Loss	PSNR	PSNR	SSIM		
A	29,409	0.0055	22.61	25.45	0.72	415.8	9.860
B	30,177	0.0048	23.20	26.52	0.76	253.4	8.203
C	29,673	0.0047	23.28	26.50	0.77	234.9	6.234

The discriminator takes in five sets of triplets: original super-resolution frames g_{t-1}, g_t, g_{t+1}, warped super-resolution frames $W(g_{t-1}), g_t, W'_{g+1}$ (where W' refers to warping backwards in time), original high-resolution frames b_{t-1}, b_t, b_{t+1}, warped high-resolution frames $W(b_{t-1}), b_t, W'_{b+1}$, and the conditional frames a_{t-1}, a_t, a_{t+1}. The architecture of the discriminator consists of four convolutional blocks followed by an MLP classifier which predicts whether the input triplet is real or fake. The architecture of the generator and discriminator can be seen in Figure 5.1.

Table 5.2: Breakdown of the different loss components in EGVSR

$\mathcal{L}_{D_{s,t}}$ for		
VSR, $D_{s,t}$	$-\mathbb{E}_{b\sim p_b(b)}[\log D(\mathrm{I}^b_{s,t})] - \mathbb{E}_{a\sim p_a(a)}[\log(1 - D(\mathrm{I}^g_{s,t}))]$	
UVT, $D^b_{s,t}$	$\mathrm{E}_{b\sim p(b)}[D(\mathrm{I}^b_{s,t}) - 1]^2 + \mathrm{E}_{a\sim p(a)}[D(\mathrm{I}^g_{s,t})]^2$	
Loss for	VSR, G & F	UVT, G_{ab}
$\mathcal{L}_{G,F}$	$\lambda_w \mathcal{L}_{\text{warp}} + \lambda_p \mathcal{L}_{\text{PP}} + \lambda_a \mathcal{L}_{\text{adv}} + \lambda_\phi \mathcal{L}_\phi + \lambda_c \mathcal{L}_{\text{content}}$	
$\mathcal{L}_{\text{warp}}$	$\sum \lVert a_t - W(a_{t-1}, \mathrm{F}(a_{t-1}, a_t)) \rVert_2$	
$\mathcal{L}_{\text{PP}}$	$\sum_{t=1}^{n-1} \lVert g_t - g_t{}' \rVert_2$	
$\mathcal{L}_{\text{adv}}$	$-\mathbb{E}_{a\sim p_a(a)}[\log D_{s,t}(\mathrm{I}^g_{s,t})]$	$-\mathbb{E}_{a\sim p_a(a)}[D^b_{s,t}(\mathrm{I}^{g^{a\to b}}_{s,t})]^2$
$\mathcal{L}_\phi$	$1.0 - \frac{\Phi(\mathrm{I}^g_{s,t}) * \Phi(\mathrm{I}^b_{s,t})}{\lVert\Phi(\mathrm{I}^g_{s,t})\rVert * \lVert\Phi(\mathrm{I}^b_{s,t})\rVert}$	$\left\lVert GM(\Phi(\mathrm{I}^g_{s,t})) - GM(\Phi(\mathrm{I}^b_{s,t}))\right\rVert_2$
$\mathcal{L}_{\text{content}}$	$\lVert g_t - b_t \rVert_2$	$\lVert g_t^{a\to b\to a} - a_t\rVert_2 + \lVert g_t^{b\to a\to b} - b_t\rVert_2$

The total loss used to optimize the GAN is a weighted sum of the DC-GAN loss, pixel loss, perceptual loss, ping-pong loss, and warp loss which

are given in Table 5.2. The most obvious component is the standard DC-GAN objective based on the classifier output of the discriminator. The warp loss minimizes the MSE error between warped frame and the actual frame. The pixel loss minimizes the MSE between the generated high-resolution and the ground-truth high-resolution frames. The perceptual loss minimizes the cosine distance between feature maps taken from a pretrained VGG-19 network. The final and most innovative component of the loss is the ping-pong loss. First, the input sequence of frames from $a \rightarrow b$ is appended with the reverse of the sequence to get the final sequence $a \rightarrow b \rightarrow a^{'}$ during the forward pass. Next, the MSE is minimized between a frame from the appended reverse sequence with its corresponding frame in the forward sequence. This process is visualized in Figure 5.2. The testing done by the authors of TecoGAN shows that the addition of the ping-pong objective enables training the network with much higher temporal sequences by reducing temporal inconsistencies.

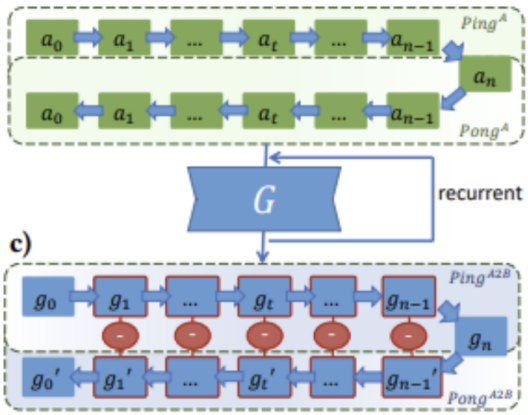

Figure 5.2: PP loss visualization from TecoGAN paper [9].

TecoGAN uses three classes of metrics to evaluate performance of the model. The first class measures the pixel distortion using PSNR and SSIM between corresponding super-resolution and high-resolution frames. The LPIPS metric captures perceptual/semantic similarities between frames by comparing feature maps from pretrained networks. Additionally, the tOF metric measures L1-loss between the optical flow between corresponding pairs of frames

in the input video $||OF(a_{t-1}, a_t) - OF(g_{t-1}, g_t)||_1$.

5.2 Model Adjustments

The pixel loss (mean squared error) between the original video sequence and the super-resolution video sequence is not a good metric of reconstruction quality. Small imperceptible changes in the frames can cause large swings in pixel loss. Jalapeño adds an additional SSIM loss to optimize. SSIM is a more perceptual metric that computes degradation in the reconstruction while taking into account spatial inter-dependencies in the frame. Jalapeño uses a differentiable multi-scale SSIM metric for optimization.

We experimented with replacing the additional DCGAN loss with the Wasserstein GAN metric with gradient penalty [10]. Instead of the discriminator outputting a sigmoid probability (optimized via cross entropy loss), the critic now outputs a score for the input sequence. The critic is updated for five iterations for every generator update. Figure 5.3 shows the algorithm for the WGAN-GP model. The gradient penalty is computed by evaluating the critic on an interpolated sequence between the reconstructed and original sequence.[2]

5.3 Dataset and Training

Large scale unsupervised video super-resolution models like EGVSR benefit from an enormous amount of training videos to achieve good performance. One of the shortcomings of the original models was training on a relatively small amount of training samples: 180 videos with roughly 100 frames each scraped from Vimeo. We outline a procedure to generate a video dataset for unsupervised video learning using the Youtube8M [11] dataset.

Using video IDs in the Youtube8M dataset, we construct a video dataset for

[2]The gradients of the grid sample operations used in the critic to warp frames based on computed optical flows are not differentiable. This is an open issue in PyTorch. We use a custom double differentiable grid sample operation that closely approximates the grid sample operations from PyTorch to compute the gradient penalty.

Algorithm 1 WGAN with gradient penalty. We use default values of $\lambda = 10$, $n_{\text{critic}} = 5$, $\alpha = 0.0001$, $\beta_1 = 0$, $\beta_2 = 0.9$.

Require: The gradient penalty coefficient λ, the number of critic iterations per generator iteration n_{critic}, the batch size m, Adam hyperparameters α, β_1, β_2.
Require: initial critic parameters w_0, initial generator parameters θ_0.
while θ has not converged **do**
  for $t = 1, ..., n_{\text{critic}}$ **do**
    for $i = 1, ..., m$ **do**
      Sample real data $\boldsymbol{x} \sim \mathbb{P}_r$, latent variable $\boldsymbol{z} \sim p(\boldsymbol{z})$, a random number $\epsilon \sim U[0, 1]$.
      $\tilde{\boldsymbol{x}} \leftarrow G_\theta(\boldsymbol{z})$
      $\hat{\boldsymbol{x}} \leftarrow \epsilon \boldsymbol{x} + (1 - \epsilon)\tilde{\boldsymbol{x}}$
      $L^{(i)} \leftarrow D_w(\tilde{\boldsymbol{x}}) - D_w(\boldsymbol{x}) + \lambda(\|\nabla_{\hat{\boldsymbol{x}}} D_w(\hat{\boldsymbol{x}})\|_2 - 1)^2$
    end for
    $w \leftarrow \text{Adam}(\nabla_w \frac{1}{m} \sum_{i=1}^{m} L^{(i)}, w, \alpha, \beta_1, \beta_2)$
  end for
  Sample a batch of latent variables $\{\boldsymbol{z}^{(i)}\}_{i=1}^{m} \sim p(\boldsymbol{z})$.
  $\theta \leftarrow \text{Adam}(\nabla_\theta \frac{1}{m} \sum_{i=1}^{m} -D_w(G_\theta(\boldsymbol{z})), \theta, \alpha, \beta_1, \beta_2)$
end while

Figure 5.3: WGAN-GP training algorithm taken from the paper [10]. Instead of sampling latent space for generating fake samples, VSR models use the low-resolution sequence to generate the super-resolution frames.

unsupervised learning with $\sim$ 165k samples with roughly 30 frames each.[3] Additionally, the videos in this newly constructed dataset are all much higher resolution than the original Vimeo dataset. Partitions of the Youtube8M dataset contain unique IDs for each video. These IDs can be translated to Youtube URLs via an API lookup (https://data.yt8m.org). Finally, the Youtube URL can be used to scrape the videos. We used a cluster of AWS instances to scrape the Youtube videos and store them in a bucket. During training, the training videos are randomly cropped to a fixed size (128×128 or 192×192) spatial resolution with random snippets of 10 frames. We also apply random flipping and rotational transforms. A batch of these videos is used for training the EGVSR model.

The training for the EGVSR model happens in two phases. In the first phase, the generator is pretrained by only optimizing the pixel, warp loss, feature loss, and the SSIM loss. With the generator having a head-start, we start additionally optimizing the W-GAN objective.[4] We compare this to the original EGVSR model with the second stage training on the DCGAN

[3]The Youtube8M dataset contains 6.1 millions video IDs. A much larger dataset for unsupervised video learning can be constructed using this approach.

[4][12] discusses how unbalanced GAN training with a pretrained generator outperforms balanced training. For our case, we can pretrain generator without a variational autoencoder.

objective. We evaluate the model on a subset of videos from the Youtube8M dataset for testing. Figures 5.4-5.11 show the various training losses and testing metrics. Figures 5.12-5.15 show example super-resolution outputs from the model. Finally Table 5.3 compares the performance of the EGVSR model vs. a bicubic upscaling approach.

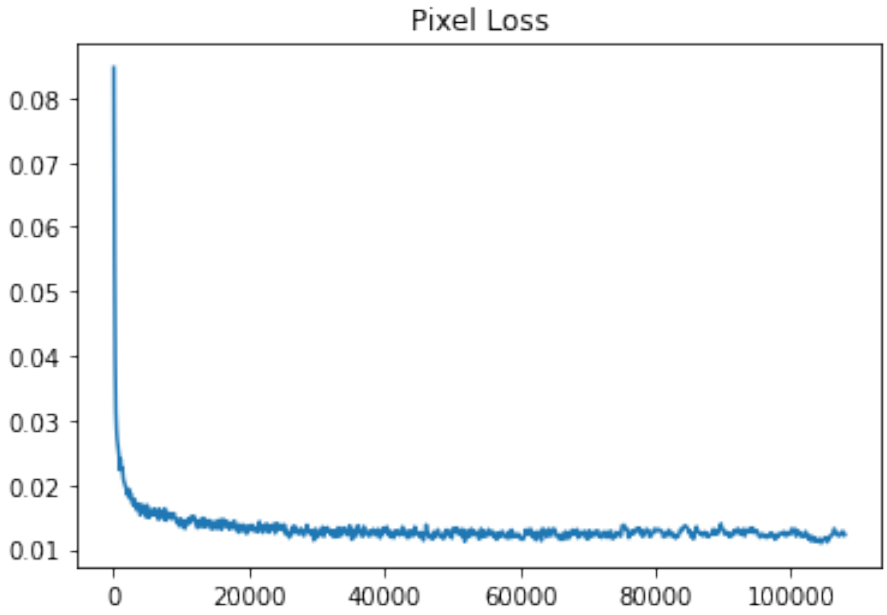

Figure 5.4: Pixel loss vs. training iterations.

Table 5.3: EGVSR performance vs. standard bicubic upscaling

	PSNR	LPIPS	tOF	SSIM
Bicubic	24.033	0.209	0.696	0.935
EGVSR	28.075	0.175	0.549	0.983

5.4 Compression-Aware Super-Resolution

A big shortcoming of video super-resolution in lossy H264 streams is that the frame artifacts caused by lost packets or compression artifacts are enhanced

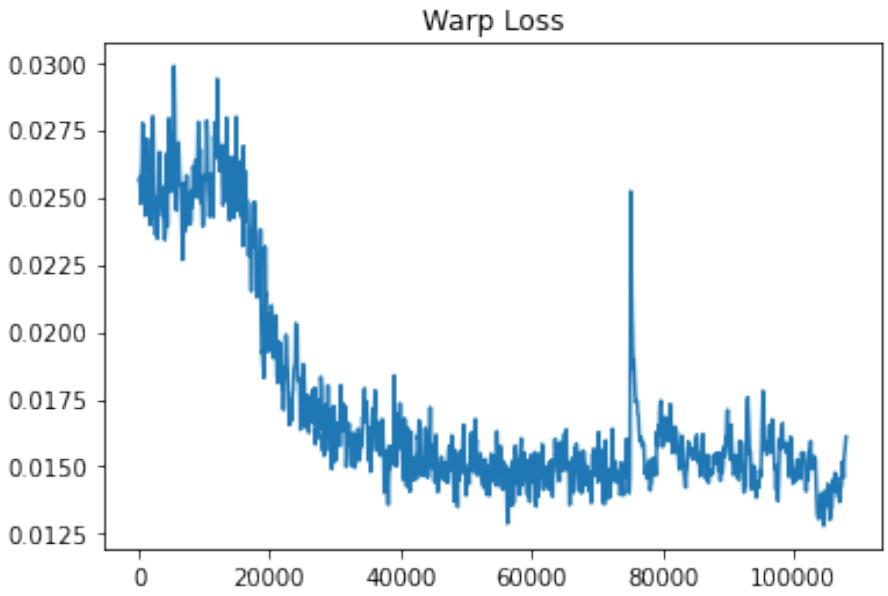

Figure 5.5: Warp loss vs. training iterations.

by the model. To overcome this issue, we attempt to train the model on compressed videos with lost packets. We construct a dataset of lossy low-resolution frames $\hat{a}_t$ by randomly dropping $0 - 10\%$ H264 packets from the low-resolution frames a_t. The generator now attempts to reconstruct the super-resolution frames g_t using the lossy low-resolution frames $\hat{a}_t$.

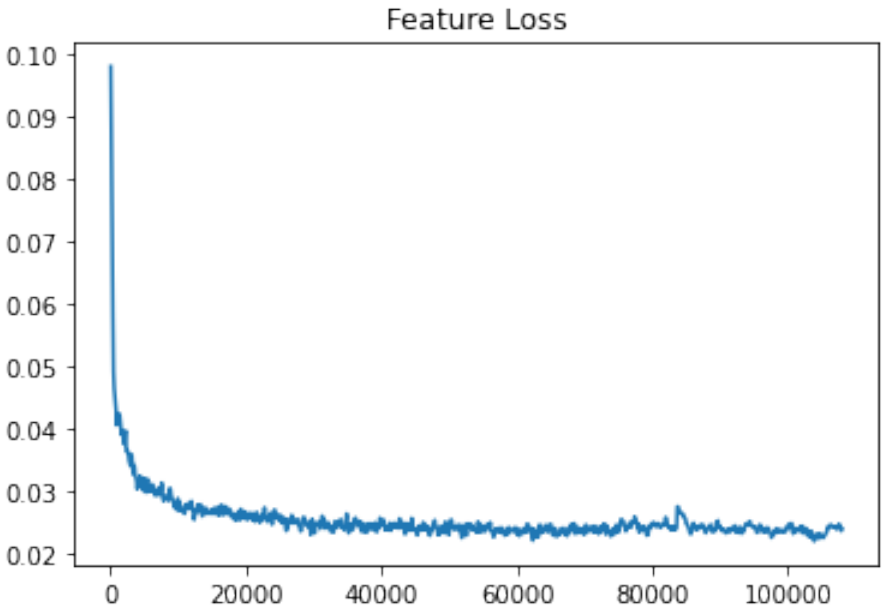

Figure 5.6: Feature loss vs. training iterations.

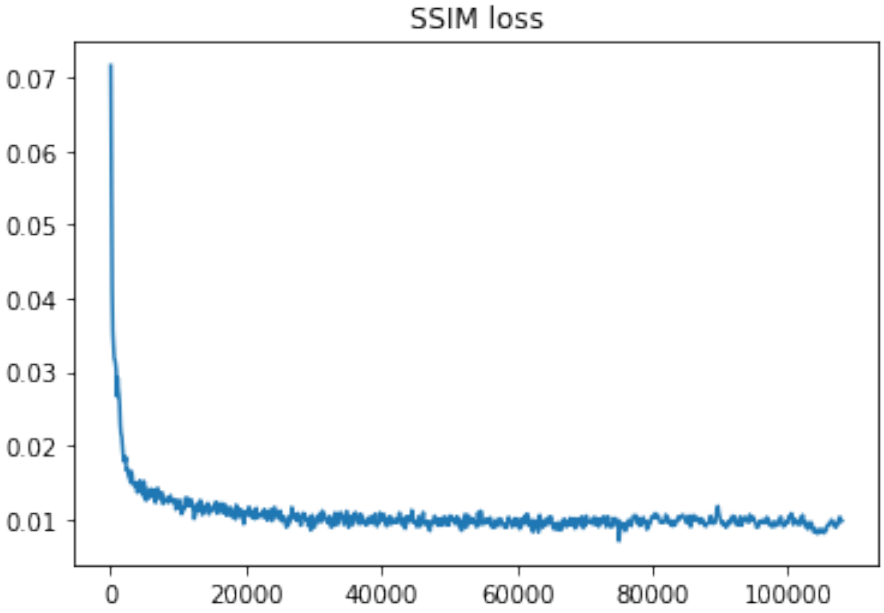

Figure 5.7: SSIM loss vs. training iterations.

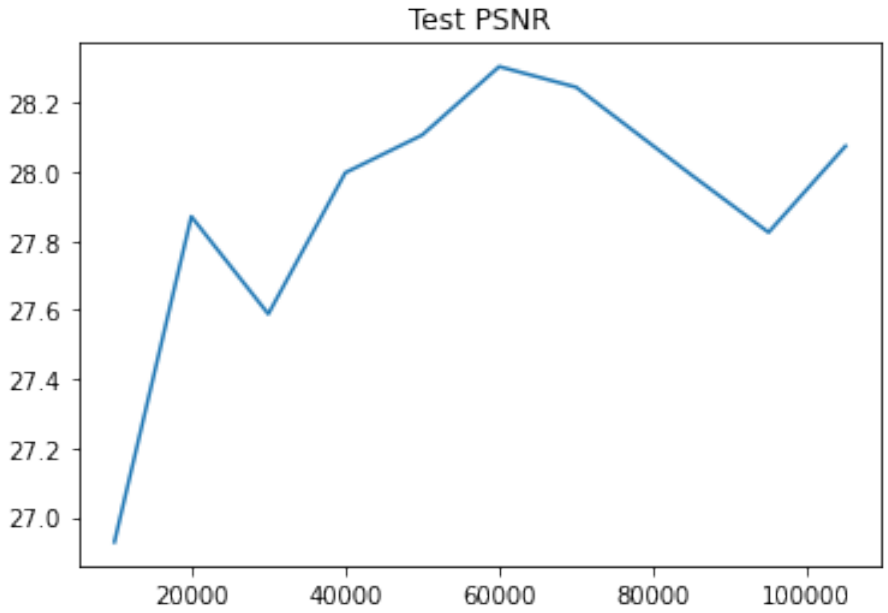

Figure 5.8: Testing PSNR (higher is better).

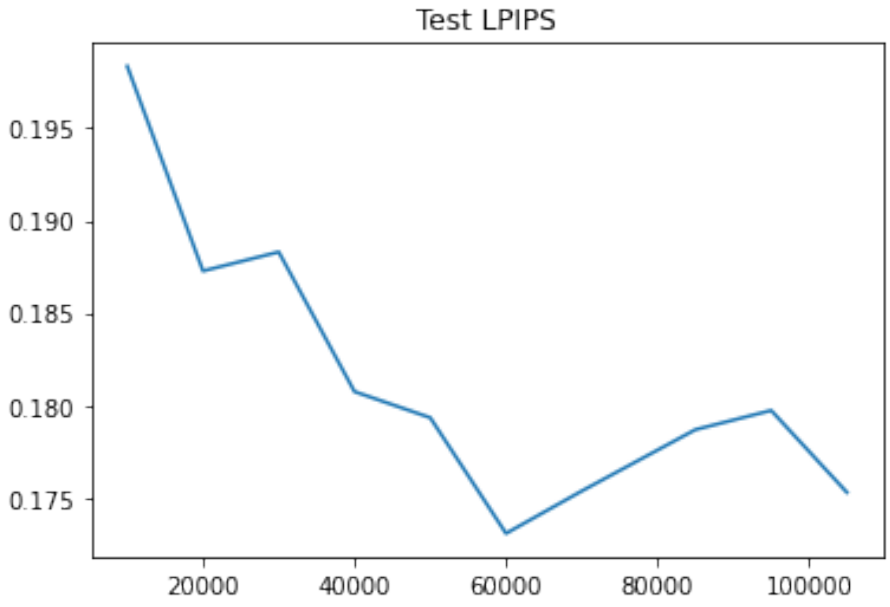

Figure 5.9: Testing LPIPS (lower is better).

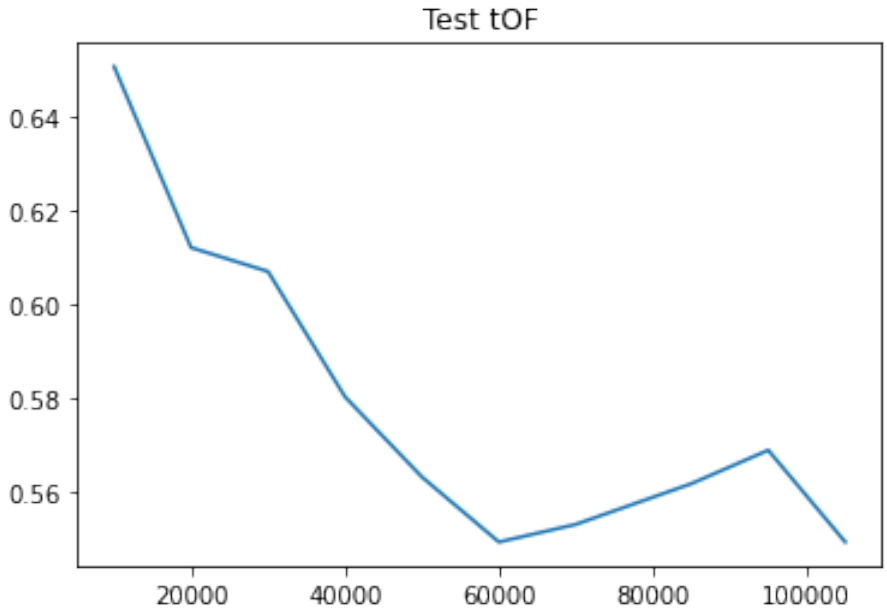

Figure 5.10: Testing tOF (lower is better).

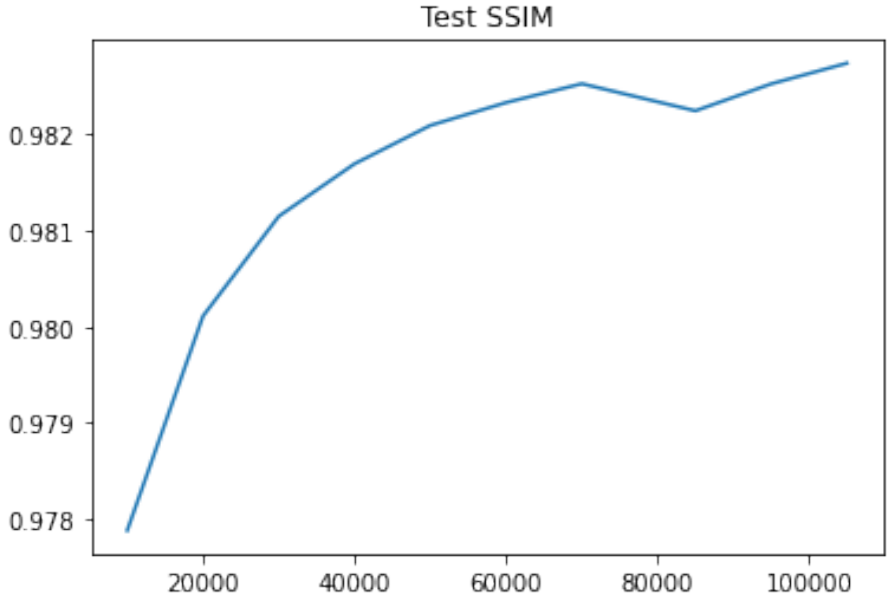

Figure 5.11: Testing SSIM (higher is better).

Figure 5.12: EGVSR example. Top to bottom: ground truth frame, video super-resolution frame, and bicubic upsample frame.

Figure 5.13: EGVSR example. Top to bottom: ground truth frame, video super-resolution frame, and bicubic upsample frame.

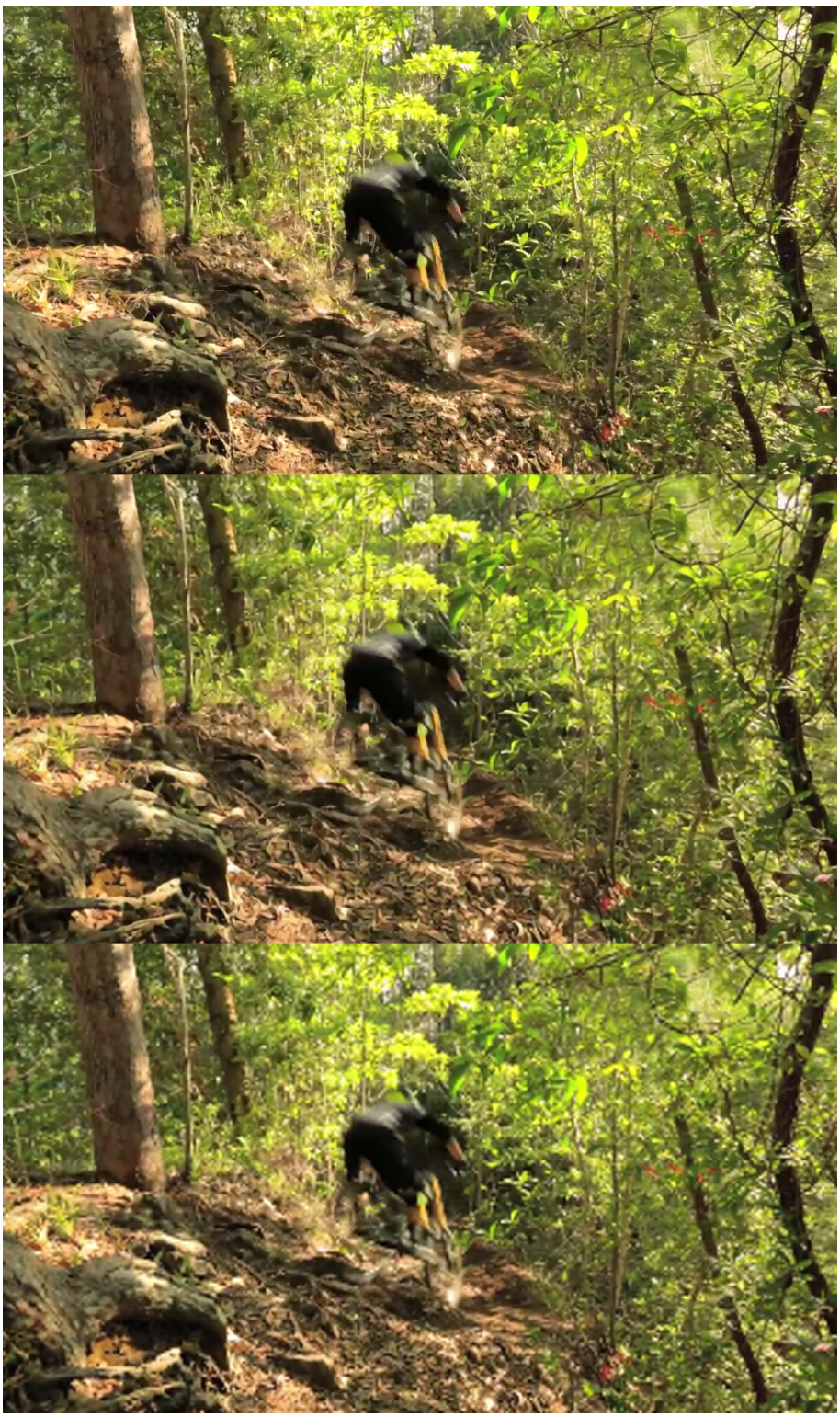

Figure 5.14: EGVSR example. Top to bottom: ground truth frame, video super-resolution frame, and bicubic upsample frame.

Figure 5.15: EGVSR example. Top to bottom: ground truth frame, video super-resolution frame, and bicubic upsample frame.

CHAPTER 6

CONCLUSION

The Jalapeño video streaming platform is optimized for real-time teleoperations. The combination of a standard H264 video compression that is adjusted with a networking stack that is constantly probing the state of the network differentiates Jalapeño from other solutions. Ideas such as the buffer reconstruction algorithm that leverage temporal SVC features in H264 are exciting for future work. Finally, the real-time video super-resolution takes the platform to the next level. Jalapeño can stream a low-resolution video with much less bandwidth. The decoder can then upscale the video in real-time without significant loss in quality. With some more experimentation, compression-aware super-resolution can simultaneously upscale the stream and alleviate artifacts caused by compression and packet-loss. Jalapeño combines all these techniques to deliver a real-time video streaming solution for applications such as teleoperations and beyond.

REFERENCES

[1] J.-W. Chen, C.-Y. Kao, and Y.-L. Lin, "Introduction to h.264 advanced video coding," in *Proceedings of the 2006 Asia and South Pacific Design and Automation Conference*, Jan. 2006, pp. 736–741.

[2] Cisco, "OpenH264 codec library." [Online]. Available: https://github.com/cisco/openh264

[3] P.-L. Ageneau, "Libjuice - UDP interactive connectivity establishment." [Online]. Available: https://github.com/paullouisageneau/libjuice

[4] H. Schulzrinne, S. L. Casner, R. Frederick, and V. Jacobson, "RTP: A transport protocol for real-time applications," RFC 3550, July 2003. [Online]. Available: https://www.rfc-editor.org/info/rfc3550

[5] R. Jesup, T. Kristensen, Y. Wang, and R. Even, "RTP payload format for H.264 video," RFC 6184, May 2011. [Online]. Available: https://www.rfc-editor.org/info/rfc6184

[6] M. Sharabayko, M. Sharabayko, J. Dube, J. Kim, and J. Kim, "The SRT Protocol," Internet Engineering Task Force, Internet-Draft draft-sharabayko-mops-srt-00, work in Progress. [Online]. Available: https://datatracker.ietf.org/doc/html/draft-sharabayko-mops-srt-00

[7] C. M. Kohlhoff, "Boost.asio cross-platform C++ library." [Online]. Available: https://www.boost.org/doc/libs/1_78_0/doc/html/boost_asio.html

[8] Y. Cao, C. Wang, C. Song, Y. Tang, and H. Li, "Real-time super-resolution system of 4k-video based on deep learning," *CoRR*, vol. abs/2107.05307, 2021. [Online]. Available: https://arxiv.org/abs/2107.05307

[9] M. Chu, Y. Xie, L. Leal-Taixé, and N. Thuerey, "Temporally coherent gans for video super-resolution (TecoGAN)," *CoRR*, vol. abs/1811.09393, 2018. [Online]. Available: http://arxiv.org/abs/1811.09393

[10] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. C. Courville, "Improved training of Wasserstein GANs," *CoRR*, vol. abs/1704.00028, 2017. [Online]. Available: http://arxiv.org/abs/1704.00028

[11] S. Abu-El-Haija, N. Kothari, J. Lee, P. Natsev, G. Toderici, B. Varadarajan, and S. Vijayanarasimhan, "Youtube-8m: A large-scale video classification benchmark." *CoRR*, vol. abs/1609.08675, 2016. [Online]. Available: http://dblp.uni-trier.de/db/journals/corr/corr1609.html#Abu-El-HaijaKLN16

[12] H. Ham, T. J. Jun, and D. Kim, "Unbalanced GANs: Pre-training the generator of generative adversarial network using variational autoencoder," *CoRR*, vol. abs/2002.02112, 2020. [Online]. Available: https://arxiv.org/abs/2002.02112