International Journal of Semantic Computing

Accelerated Bitrate Ladder Creation for Video Live Streaming using Temporal Layer Injection

Hannes Mareen, Casper Haems, Tim Wauters,
Filip De Turck, Peter Lambert, and Glenn Van Wallendael

Ghent University imec, IDLab, Department of Electronics and Information Systems,
Technologiepark-Zwijnaarde 122, 9052 Gent, Belgium
firstname.lastname@ugent.be
http://media.idlab.ugent.be

Received (Day Month Year)
Revised (Day Month Year)
Accepted (Day Month Year)

Video streaming systems aim to provide high-quality video adapted to clients' device and network conditions. For this purpose, adaptive streaming architectures encode video content at a variety of quality levels, organized in a bitrate ladder. However, compressing a video into multiple streams is resource-intensive, which may become especially problematic in live streaming applications with real-time demands. Therefore, this paper proposes a novel solution for fast bitrate ladder creation, and provides the requirements for implementation in the H.266/VVC standard. More specifically, the proposed method creates new intermediate Combined Streams by injecting the lowest temporal layers of a higher-quality Augmentation Stream in a lower-quality Base Stream. Since the lowest layers are used as reference by the remaining layers, this procedure indirectly increases the quality of the frames in those untouched remaining layers as well. Although altering the reference frames may cause some drift-error artifacts, we demonstrate that, on average, the quality of the intermediate streams are higher than the base stream. Additionally, we demonstrate that injecting more layers brings both the quality and bitrate closer to that of the Augmentation Stream. The disadvantage of the Combined Streams is that their quality fluctuates more than the quality of the source streams, and that they are compressed less efficiently, comparable to going from a *slower* to *fast* or *faster* preset in the VVenC encoder. Most importantly, their main advantage is that they were generated at no significant additional computational complexity. In this way, the proposed method is of great benefit when generating a bitrate ladder of video streams under constrained computational resources. The code has been made open source and is available on `https://github.com/IDLabMedia/NALUProcessing`.

Keywords: Adaptive Streaming; Bitrate Ladder; Live Streaming; Versatile Video Coding (VVC).

1. Introduction

Video streaming providers aim to provide high-quality content to their clients, while making efficient use of the clients' device and network conditions. This aspiration is accompanied by the challenge of encoding videos to various quality levels, which

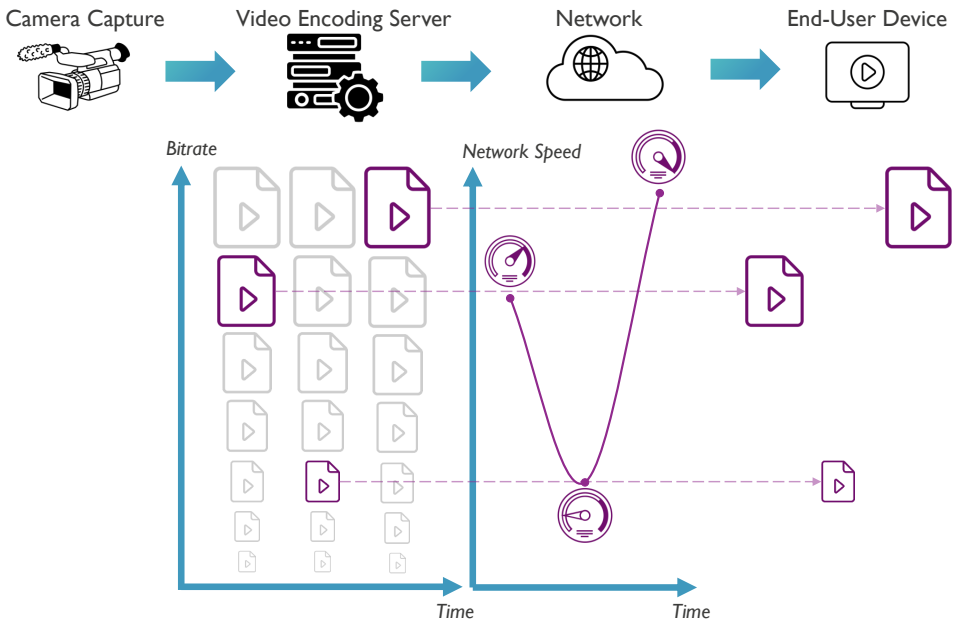


Fig. 1. Adaptive Bitrate Streaming diagram. The client adaptively chooses a video stream segment with the best matching bitrate to their current network conditions. For this reason, the video encoding server creates a bitrate ladder of video streams at various quality and bitrate levels. Separately encoding many streams is computationally expensive and may therefore not be feasible in live streaming scenarios with limited resources.

is resource-intensive. This challenge becomes even more pronounced in the context of live streaming, where real-time demands further amplify the complexity of maintaining quality while managing computational resources.

A commonly-used approach to provide users with tailored video streams is adaptive bitrate streaming, in which viewers switch between different quality streams at regular intervals. This is illustrated in Fig. 1. Clients adapt the chosen stream based on their network conditions. For this reason, the streaming provider needs to encode a so-called bitrate ladder of video streams. More steps on the bitrate ladder enable more efficient usage of the clients' available bandwidth, but the required computational resources scale linearly with the number of encoded streams. As such, encoding a bitrate ladder of streams may not be feasible in live streaming scenarios with limited resources. Additionally, in some video sharing platforms, only a small portion of the video library is highly viewed. In contrast, a large portion of the videos are viewed infrequently. Spending a large amount of computational resources on encoding these infrequently watched videos is undesired.

An approach to reduce the encoding complexity is using faster presets or configuration sets. For example, the *x264* [1] and *x265* [2] encoders provide 10 presets with various computational complexities and compression efficiency levels (i.e., ultrafast,

superfast, veryfast, faster, fast, medium, slow, slower, veryslow, and placebo). Faster presets decrease computational complexity by limiting the search space of coding decisions and disabling complex coding tools. However, this comes at the cost of a decrease in compression efficiency.

Similarly, older or faster coding standards can be used to reduce the encoding complexity. For example, H.264/AVC is much faster at encoding than H.265/HEVC. However, older coding standards are typically less efficient at compression. Additionally, using multiple coding standards is a disadvantage as the corresponding decoders need to be supported at the client device. Hence, there is a need for yet another alternative solution in which multiple quality levels can be provided with a low computational complexity.

This paper proposes a novel solution for the fast generation of intermediate steps in a bitrate ladder of video streams. The target use-cases are video live streaming and other video streaming scenarios with constrained computational resources. The proposed approach is called temporal layer injection, in which the lowest temporal layer(s) of a base stream are replaced by those of a higher-quality augmentation stream. As such, one can create new combined streams by splicing temporal layers of two source streams. When the source streams contain 6 temporal layers, one can create up to 5 intermediate combined streams with increasing bitrates. Example crops of the source streams and combined streams are given in Fig. 2, which showcases the increase in quality by gradually injecting more temporal layers.

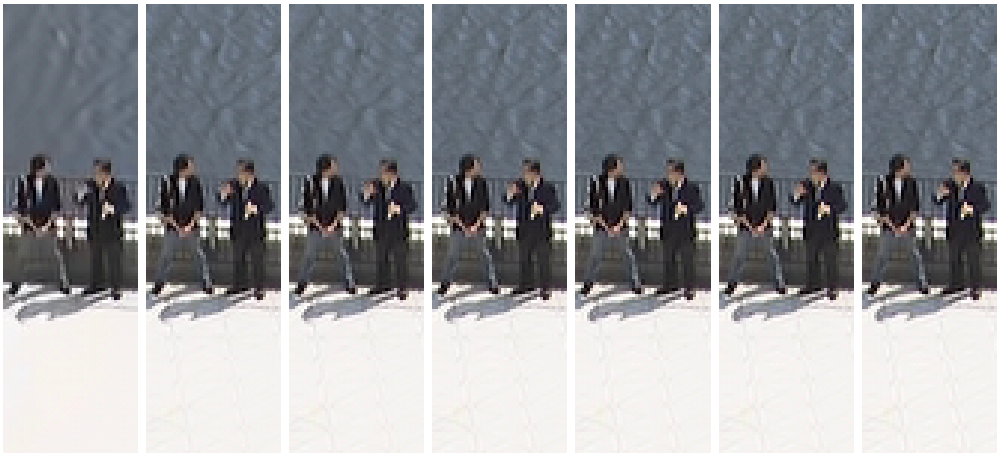
The contributions of this paper are the following. First, the novel concept of temporal layer injection for fast bitrate ladder creation is presented for the first time. Second, the requirements on implementing the proposed strategy in the H.266/VVC coding standard are listed and motivated. Third, the source code and its documentation has been made available for reproducibility[a]. Fourth, an in-depth analysis on the intermediate streams is performed in Section 4.

This paper is an extension of our previous work [3]. New material includes a more extensive and illustrated description of the problem statement and proposed method (Section 1 and Section 3, respectively), the open source code and its documentation (Section 3.3), as well as more experimental results. More specifically, we included the results of more quality levels combinations (Section 4.6), more presets (Section 4.7), and more quality (distortion) metrics (Section 4.8).

2. Related Work

When constructing a bitrate ladder, the first aspect to consider is which parameters to use for encoding the different steps of the ladder. Such aspects entail the resolution, the bitrates, the profile, and so on. Choosing these parameters efficiently can be done using bitrate ladder estimation algorithms [4–7].

[a]Code available on
`https://github.com/IDLabMedia/NALUProcessing`



(a) Base Stream (QP_B=32) (b) Combined Stream (tid=0) (c) Combined Stream (tid=1) (d) Combined Stream (tid=2) (e) Combined Stream (tid=3) (f) Combined Stream (tid=4) (g) Augmentation Stream (QP_A=22)

Fig. 2. The proposed method combines a (a) lower-quality Base Stream and (g) higher-quality Augmentation Stream to create five intermediate Combined Streams (b)-(f) by replacing the temporal layers from the Base Stream with those of the Augmentation Stream (which we call temporal layer injection). By injecting more layers, more detail is added in the video (most prominent in the waves of the river, the tile borders, and the heads of the people). Each crop has a resolution of 45×150 pixels, and is enlarged in this figure to make it easier to spot the subtle differences in detail. More visual examples can be found at `https://media.idlab.ugent.be/temporal-layer-injection`.

Irrespective of how efficiently the parameters for the bitrate ladder are estimated, on top of that, efficient multirate encoding needs to take place [8–11]. The representations that are deemed most efficient by the estimation algorithm need to actually be encoded. Efficiently performing multirate encoding can be performed by reusing redundant steps in a video encoder such as a computationally heavy analysis step [8]. Alternatively, the high-quality representation could utilize the encoding decisions taken in the low-quality representation as a starting point for prediction [9,11]. Afterwards, both the high-quality and low-quality representations can be used to accelerate the encoding of the intermediate representations [9].

A lot of the concepts used in multirate encoding originate from the concepts of transcoding [12–14]. Although the focus of transcoding is on processing complexity reduction, all these techniques still require a lot of processing. With complexity reductions around 82% compared to reference encoders [14], this processing remains excessively expensive.

Drastically reducing complexity can only be obtained by techniques that fall under the umbrella of network-distributed video coding [15, 16]. Such techniques are capable of reducing encoding complexity with 99.2% by using coding information calculation (CIC) and residual encoder (RE) modules [15]. First, the high-

complexity CIC modules calculate coding information for the video at certain bitrates. Then, the low-complexity RE modules use the information from the CIC modules to skip all encoding steps of a traditional encoder, except for the encoding of the residual. In this way, only the low-complexity entropy decoding and entropy encoding steps from the video coder are required to generate new streams.

A totally different approach to provide multiple streams from a single encoding is to utilize the temporal scalability that is supported by modern video standards. More specifically, a stream is encoded using temporal layers, where each temporal layer only uses frames from the same layer or below as reference, as is visualized in Fig. 4. As such, the upper temporal layer(s) can be dropped without breaking the inter-frame dependencies of the remaining layers. In this way, one can create new, lower-bitrate version(s) of a video stream with a lower temporal resolution or framerate. However, in typical adaptive streaming scenarios, we desire the bitrate ladder to vary more in spatial quality levels than in the temporal dimension. Hence, this approach is only practical to scale down video streams with high framerates. For clarity, our proposed approach called temporal layer injection also utilizes the temporal hierarchy in which coded videos are structured, but it does not adapt the framerate.

An alternative approach for tailored video streaming is using scalable video encoding. Using scalable coding, the video quality can be enhanced by adding higher-quality augmentation layers on top of a lower-quality base stream. This strategy was used in the Scalable Video Coding (SVC) [17] as an annex of H.264/Advanced Video Coding (AVC), and in Scalable High efficiency Video Coding (SHVC) [18] as an extension of H.255/High Efficiency Video Coding (HEVC), and in the Multilayer Main 10 profile in H.266/Versatile Video Coding (VVC) [19]. However, there is no market adoption of decoders with support for these scalable extensions or profiles. Additionally, the required computational resources still scale linearly with the number of augmentation layers.

The optimum in low-complexity methods that process video stream is the concept of frame injection. Although these methods were not created nor ever used for bitrate ladder creation, we use them as inspiration for the proposed method in this paper. Therefore, we briefly discuss their existing applications. Farber *et al.* first introduced this concept using the H.263 standard [20], and later extended it using S-frames [21]. These frames solve error propagation caused by frame injection at the cost of bitrate overhead. Later, Boyce and Tourapis presented the frame-injection concept for H.264/AVC [22], and Jennehag and Pettersson further analyzed this method [23]. Nowadays, frame injection is used specifically with keyframes in HESP [24–26]. Using modern compression standards, this concept has been extended to mixed-resolution keyframe injection in the H.266/VVC standard [27]. Finally, keyframe injection has also been demonstrated to work for quality enhancement in adaptive streaming use cases [28]. As such, frame injection proved to be a valuable method with an extremely low complexity. Therefore, it is used as inspiration for the temporal layer injection method proposed in Section 3.

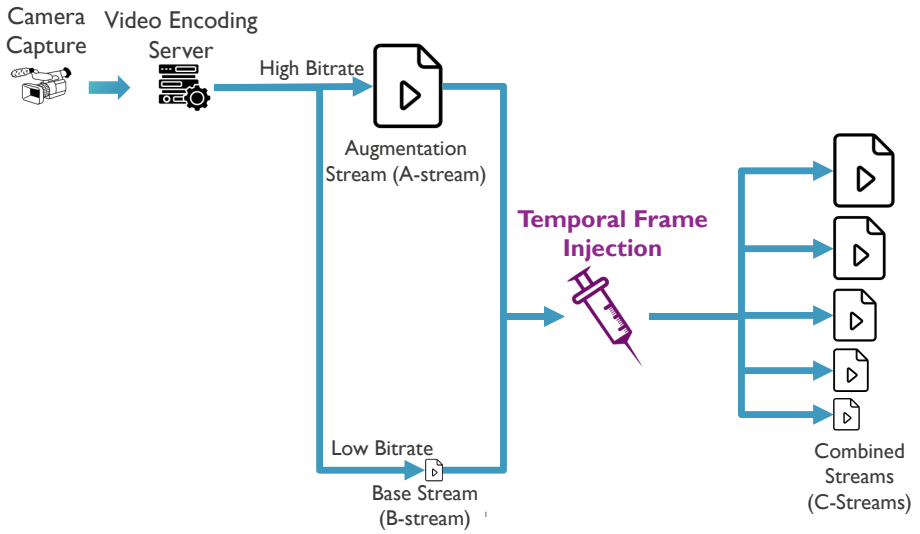


Fig. 3. Temporal Layer Injection diagram.

3. Proposed Method

This section first discusses the general concept of temporal layer injection in Section 3.1. Afterwards, Section 3.2 discusses how to enable standard-compliant injection using the H.266/VVC compression standard. Finally, Section 3.3 provides some documentation and a reference to the open source code of our framework.

3.1. *Temporal Layer Injection*

A high-level diagram of our proposed method is given in Fig. 3. Our proposed method requires two source streams as input, named the Base Stream (B-stream) and Augmentation Stream (A-stream). As output, these source streams are combined into multiple intermediate Combined Streams (C-streams). In this work, the B-stream is a stream on the lower-quality end of the bitrate ladder, whereas the A-stream corresponds to a higher-quality version of the same content, resolution, and Group of Pictures (GOP) structure.

We propose to inject temporal layers from the A-stream into the B-stream, as such creating new intermediate C-streams. In other words, temporal layers from the B-stream are replaced by those of the A-stream. More specifically, the lowest temporal layers of the A-stream are combined with the highest temporal layers of the B-stream (together covering all temporal layers). In practice, this means that the C-stream contains high-quality Intra frames (I-frames) and Predicted frames (P-frames) from the A-stream, interleaved with low-quality Bipredicted frames (B-frames) from the B-stream. This procedure is illustrated in Fig. 4, in which we inject the first three temporal layers (i.e., those with temporal layer id (tid) ≤ 2) of the

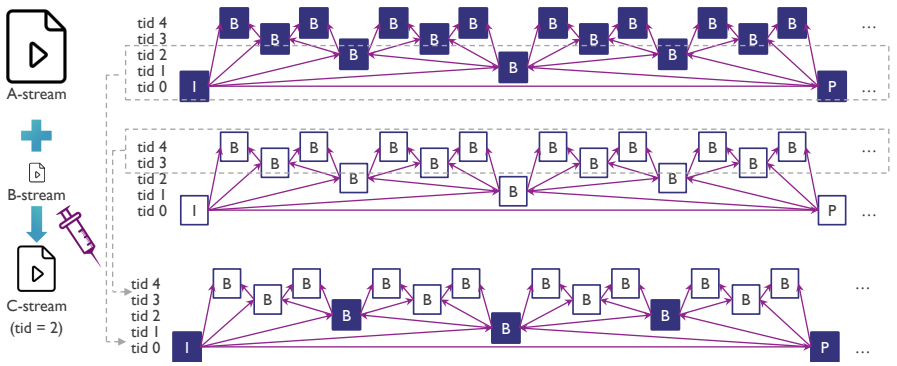


Fig. 4. Lower temporal layers of the Augmentation Stream (A), having tid smaller than or equal to 2, combined with higher layers of the Base Stream (B), having tid larger than 2, results in the Combined Stream (C). In this example, the GOP-size is 16 with five temporal layers.

A-stream into the B-stream, in order to create a C-stream. As such, the quality and bitrate of the C-stream is between those of the B-stream and A-stream.

Multiple intermediate C-streams can be created from a single B- and A-stream by injecting a different number of temporal layers. Therefore, when mentioning a C-stream, we will additionally mention the tid which specifies layer that splits the source streams. For example, the intermediate C-stream (tid = 0) is created by injecting only the lowest layer with tid = 0 of the A-stream into the B-stream. A second C-stream (tid = 1) injects the lowest two layers with tid $\leq$ 1. A third C-stream (tid = 2) injects the lowest three layers with tid $\leq$ 2. And so on. Note that the framerate of all streams is equal, i.e., temporal layers are replaced, and not completely deleted. That is why the number of frames in the A-, B-, and C-stream is equal. In general, when the source streams consist of l temporal layers, we can create up to $l - 1$ C-streams.

It should be stressed that the injection of temporal layers can be done with no significant computational overhead. That is because the procedure comes down to splicing Network Abstraction Layer (NAL) packets from the B- and A-stream. More specifically, the binary NAL Units (NALUs) corresponding to the frames of the selected temporal layers are removed from the B-stream, and are replaced by the corresponding NALUs from the A-stream, forming the C-stream. This operation can be performed purely on the binary-encoded streams, parsing only the NAL header information. No entropy decoding or other complex operations are needed to perform these operations. Hence, the proposed solution can greatly benefit applications with real-time demands.

3.2. *Requirements in H.266/VVC*

When performing temporal layer injection in H.266/VVC, one should enforce that the parameter sets of the B-stream and A-stream are compatible. Parameter sets are NALUs containing contextual information of the video. The H.266/VVC standard uses the following parameter sets: Video Parameter Set (VPS), Sequence Parameter Set (SPS), Picture Parameter Set (PPS), and Adaptation Parameter Sets (APS) [19]. This section discusses the possible strategies to comply with this requirement of compatible parameter sets.

In general, there are two main ways to enforce packet set compatibility. First, identical parameter sets could be forced during encoding of the source streams. Second, the parameter sets could be additionally injected during temporal layer injection. The remainder of this section discusses how these strategies can be applied for the different parameter set types.

For the VPS, SPS, and PPS types, identical packets in the source streams can be obtained with reasonably simple modifications to a traditional video encoder. That is, the initialization QP (pps_init_qp_minus26) specified in the PPS should be set to identical values in the source streams (e.g., fixed to 0), whereas traditional encoders typically adapt this value to the QP or Constant Rate Factor (CRF) specified as an input quality setting. As this is a minor encoder modification, we advise this strategy. If, for some reason, there is another difference between the sets of both streams, then an alternative solution is additionally injecting these parameter sets. More specifically, each picture should be preceded with the parameter sets of the picture's corresponding source stream. Hence, the parameter sets of the A-stream should be additionally injected before a range of injected frame packets, and the parameter sets of the B-stream should be additionally injected after these injected packets. As such, VPS, SPS and PPS compatibility is ensured.

For APS packets, the situation is more complicated than for VPS, SPS and PPS packets. That is because forcing the encoders to use identical APS packets would result in compression efficiency loss. Therefore, we recommend the alternative strategy to additionally injecting the APS packets during the temporal layer injection process. To do this, one needs to take the following into account. An APS packet provides parameters for one of three types of compression tools (specified by aps_params_type), namely Adaptive Loop Filter (ALF), luma mapping with chroma scaling (LMCS), and scaling list. For each of these tool types, APS packets can be addressed with a unique identification (ID) number (aps_adaptation_parameter_set_id). By referring to this ID, APS information can be reused by succeeding frames in the video stream. When the APS information with a certain ID is deemed irrelevant by the encoder for future frames, the APS information can be overwritten by reusing the ID. Therefore, depending on the position in the video stream, identical IDs can correspond to different informational content.

To facilitate this sequential overwriting of information throughout the stream, dictionaries of APS packets need to be retained and updated, which can be done during the proposed temporal layer injection process. For both source videos and for each of the three involved tools, such a dictionary is built in which the key values are represented by the APS ID. In other words, when parsing the source streams, every time an APS packet is encountered, it should be stored in the corresponding dictionary. When an injection of an augmentation picture takes place, first, the APS packets corresponding to the inserted picture need to be inserted in their corresponding dictionary. Then, the up-to-date dictionaries of APS packets from the A-stream need to be inserted. If immediately succeeding frames are additionally injected from the A-stream, then no further APS packet injection needs to be performed. However, note that it is still necessary to update the APS dictionaries further, though. When switching back to frames from the B-stream, again the APS packets of the B-stream frame need to be inserted in their corresponding dictionaries, and all dictionaries from the B-stream need to be inserted in front of the B-stream frame(s). In this way, it can always be guaranteed that all frames have up to date information on their APS parameters. As such, one can implement the proposed temporal layer injection method without breaking parameter set compatibility.

3.3. *Open Source Code Documentation*

The source code of the proposed work can be found on `https://github.com/IDLabMedia/NALUProcessing`. The software provides two different ways of working. Using the first way, called *NALUProcessing*, it can replace a single burst of NALU packets from one stream (inject) in the other (source) (useful for related work [25–27]). Using the second way of working, called *StreamProcessing*, it can be provided with a list of packets that need to be replaced in the source stream, originating from the inject stream. The second way of working is advised for the proposed work, as it enables one to replace all packets corresponding to certain temporal layers.

As parameters, the *StreamProcessing* software uses the filenames of the source stream, the inject stream and the output stream, a file containing Video Coding Layer (VCL) packet numbers, and a coding standard id (0: H.264/AVC; 1: H.265/HEVC; 2:H.266/VVC). Note that an important restriction is that the source and inject video should be in an annex B RAW format. This can be easily accomplished using the ffmpeg command line as provided on the website. A visual example of the software's workflow is given in Fig. 5.

4. Evaluation

This section evaluates the proposed solution. First, Section 4.1 describes the experimental setup. Then, Section 4.2 reports on the transfer of bitrate and quality when injecting temporal layers from a high-bitrate-high-quality Augmentation Stream into a low-bitrate-low-quality Base Stream. Next, the impact on the compression

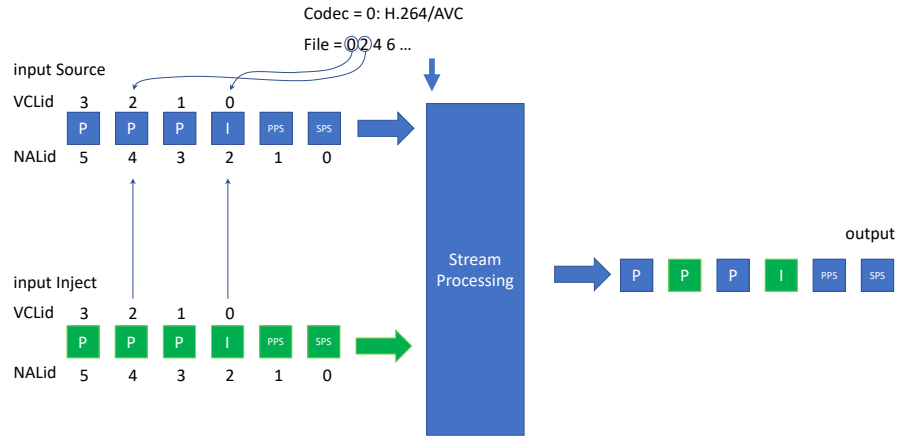


Fig. 5. The *StreamProcessing* software accompanying this work allows to combine video packets from the inject stream to be inserted in the source stream corresponding to the VCL packet numbers as provided through a text file.

efficiency is given in Section 4.3. Finally, Section 4.4 considers the jumps in quality that occur due to frequently injecting high-quality frames. Note that we do not experimentally evaluate the computational complexity of creating the intermediate streams. That is because combining the NALUs of two streams is practically instant, as no video or entropy decoding is required.

4.1. *Experimental Setup*

The experiments were performed on 22 sequences with resolutions between 416×240 and 2560×1600, namely *BlowingBubbles, BasketballPass, BQSquare, RaceHorses, BasketballDrill, BasketballDrillText, PartyScene, RaceHorses, BQMall, Johnny, FourPeople, KristenAndSara, SlideEditing, SlideShow, ChinaSpeed, BasketballDrive, BQTerrace, Cactus, Kimono, ParkScene, ParkJoy, Traffic,* and *PeopleOnStreet* [29]. These sequences contain between 150 and 600 frames and have a frame rate between 20 and 60 frames per second (fps).

To create the source encodings, each of the sequences are compressed using a random-access configuration, with the *slower*, *slow* and *faster* presets of the Fraunhofer Versatile Video Encoder (VVenC) [30] version 10.2 (which uses the H.266/VVC standard). We used the random-access configurations, using 6 hierarchical temporal layers (tid from 0 to 5), where the first frame is an I-frame and all other frames are P- or B-frames. In other words, there is a P-frame every $2^(6-1) = 32$ frames, which is also the GOP size. Each sequence is compressed using four different Quantization Parameters (QPs), namely 22, 27, 32, and 37. Then, for each combination of two different QPs at the same preset, the two corresponding source streams are combined into 5 intermediate C-streams.

To measure the quality, we used the Peak Signal-to-Noise Ratio (PSNR), Video Multimethod Assessment Fusion (VMAF) [31] and Structural SIMilarity

Table 1. Experimental results, using QP_B=32 and QP_A=22 and the slower preset (median values).

tid	Transfer_{BR}	Transfer_{PSNR}	$\text{Inefficiency}_{PSNR}$	Frame PSNR MAD
0	20%	21%	38%	0.9 dB
1	34%	27%	52%	1.2 dB
2	52%	38%	66%	1.8 dB
3	67%	53%	51%	2.3 dB
4	82%	76%	30%	3.0 dB

(SSIM) [32] quality metrics.

Note that our previous work only reported the results of the *slower* preset and a combination of two specific quality levels, namely QP_B=32 and QP_A=22. Moreover, it only reported the PSNR quality metric results. In contrast, this extended work presents the results for more QP pairs in Section 4.6, other presets in Section 4.7, as well as the VMAF and SSIM quality metric results in Section 4.8. Additionally, all raw results are available on our website[b].

4.2. *Transfer of Rate & Transfer of Quality*

This section discusses the Transfer of Rate and Transfer of Quality, which signifies the extent to which the C-stream bridges the gap between the B-stream and A-stream. Generically, the Transfer of *Metric* is expressed as a percentage between the *Metric* of the B-stream and A-stream, where *Metric* could be a rate or quality metric. $\text{Transfer}_{\text{Metric}}$ is calculated as in Eq. (1), in which Metric_A, Metric_B, and Metric_C represent the measured value of the A-, B-, and C-stream, respectively. A transfer of 0% means the C-stream is at the same level of the B-stream, whereas a transfer of 100% means the C-stream is at the same level of the A-stream. In this section, we consider the transfer or BitRate (BR) and the PSNR quality metric.

$$\text{Transfer}_{\text{Metric}} = \frac{\text{Metric}_C - \text{Metric}_B}{\text{Metric}_A - \text{Metric}_B} \tag{1}$$

Table 1 shows the median Transfer_{BR} and Transfer_{PSNR} values for each tid value, when combining a B-stream with QP_B=32 and A-stream with QP_A=22, both encoded using the *slower* preset. We observe that the bitrates and PSNR values of the 5 intermediate C-streams are distributed relatively evenly between the bitrates and PSNR values of the two source streams.

To give the reader some more feeling with the Transfer_{BR} and Transfer_{PSNR} of the intermediate representations, we present Fig. 6a. The figure shows the RD-curve of four *slower* encodes of the *BQSquare* sequence (including the two source streams

[b]Experimental results are available on
`https://media.idlab.ugent.be/temporal-layer-injection`

Table 2. Experimental results, using QP_B=32 and **QP_A=27** and the slower preset (median values).

tid	Transfer_{BR}	Transfer_{PSNR}	$\text{Inefficiency}_{PSNR}$	Frame PSNR MAD
0	26%	28%	12%	0.7 dB
1	40%	37%	16%	0.8 dB
2	51%	48%	19%	1.0 dB
3	67%	60%	18%	1.3 dB
4	86%	81%	13%	1.6 dB

Table 3. Experimental results, using **QP_B=37** and QP_A=22 and the slower preset (median values).

tid	Transfer_{BR}	Transfer_{PSNR}	$\text{Inefficiency}_{PSNR}$	Frame PSNR MAD
0	22%	22%	103%	1.0 dB
1	35%	29%	133%	1.4 dB
2	52%	38%	150%	2.1 dB
3	66%	54%	103%	3.0 dB
4	82%	76%	58%	3.9 dB

Table 4. Experimental results, using QP_B=32 and QP_A=22 and the **slow** preset (median values).

tid	Transfer_{BR}	Transfer_{PSNR}	$\text{Inefficiency}_{PSNR}$	Frame PSNR MAD
0	20%	20%	39%	0.9 dB
1	34%	27%	54%	1.3 dB
2	52%	36%	68%	1.9 dB
3	67%	53%	53%	2.4 dB
4	82%	75%	30%	3.2 dB

at QP_B=32 and QP_A=22), as well as the corresponding 5 C-streams. This figure showcases that our proposed solution can effectively create 5 combined streams with increasing bitrates and corresponding quality values. Additionally, corresponding visual examples of a crop of a frame of the *BQSquare* sequence are given in Fig. 2. By injecting more temporal layers (i.e., increasing tid), both the bitrate and quality increase gradually. More visual high-quality examples are given on our website[c].

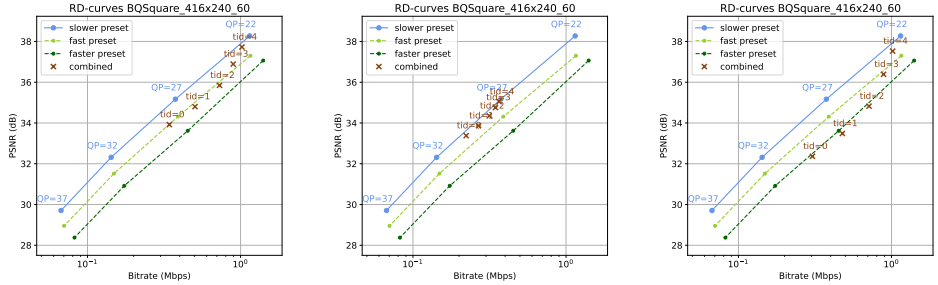
[c]More visual examples are available on
`https://media.idlab.ugent.be/temporal-layer-injection`

Table 5. Experimental results, using QP_B=32 and QP_A=22 and the **faster** preset (median values).

tid	Transfer_{BR}	Transfer_{PSNR}	$\text{Inefficiency}_{PSNR}$	Frame PSNR MAD
0	18%	19%	34%	1.0 dB
1	32%	28%	50%	1.4 dB
2	50%	37%	63%	1.9 dB
3	65%	53%	50%	2.6 dB
4	81%	76%	27%	3.5 dB

Table 6. Experimental results, using QP_B=32 and QP_A=22 and the slower preset (median values), for the **VMAF and SSIM quality metrics**.

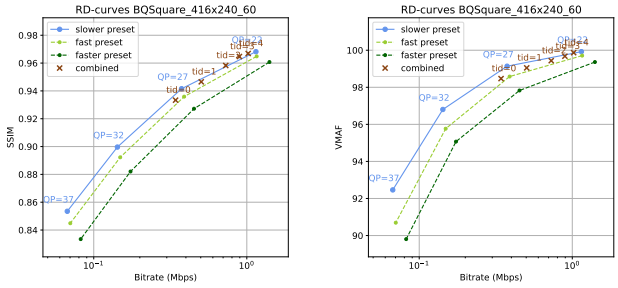
tid	Transfer_{SSIM}	$\text{Inefficiency}_{SSIM}$	Transfer_{VMAF}	$\text{Inefficiency}_{VMAF}$
0	33%	31%	25%	49%
1	48%	49%	34%	76%
2	63%	57%	47%	85%
3	71%	42%	63%	87%
4	85%	22%	87%	70%



(a) QP_B=32 and QP_A=22 (b) QP_B=32 and QP_A=27 (c) QP_B=37 and QP_A=22

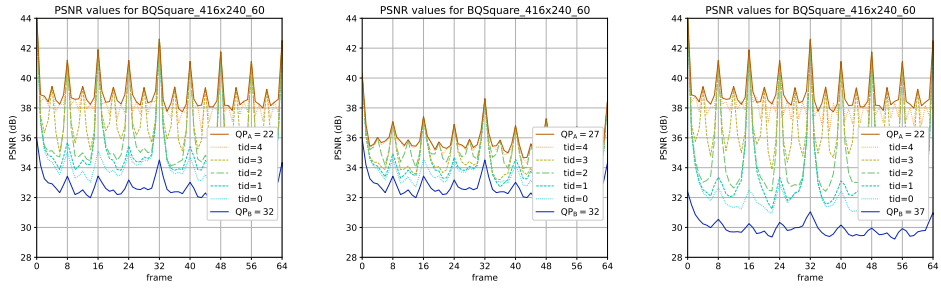
Fig. 6. RD-curves of encodes of *BQSquare* using the *slower*, *fast*, and *faster* presets, in addition to the 5 intermediate version that can be created by combining two *slow* encodes.

The points marked on Fig. 6a are obtained by averaging the PSNR values of all frames of the *BQSquare* sequence. To enable a deeper understanding of the quality improvements due to injecting more temporal layers, we present Fig. 8a. In this figure, the PSNR values of each of the first 64 frames are plotted for the source streams, as well as of the five C-streams with increasing tids. For the C-stream with tid=0, the PSNR values at frame 0, 32, and 64 are equal to the PSNR value of the



(a) SSIM (b) VMAF

Fig. 7. RD-curves of encodes of *BQSquare* using the *slower*, *fast*, and *faster* presets, in addition to the 5 intermediate version that can be created by combining two *slow* encodes. We utilize QP_B=32 and QP_A=22. We use the **SSIM (a) and VMAF (b)** as quality (distortion) metrics.



(a) QP_B=32 and QP_A=22 (b) QP_B=32 and QP_A=27 (c) QP_B=37 and QP_A=22

Fig. 8. PSNR values for the first 64 frames of *BQsquare*, for the two source streams with certain QP_B and QP_A values, as well as the 5 corresponding combined streams.

A-stream (QP_A=22). Most importantly, the PSNR values of all other frames are higher than those of the B-stream (QP_B=32). In other words, the quality increased over the entire video, also for the frames that were not injected. This means that replacing the lowest temporal layers with higher-quality layers of the A-stream has a quality-increasing effect on the frames of higher temporal layers as well (i.e., those with a higher tid). With each additional temporal layer that is injected, the PSNR values of the entire video gradually increase. As such, the 5 C-streams have quality levels that are relatively evenly distributed between the B- and A-stream.

4.3. *Compression Inefficiency*

The C-streams are not coded as efficiently as the source streams. To quantify this, this section measures and reports the compression inefficiency of the C-streams.

The compression efficiency is often measured using the Bjøntegaard-Delta rate (BD-rate) [33]. The BD-rate measures the average difference in bit-rate between two sets of video encodings for the same quality. It does this by fitting third-order logarithmic polynomials on the two sets of (Rate, PSNR) tuples. However, the set of Rate-Distortion tuples of our C-streams do not behave like the tuples of traditional video encodings. More specifically, the RD-curve of the C-streams is not a relatively straight line when plotted on a log-log-scale (e.g., see Fig. 6a). Instead, some C-streams are significantly more or less efficient than others. As the BD-rate was designed for streams following a traditional video encoding behavior and not for this new observed behavior, we do not use the BD-rate to report the compression inefficiency of the C-streams.

Since the BD-rate is not an adequate measure in our use case, we separately measure the compression inefficiency of each combined stream. We do this by comparing the bitrate of the combined stream with the theoretically optimal bitrate at the same quality value. More specifically, we define the compression inefficiency as the relative difference from the theoretically optimal bitrate to the actual bitrate of the combined stream. In this paper, we estimate the theoretically optimal bitrate by using linear interpolation of the two nearest neighboring log-bitrates of potential source streams (i.e., those encoded with the *slower* preset, using the QPs mentioned in Section 4.1, namely 22, 27, 32, and 37). In Fig. 6a, the optimal log-bitrate can be visually found by drawing a horizontal line from the position of a combined stream, and finding the intersection with the straight blue line of the *slower* preset. Finally, the exponential is taken of the theoretically optimal log-bitrate to obtain the optimal bitrate, before calculating the relative difference to the actual bitrate.

Table 1 gives the median compression inefficiency values for each tid value of the C-streams (using the PSNR as quality metric). For the considered configuration in the median case, we observe that the combined streams require between 30% and 66% more bits than the theoretically ideal case. To put these values in context, they are compared to the inefficiencies when using other, faster presets in Section 4.5.

4.4. *Quality Fluctuations*

As can be seen in Fig. 8a, the PSNR value is not constant over the entire video. This is the case for the source streams encoded with the random-access configuration, but even more so for the combined streams. This section quantifies these quality jumps using the Frame PSNR Mean Absolute Difference (MAD), which is the MAD of the PSNR values of each two subsequent frames. A Frame PSNR MAD of zero means the PSNR is constant over the entire video, whereas a high Frame PSNR MAD signifies many quality fluctuations.

Table 1 shows the median quality fluctuations results in the last column. We observe an increasing Frame PSNR MAD for increasing tid. This is intuitive and can be visually confirmed in Fig. 8a, where we observe that the quality jumps more frequently to the higher quality of the A-stream (QP_A=22) for C-streams with

Table 7. Compression inefficiencies when going from a slower preset to the other VVenC presets.

Preset	Inefficiency (%) QP = 22	27	32	37	BD-Rate (%)	Speed-up
slow	5	6	7	7	6.7	x4.7
medium	12	14	14	15	14.9	x13.3
fast	29	32	32	34	34.1	x62.7
faster	54	57	58	63	60.5	x146.6

higher tids.

For example, the C-stream with tid = 4 has frequent quality jumps (i.e., every 2^{nd} frame), in contrast to the combined stream with tid = 0, which has infrequent quality jumps every 32^{nd} frame. However, it should be noted that the infrequent quality jump of the C-stream with tid = 0 is much larger than the frequent quality jump of the C-stream with tid = 4. Hence, C-streams with higher tids may exhibit more frequent flickering artifacts, whereas those with lower tids may experience more intense but infrequent flickering artifacts.

Future work could adapt the default random-access configuration such that the quality of the encoded streams remains more stable over time. By reducing the quality fluctuations in the source streams, the jumps in the combined streams could be less intense as well. However, the impact on the metrics presented in this paper should be analyzed.

4.5. *Comparison with Bitrate Ladder Creation using Faster Presets*

This section gives some context on how the proposed method performs in comparison with an alternative traditional solution to create a bitrate ladder under constrained computational resources. That is, when there are insufficient resources to encode a stream at each quality level using the *slower* preset, it is an option to switch to other, faster presets.

Table 7 shows the compression inefficiencies when going from the *slower* preset to the *slow*, *medium*, *fast*, and *faster* presets. Comparing these preset inefficiencies to those from the proposed solution (in Table 1), we see that the compression inefficiency values from the C-streams are comparable to the inefficiency values when going from a *slower* to a *fast* or *faster* preset.

To visualize the relationship, in Fig. 6, we additionally plotted the RD-curve of the *BQSquare* sequence using the *fast* and *faster* preset using the green lines. For this sequence and configuration, the C-streams are approximately as efficient as the *fast* encodings, but less efficient than the theoretically optimal encodings using the *slower* preset.

For reference and better interpretation of the utilized compression inefficiency

measure, Table 7 also reports the median BD-rates of the four presets (with reference to the *slower* preset). We observe that the compression inefficiency values are approximately equal at different QP values using the same preset, and that these are also approximately equal to the BD-rate.

Finally, the last column of Table 7 gives the reported speed-ups [34] for high-definition resolutions. Although faster speed-ups lead to a larger compression efficiency loss, it should be stressed that they still require significant computational resources. In contrast, the proposed method has a negligible computational complexity.

In summary, when the intermediate bitstreams are obtained by combining source streams encoded with the *slower* preset, the combined streams are about as compression-efficient as source streams encoded with the *fast* or *faster* preset. However, encoding the streams with these faster presets still requires significant computational resources. In contrast, the proposed method creates five intermediate bitstreams at no additional computational cost, and is hence ideal for real-time applications.

4.6. *Other Source Stream Quality Levels*

This section covers other quality levels for the two source streams, compared to QP_B=32 and QP_A=22 that have been used in the previous sections.

First, we decrease the difference in quality between the B-stream and A-stream to QP_B=32 and QP_A=27, of which the results are shown in Table 2, Fig. 6b, and Fig. 8b. We can make the following observations:

- We notice a better compression efficiency in the C-streams. That is, in Fig. 6b, the intermediate streams are closer to the blue line of the source streams than in Fig. 6a. Additionally, in Table 2, the Inefficiency values are much smaller than those in Table 1.
- The quality fluctuations in the C-streams are decreased, as can be seen in the Frame PSNR MAD values in Table 2 and Fig. 8b.
- Although the reported relative $\text{Transfer}_{\text{BR}}$ and $\text{Transfer}_{\text{PSNR}}$ values are similar as percentages, the corresponding absolute bitrate/PSNR steps between the C-streams is much smaller. This can be seen in Fig. 6b. As such, the C-streams span less of the desired range in bitrates and quality levels, and are hence less useful in the target use-case of bitrate ladder creation.

Second, we increase the difference in quality between the B-stream and A-stream to QP_B=37 and QP_A=22, of which the results are shown in Table 3, Fig. 6c, and Fig. 8c. We make the following observations:

- The C-streams are less compression efficient. That is, the C-streams in in Fig. 6c are closer to the dark-green line of the *faster* preset, in contrast to Fig. 6c where the C-streams are closer to the light-green line of the *fast*

preset. Additionally, in Table 3, the Inefficiency values are much larger than those in Table 1.
- The quality fluctuations are also more intense when using a larger quality difference between the source streams, as can be seen in the Frame PSNR MAD values in Table 3 and Fig. 8b.
- The steps in bitrate and quality levels between C-streams is much larger, as can be seen in Fig. 6c.

Taking the above observations into account, this paper mostly focuses on QP_B=32 and QP_A=22, which offers reasonable trade-off in quality/bitrate steps of the bitrate ladder, a reasonable compression efficiency loss, and reasonable intensity of quality fluctuations.

4.7. *Other Source Stream Presets*

Table 4 and Table 5 show the experimental results when using the *slow* and *faster* presets, rather than the *slower* preset that was used in the previous sections. For all these tables, we use the default QP_B=32 and QP_A=22. We observe similar results for all metrics, demonstrating that our proposed method works for a variety of presets, and hence codec settings in general.

4.8. *Other Quality Metrics*

Table 6 shows the experimental results when using the SSIM and VMAF as quality metric for the Transfer of Quality and Compression Inefficiency metric. This is in contrast with Table 1 that used the PSNR as quality metric. For both these tables, we use the default QP_B=32 and QP_A=22. In terms of Transfer of Quality, we notice slightly higher values for the $\text{Transfer}_{\text{VMAF}}$, and even higher for the $\text{Transfer}_{\text{SSIM}}$. The general pattern is the same, though: the quality levels are distributed relatively evenly. In terms of compression ineffeciency, we notice slightly lower values for $\text{Inefficiency}_{\text{SSIM}}$, and slightly higher values for $\text{Inefficiency}_{\text{VMAF}}$. In Fig. 7, we additionally plot the RD-curves using the SSIM and VMAF as quality (distortion) metrics, for the *BQSquare* sequence, similar as in Fig. 6 (which used the PSNR). In general, in both figures, we can see that the intermediate streams are close in efficiency to the light green light of the *fast* preset.

5. Conclusion

To enable video streaming providers to encode videos at various quality levels under constrained computational resources, this paper proposed a novel method to create up to five intermediate bitstreams by splicing two encoded source streams. The intermediate bitstreams are created by injecting temporal layers from a higher-quality Augmentation Stream into a lower-quality Base Stream. We listed the requirements to implement this in the H.266/VVC standard, provided the source code and corresponding documentation, and performed extensive experimental evaluations.

The experimental results demonstrated that both the quality and bitrate of the Combined Stream increase when injecting more temporal layers. Additionally, the created intermediate streams are relatively evenly distributed between the bitrates and quality levels of the source streams. The disadvantage is that the combined streams are coded less efficiently in comparison with the source streams - about as efficient as using a *fast* or *faster* preset in comparison with a *slower* preset. Additionally, the intermediate streams exhibit some quality fluctuations that may become perceptible when the quality difference between the source streams is large. Most importantly, the advantage is that the new intermediate streams are created at no additional encoding complexity.

Future work could mix resolutions of the source streams. As such, we could provide better compatibility with bitrate ladders in practical adaptive streaming scenarios. Additionally, future work may assess the quality using subjective user experiments.

In conclusion, the proposed method enables the generation of new steps in a bitrate ladder at no computational overhead. As such, it is an attractive alternative to transcoding or encoding video streams with faster presets or older coding standards. This is a significant benefit in adaptive streaming architectures with constrained computational resources, such as in live streaming scenarios.

Acknowledgment

This work was funded in part by IDLab (Ghent University – imec), in part by Flanders Innovation & Entrepreneurship (VLAIO) project 5G-BROADCAST (HBC.2021.0674), in part by Research Foundation – Flanders (FWO), and in part by the European Union.

The computational resources (STEVIN Supercomputer Infrastructure) and services used in this work were kindly provided by Ghent University, the Flemish Supercomputer Center (VSC), the Hercules Foundation and the Flemish Government department EWI.

References

[1] VideoLAN, x264, the best H.264/AVC encoder.
[2] VideoLAN, x265, the free H.265/HEVC encoder.
[3] H. Mareen, C. Haems, T. Wauters, F. De Turck, P. Lambert and G. Van Wallendael, Temporal layer injection for fast bitrate ladder creation in video live streaming, in *International Symposium on Multimedia* 2023.
[4] A. V. Katsenou, J. Sole and D. R. Bull, Efficient bitrate ladder construction for content-optimized adaptive video streaming, *IEEE Open Journal of Signal Processing* **2** 496–511 (2021).
[5] P. Lebreton and K. Yamagishi, Network and content-dependent bitrate ladder estimation for adaptive bitrate video streaming, in *ICASSP 2021 - 2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)* 2021, pp. 4205–4209.

[6] V. V. Menon, H. Amirpour, M. Ghanbari and C. Timmerer, Efficient bitrate ladder construction for live video streaming, in *Proceedings of the 1st Mile-High Video Conference MHV '22*, (Association for Computing Machinery, New York, NY, USA, 2022), p. 99–100.

[7] A. Zabrovskiy, P. Agrawal, C. Timmerer and R. Prodan, Faust: Fast per-scene encoding using entropy-based scene detection and machine learning, in *2021 30th Conference of Open Innovations Association FRUCT* 2021, pp. 292–302.

[8] D. H. Finstad, H. K. Stensland, H. Espeland and P. Halvorsen, Improved multi-rate video encoding, in *2011 IEEE International Symposium on Multimedia* 2011, pp. 293–300.

[9] H. Amirpour, E. Çetinkaya, C. Timmerer and M. Ghanbari, Fast multi-rate encoding for adaptive http streaming, in *2020 Data Compression Conference (DCC)* 2020, pp. 358–358.

[10] D. Schroeder, A. Ilangovan, M. Reisslein and E. Steinbach, Efficient multi-rate video encoding for hevc-based adaptive http streaming, *IEEE Transactions on Circuits and Systems for Video Technology* **28**(1) 143–157 (2018).

[11] C. Van Goethem, J. De Praeter, T. Paridaens, G. Van Wallendael and P. Lambert, Multistream video encoder for generating multiple dynamic range bitstreams, in *2016 Picture Coding Symposium (PCS)* **1**1 2016, pp. 1–5.

[12] A. Vetro, C. Christopoulos and H. Sun, Video transcoding architectures and techniques: an overview, *IEEE Signal Processing Magazine* **20**(2) 18–29 (2003).

[13] A. Erfanian, F. Tashtarian, R. Farahani, C. Timmerer and H. Hellwagner, On Optimizing Resource Utilization in AVC-based Real-time Video Streaming, in *2020 6th IEEE Conference on Network Softwarization (NetSoft)* 6 2020, pp. 301–309.

[14] L. Pham Van, J. De Praeter, G. Van Wallendael, S. Van Leuven, J. De Cock and R. Van de Walle, Efficient Bit Rate Transcoding for High Efficiency Video Coding, *IEEE Transactions on Multimedia* **18** 364–378 (3 2016).

[15] J. De Praeter, G. Van Wallendael, J. Slowack and P. Lambert, Video Encoder Architecture for Low-Delay Live-Streaming Events, *IEEE Transactions on Multimedia* **19** 2252–2266 (10 2017).

[16] J. D. Praeter, C. Hollmann, R. Sjoberg, G. V. Wallendael and P. Lambert, Network-Distributed Video Coding, *arXiv* (2021).

[17] H. Schwarz, D. Marpe and T. Wiegand, Overview of the scalable video coding extension of the H.264/AVC standard, *IEEE Transactions on Circuits and Systems for Video Technology* **17**(9) 1103–1120 (2007).

[18] J. M. Boyce, Y. Ye, J. Chen and A. K. Ramasubramonian, Overview of SHVC: Scalable extensions of the high efficiency video coding standard, *IEEE Transactions on Circuits and Systems for Video Technology* **26**(1) 20–34 (2016).

[19] B. Bross, Y.-K. Wang, Y. Ye, S. Liu, J. Chen, G. J. Sullivan and J.-R. Ohm, Overview of the versatile video coding (VVC) standard and its applications, *IEEE Trans. Circuits Syst. Video Technol.* **31**(10) 3736–3764 (2021).

[20] N. Farber, E. Steinbach and B. Girod, Robust h.263 compatible transmission for mobile video server access, in *Proceedings of First International Workshop on Wireless Image/Video Communications* 1996, pp. 8–13.

[21] N. Farber and B. Girod, Robust h.263 compatible video transmission for mobile access to video servers, in *Proceedings of International Conference on Image Processing* **2**1997, pp. 73–76 vol.2.

[22] J. M. Boyce and A. M. Tourapis, Fast efficient channel change [set-top box applications], in *2005 Digest of Technical Papers. International Conference on Consumer Electronics, 2005. ICCE.* Jan. 2005, pp. 1–2.

[23] U. Jennehag and S. Pettersson, On synchronization frames for channel switching in a GOP-based IPTV environment, in *2008 5th IEEE Consumer Communications and Networking Conference, CCNC 2008* Jan. 2008, pp. 638–642.
[24] P.-J. Speelmans, HESP - High Efficiency Streaming Protocol, Internet-Draft draft-theo-hesp-01, Internet Engineering Task Force (November 2021), Work in Progress.
[25] G. Van Wallendael, H. Mareen, J. Vounckx and P. Lambert, Keyframe Insertion: Enabling Low-Latency Random Access and Packet Loss Repair, *Electronics* **10**(6) (2021).
[26] H. Mareen, M. Courteaux, P.-J. Speelmans, P. Lambert and G. V. Wallendael, A study on keyframe injection in three generations of video coding standards for fast channel switching and packet-loss repair, *Multimedia Tools and Applications* 1–17 (2023).
[27] G. Van Wallendael, P. Lambert, P.-J. Speelmans and H. Mareen, Mixed-resolution HESP for more efficient fast channel switching and packet-loss repair, in *Picture Coding Symposium (PCS)* 2022, pp. 319–323.
[28] M. N. Akcay, B. Kara, A. C. Begen, S. Ahsan, I. D. Curcio, K. Kammachi-Sreedhar and E. Aksu, Quality upshifting with auxiliary i-frame splicing, in *2023 15th International Conference on Quality of Multimedia Experience (QoMEX)* 2023, pp. 119–122.
[29] F. Bossen, Common test conditions and software reference configurations, Tech. Rep. JCTVC-L1100, ITU-T Joint Collaborative Team on Video Coding (JCT-VC) (2013).
[30] A. Wieckowski, J. Brandenburg, T. Hinz, C. Bartnik, V. George, G. Hege, C. Helmrich, A. Henkel, C. Lehmann, C. Stoffers, I. Zupancic, B. Bross and D. Marpe, VVenC: An open and optimized VVC encoder implementation, in *Proc. IEEE Int. Conf. Multimedia Expo Workshops (ICMEW)* 2021, pp. 1–2.
[31] Netflix Technology Blog, Toward A Practical Perceptual Video Quality Metric.
[32] Z. Wang, A. C. Bovik, H. R. Sheikh and E. P. Simoncelli, Image quality assessment: from error visibility to structural similarity, *IEEE Transactions on Image Processing* **13**(4) 600–612 (2004).
[33] G. Bjøntegaard, Calculation of average PSNR differences between RD-curves, Tech. Rep. VCEG-M33, ITU-T Video Coding Experts Group (VCEG) (Apr. 2001).
[34] Fraunhofer, VVenC Encoder Performance Accessed on 25 August 2025.