Introduction

In the context of heterogenous networks, it's a must to adapt multimedia contents in order to meet end user requirements. Using a study case, the goal of this report is to evaluate the performance of techniques for adaptation of compressed video and analyze through metrics (PSNR, bitrate, storage) which solution gives the best trade-off to implement by the operator SmartVideo4U.

The following diagram is for the required system to analyze:

Figure 1: General Case

The next table describes the streams and their requirements as a map legend for the general case:

Stream	Requirement
R1	704x576@60Hz with premium quality
R2	704x576@60Hz with standard quality
R3	352x288@30Hz

Table 1: Map Legend of General Case

There are additional requirements to fulfill:

- The operator wants a comparative analysis in terms of: bitrate vs PSNR; storage capacity needed and computational power (processing time) related to an encoder.
- The operator wants to keep available the video content with a lapse of 30 days with *4* different streams, considering the following options:
    - 1) Compressed ***non-scalable*** video (or single-layer) using **simulcast**;

- 2) Compressed ***non-scalable*** video (or single-layer) using **transcoding**;
- 3) Compressed ***scalable*** video (or single-layer) using **3 layers**

- R1 and R3 must provide an *average* quality (PSNR) approximately of 40 [dB], while R2 must be in the range of 30 -35 [dB]. A side note is that values can differ between them, as long as the difference in values between R1 and R2 is 5 [dB].
- The storage capacity given for the 3 previous cases (**simulcast**, **transcoding** and **scalable with 3 layers**) using 2 sequences as a comparison (must have at least 50 frames when using the encoder).
- Diagram blocks of the 3 cases.
- Post-analysis considering: storage, the requirements of real time vs offline operation (acquisition, encoding and transmission in real time), transcoding requirements and number of end-users simultaneously.

The sequences that fulfill the criteria were (same size):

- SOCCER_704x576.y4m -> 4CIF sequence
- SOCCER_352x288.y4m -> CIF sequence
- CITY_704x576.y4m -> 4CIF sequence
- CITY_352x288.y4m -> CIF sequence

Some sequences contain 600 frames, complying with the requirement for *at least* 50 frames in a sequence. The *CIF sequences* are obtained through a ***down conversion*** process. The sequences are available in: https://media.xiph.org/video/derf/

Note 2: There are other 2 sequences named **harbour** and **crew** which has also *600* frames, but **ice** has just *480*. A reminder will be that we can also extract frames to achieve from the ones who has 600 up to 480 or even less (for example, have sequences with 50,100,200, etc. frames) so at the end we'll have a set of sequences with the same number of frames to analyze. This can be performed as an input parameter in the encoder. From now on, we'll set 100 frames to be encoded in order to gain in speed to perform tasks.

We'll use the JSVM Software for Scalable Video Coding (SVC). The software consists of a group of functions that characterizes some essential aspects for Scalable Video Coding technique. The following list describes what are the before mentioned functions:

- ***H264AVCEncoderLibTestStatic***: executable used to encode the video in the scalable or non-scalable format. There's an important parameter that determines the quantization step to trade between compression quality and bitrate compression factor.
- ***H264AVCDecoderLibTestStatic***: executable used to decode the bitstream (*.264) generated by the scalable encoder.
- ***BitStreamExtractorStatic***: executable used to extract temporal, spatial and quality scalable sub-bitstreams from the bitstream generated by the scalable encoder.
- ***PSNRStatic***: executable used to compare the quality between two video sequences.

- **DownConvertStatic**: Executable for converting video to different resolutions.

We'll analyze the following techniques: simulcast, scalable coding and transcoding, showing the diagram block for every system, the configuration files (see Appendix 1), program outputs (see Appendix 2) for later on results and analysis to draw some conclusions.

Method

Simulcast

A generic schema for simulcast is based on the fact that the content (video from now on) is coded in ***independent*** streams where every stream will be provided to an end user (resolution and bitrate), meaning that the provider must store every stream. As a first conclusion, this could not be suitable in some situations for multiple streams because it will lead in a huge storage requirement.

The following diagram was adapted for the case study:

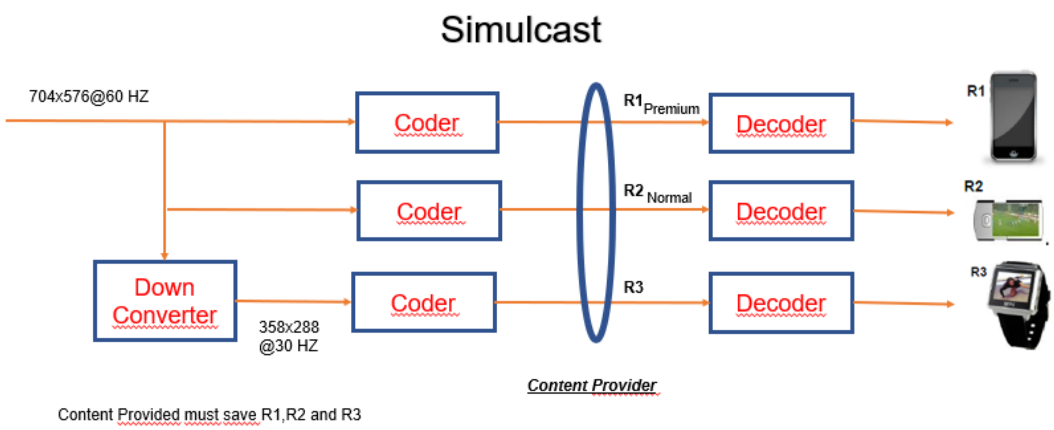

Figure 2: Simulcast Solution

The coder is configured as **Single Layer** (AVCMode =1 in the configuration file), meaning no scalability configuration or in more simple terms, typical encoding method (that's why the *single layer*).

We need to store all the 3 streams generated for R1, R2 and R3 respectively. An important mention is the difference between R1 and R2 is given by the quality which can be referred in the quantization step and the SNR as a consequence.

A main configuration file will start like this:

```
# JSVM Configuration File in AVC mode
#====================== GENERAL ================================================
AVCMode             1          # must be one for AVC simulations
InputFile           SEQUENCE_width_valuexheight_value_numframes_framerate_value.y4m
OutputFile          SEQUENCE_width_valuexheight_value_numframes_framerate_value.264
SourceWidth         width_value        # input  frame width or
SourceHeight        height_value       # input  frame height
FrameRate           framerate_value    # frame rate [Hz]
```

where we only need to specify certain parameters:

InputFile	Name of the sequence to encode
OutputFile	Encoded sequence in 264
SourceWidth	704 or 352
SourceHeight	576 or 288
FrameRate	60(4CIF) or 30(CIF)

We also need to set the **FramesToBeEncoded** = 100. The Quantization Parameter (in the Configuration File is a sum of **BasisQP** and **DeltaLayerXQuest**,i.e, **QP = BasisQP + DeltaLayerXQuest**) we will consider it as BasisQP for the sake of synthetize the analysis. The rest of the parameters will remain the same as in the manual for JSVM Software.

We'll move from different ranges of QP and increasing it will determine a deterioration in the PSNR but a lower value will increase the computational complexity. We could obtain some PSNR-Bitrate curves for different QP: 20,22,24,26,28,30,32,35 ,40 as input values in the encoder. From those values, we'll compromise 4 different ones that suffice the tradeoff between quality (30-40 dB, 35 dB) and computational complexity.

In the Appendix 1 are listed all the configuration files needed. Some outputs look like the following image:

```
  86 B REF   86  QP 23  Y 38.5047 dB  U 43.4559 dB  V 45.5582 dB  bits  148776
  87 B       85  QP 24  Y 38.1098 dB  U 43.0227 dB  V 45.5461 dB  bits   96856
  88 B       87  QP 24  Y 38.2386 dB  U 43.0890 dB  V 45.8052 dB  bits   91976
  89 P REF   92  QP 20  Y 41.3239 dB  U 45.9631 dB  V 47.6293 dB  bits  426760
  90 B REF   90  QP 23  Y 38.7869 dB  U 43.2686 dB  V 45.4104 dB  bits  138072
  91 B       89  QP 24  Y 38.4040 dB  U 42.9407 dB  V 45.5286 dB  bits   90688
  92 B       91  QP 24  Y 38.3475 dB  U 43.1071 dB  V 45.6808 dB  bits   97448
  93 P REF   96  QP 20  Y 41.2329 dB  U 45.8982 dB  V 47.5095 dB  bits  443560
  94 B REF   94  QP 23  Y 38.5878 dB  U 43.3338 dB  V 45.5202 dB  bits  135200
  95 B       93  QP 24  Y 38.2413 dB  U 43.1108 dB  V 45.8004 dB  bits  100832
  96 B       95  QP 24  Y 38.0020 dB  U 43.0161 dB  V 45.5509 dB  bits  100664
  97 B REF   98  QP 23  Y 38.7029 dB  U 43.5859 dB  V 45.6789 dB  bits  174936
  98 B       97  QP 24  Y 38.2798 dB  U 43.2822 dB  V 45.8020 dB  bits   97176
  99 B       99  QP 24  Y 38.2116 dB  U 42.9894 dB  V 45.3513 dB  bits  119008

   100 frames encoded:  Y 38.9891 dB  U 43.9047 dB  V 46.0359 dB

     average bit rate:  12049.3968 kbit/s  [2510291 byte for 1.667 sec]

 Encoding speed: 11765.530 ms/frame, Time:1176553.000 ms, Frames: 100
```

Figure 3: Output file for Soccer Sequence at 4CIF

Transcoding

Introduction

A **transcoding** function *adapt* a single bitstream with a given resolution and bitrate, into one or different with lower (for our purposes). There's just one stream in the server and the transcoder can work in the server or intermediate nodes of the network. It could have an increase in computational efficiency and some loss of quality, but depends on the solution itself. The transcoder can adapt bit rate(transrate), frame rate, spatial coding (transize) and coding standards. The following image shows a general schema:

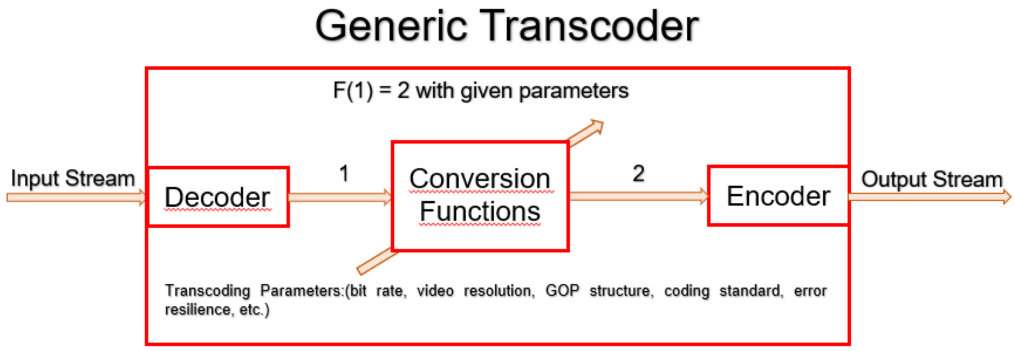

Figure 4: General Transcoding Process

We can separate the transcoding in two architectures: homogeneus (from now on, transrating) and heterogeneus (format conversion). In this document, we'll use the first one: transrating. This is done by implementing a ***cascading*** *encoder-decoder*. specifically, SNR (for generating R2) and Spatial (for generating R3).

Proposed Solution

The following block diagram represents the intended solution:

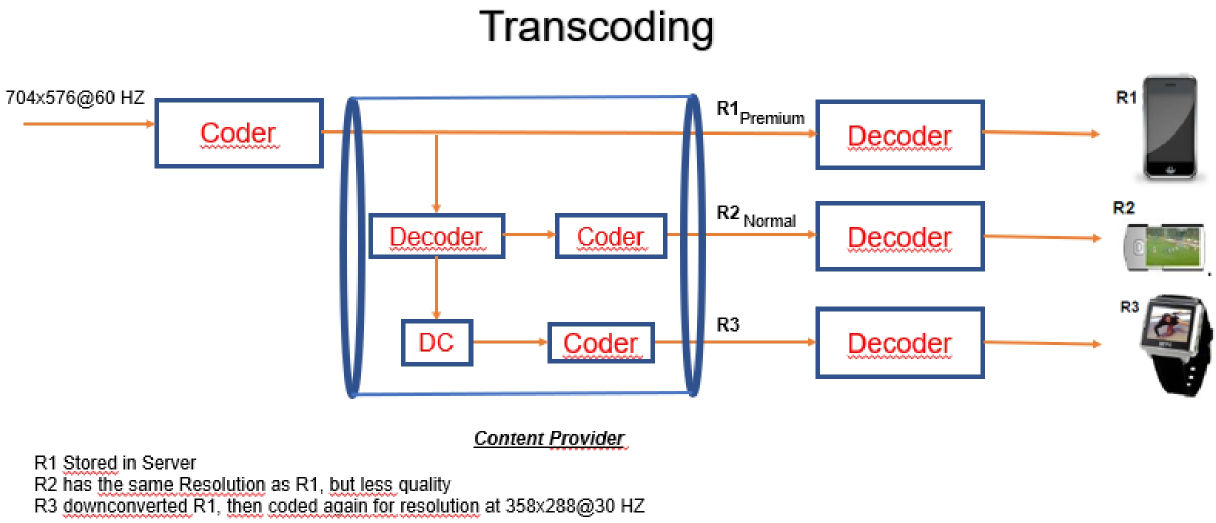

Figure 5:Transcoding Solution

The process to get the R1 service is the same as the singlelayer process, and it is the streams originated from the encoding to get R1 that are stored on the server. The server keeps only one stream in memory, R1. To obtain the R2 service which has the same resolution as the R1 service, but worse quality, it is necessary to decode R1 and encode it again with a higher encoding parameter (**QP**) to decrease the quality. To get R3 you must decompress the R1 stream, ***downconvert*** it to reduce the resolution and re-encode so that the stream can be sent to the user and then decoded. These decoding, re-encoding and downconverting processes are done on the server so that it is possible to transmit the stream the user wants.

The configuration files are very similar to simulcast, so we'll omit as an example but we'll focus on the **commands configuration** to use instead:

For R2:

```
R2 (Decode, Encode Again)
--------------------------------------------------------------------------------
H264AVCDecoderLibTestStatic coded.264 decoded.y4m
H264AVCEncoderLibTestStatic -pf differentQP.cfg > encoded_differentQP.txt
H264AVCDecoderLibTestStatic coded_differentQP.264 differentQP.y4m>decoded_differentQP.txt
PSNRStatic 704 576 file.y4m decoded_differentQP.y4m 0 0 coded_differentQP.264 60>psnr_differentQP.txt
```

Figure 6: Commands to generate stream R2

For R3:

```
R3
----------------------------------------------------------------------
H264AVCDecoderLibTestStatic 704_576_60_coded.264 704_576_60_decoded.y4m
DownConvertStatic 704 576 704_576_60_decoded.y4m 352 288 352_288_30_downconv.y4m 0 1
H264AVCEncoderLibTestStatic -pf 352_288_30.cfg > encoded_352_288_30.txt
H264AVCDecoderLibTestStatic 352_288_30_coded.264 352_288_30__transcoding.y4m>decoded_352_288_30.txt
PSNRStatic 352 288 352_288_30_orig.y4m 352_288_30__transcoding.y4m 0 0 352_288_30_coded.264 60>psnr_352_288_30_transcoding_orig.txt
```

Figure 7: Commands to generate R3 Stream

Some outputs look like the following image (this is for CIF case of sequence City:

```
transcoding.bat    psnr_singlelayer_city_352_288_30_transcoding_orig.txt ×
psnr_singlelayer_city_352_288_30_transcoding_orig.txt
45   39,3876 41,8426 44,2344
46   39,4549 42,5562 44,5600
47   39,0258 41,7394 44,0908
48   40,6135 44,6985 46,2450
49   38,6998 41,4657 43,8621

2775,4656    39,4028 42,8227 44,7813
```

Figure 8: PSNR Output for City CIF Sequence (QP = 20)

An important detail for transcoding case (and scalable coding) is related to storage size could be less because the server only stores one stream. In any case, this will be analyzed later on.

Scalable Coding

A video stream encoded in scalable mode can be decoded truncating parts of the bitstream (i.e., extracting some parts of it). This truncation could be at the content provider or other nodes in the network. This way, we only have one stream and then we discard bits, in accordance of our requirements as end-user. The goal is to keep "watching something", even when the quality deteriorates for a given factor, ex., congestion in the network.

It's characterized in having a layer base with a given bitrate and characteristics that can differ from the original stream. The base layer can be decoded independent of the enhancement layers, but in reverse, the upper layers require of the base layer. It's desirable and common to pick as the base layer the stream with the least resolution and the enhancement layers upper layers. This way, we guarantee the condition before mentioned of watching something. The following schema characterizes a scalable flux in detail:

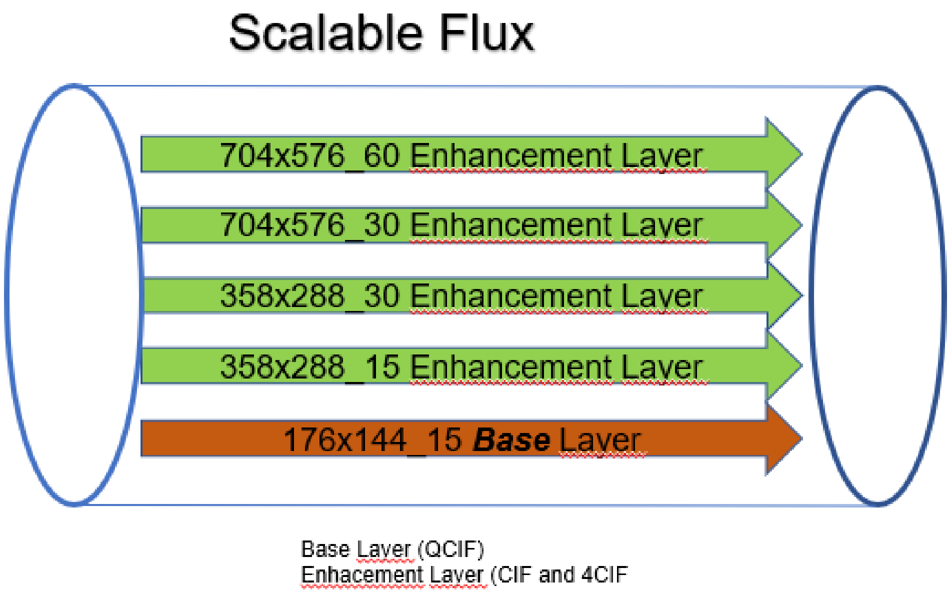

Figure 9: Example of SVC with base layer as lowest resolution

Scalable coding in three aspects: quality (SNR), spatial and temporal scalability. It's important to remark that these methods can be combined between each other for better improvement.

Temporal Scalability

Temporal scalability: Refers to the ability to reduce the frame rate of an encoded bitstream by dropping packets, thereby, reducing the bit rate of the stream.

Spatial Scalability

Spatial scalability: This spatial domain method encodes the base layer at a lower sampling dimension (e.g. resolution) than the upper layers. The reconstructed lower (base) layers of the sample are used as prediction of the upper layers.

Quality Scalability (SNR Scalability)

SNR scalability: is a spatial domain method where channels are encoded at identical sampling rates, but with different image qualities. The high priority binary string contains base layer data that can be added to the low priority refinement layer to build a high-quality image.

Proposed Solution

The requirement imposed by the operator in the case study refers to **scalable with 3 layers**. As a comparison, they could be using spatial or SNR or combined between these, but varying the Quantization Parameter for PSNR. We'll use ***Combined Scalability***. The obtained bitrate in R2 and R3 should be comparable with the other solutions (simulcast and transcoding). The range of PSNR and bitrate of the total scalable stream must match with the range of bitrates obtained for R1.

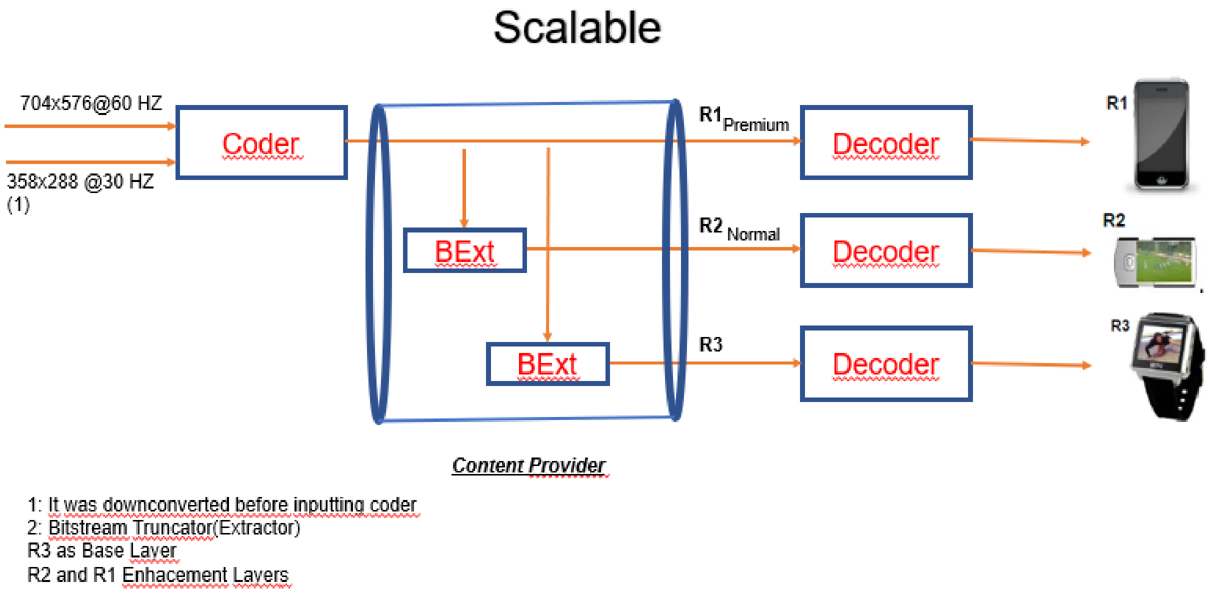

Figure 10 SVC Solution

We only save *one* stream and the rest will be generated in accordance if needed.

The concepts for spatial and SNR scalable coding can be combined for generating a bit-stream that supports a variety of spatio-temporal resolutions and rate points. The coder is configured as **Combined Scalability** with 4 configuration files (**main** and **layer0**, **layer1** and **layer2**). R2 will be obtained using **SNR Scalability** (less quality than R1, meaning higher QP) and R3 **Spatial Scalability.**

A main configuration file will start like this:

```
scalable > scalable_3L_city_4cif.cfg
# JSVM Main Configuration File
OutputFile              soccer_4cif_3L.264   # Bitstream file
FrameRate               60           # Maximum frame rate [Hz]
FramesToBeEncoded       100
GOPSize                 16            # GOP Size at maximum frame rate
CgsSnrRefinement        1             # SNR refinement as 1: MGS; 0: CGS
EncodeKeyPictures       1             # Key pics at T=0 0:none, 1:MGS, 2:all
MGSControl              1             # ME/MC for non-key pictures in MGS layers
                                      # 0:std, 1:ME with EL, 2:ME+MC with EL
BaseLayerMode           0             # Base layer mode 0,1: AVC compatible,
                                      #                     2: AVC w subseq SEI
SearchMode              4             # Search mode 0:BlockSearch, 4:FastSearch
SearchRange             32            # Search range Full Pel
NumLayers               3             # Number of layers
LayerCfg                layer0_CombScalable_soccer.cfg # Layer configuration file
LayerCfg                layer1_CombScalable_soccer.cfg # Layer configuration file
LayerCfg                layer2_CombScalable_soccer.cfg # Layer configuration file
```

Figure 11: Main Configuration file for City Sequence

The layers are as follows:

Layer 0:

```
scalable > layer0_CombScalable_city.cfg
# JSVM Layer Configuration File
InputFile           city_cif.y4m # Input  file
SourceWidth         352           # Input  frame width
SourceHeight        288           # Input  frame height
FrameRateIn         30           # Input  frame rate [Hz]
FrameRateOut        30           # Output frame rate [Hz]
ReconFile           city_cif_reclayer0.y4m
QP                  20           # Quantization parameters
MeQP0               28           # QP for mot. est. / mode decision stage 0
MeQP1               28           # QP for mot. est. / mode decision stage 1
MeQP2               28           # QP for mot. est. / mode decision stage 2
MeQP3               28           # QP for mot. est. / mode decision stage 3
MeQP4               28           # QP for mot. est. / mode decision stage 4
MeQP5               28           # QP for mot. est. / mode decision stage 5
```

Figure 12: Layer0 of City Sequence (CIF Resolution or R3)

Layer 1:

```
scalable > layer1_CombScalable_city.cfg
# JSVM Layer Configuration File
InputFile           city_4cif.y4m # Input  file
SourceWidth         704           # Input  frame width
SourceHeight        576           # Input  frame height
FrameRateIn         60           # Input  frame rate [Hz]
FrameRateOut        60           # Output frame rate [Hz]
ReconFile           city_4cif_reclayer1.y4m
InterLayerPred      2              # Inter-layer Pred. 0: no, 1: yes, 2:adap.
QP                  24             # Quantization parameters
MeQP0               32             # QP for mot. est. / mode decision stage 0
MeQP1               32             # QP for mot. est. / mode decision stage 1
MeQP2               32             # QP for mot. est. / mode decision stage 2
MeQP3               32             # QP for mot. est. / mode decision stage 3
MeQP4               32             # QP for mot. est. / mode decision stage 4
MeQP5               32             # QP for mot. est. / mode decision stage 5
```

Figure 13 : sLayer1 of City Sequence (Normal or R2)

Layer 2:

```
scalable > layer2_CombScalable_city.cfg
# JSVM Layer Configuration File
InputFile           city_4cif.y4m # Input  file
SourceWidth         704           # Input  frame width
SourceHeight        576           # Input  frame height
FrameRateIn         60           # Input  frame rate [Hz]
FrameRateOut        60           # Output frame rate [Hz]
ReconFile           city_4cif_reclayer2.y4m
InterLayerPred      1              # Inter-layer Pred. 0: no, 1: yes, 2:adap.
QP                  20             # Quantization parameters
MeQP0               28             # QP for mot. est. / mode decision stage 0
MeQP1               28             # QP for mot. est. / mode decision stage 1
MeQP2               28             # QP for mot. est. / mode decision stage 2
MeQP3               28             # QP for mot. est. / mode decision stage 3
MeQP4               28             # QP for mot. est. / mode decision stage 4
MeQP5               28             # QP for mot. est. / mode decision stage 5
```

Figure 14: Layer2 of City Sequence (Premium or R1)

The commands to execute the solution are:

```
scalable.bat
echo "Scalable Case"
echo "Scalable 3 layers city"
H264AVCEncoderLibTestStatic.exe -pf scalable_3L_city_4cif.cfg> scalable_city.txt
echo "Scalable 3 layers soccer"
H264AVCEncoderLibTestStatic.exe -pf scalable_3L_soccer_4cif.cfg> scalable_soccer.txt
echo "Showing layers city"
BitStreamExtractorStatic.exe city_4cif_3L.264> scalable_city_bsextractor.txt
echo "Showing layers soccer"
BitStreamExtractorStatic.exe soccer_4cif_3L.264> scalable_soccer_bsextractors.txt
pause
```

Figure 15:Scalable Coding for Sequences

```
bitstream.bat
BitStreamExtractorStatic.exe city_4cif_3L.264 city_r3.264 -sl 3> scalable_bsextractor_city_r3.txt
BitStreamExtractorStatic.exe city_4cif_3L.264 city_r2.264 -sl 8> scalable_bsextractor_city_r2.txt
BitStreamExtractorStatic.exe city_4cif_3L.264 city_r1.264 -sl 13> scalable_bsextractor_city_r1.txt

H264AVCDecoderLibTestStatic city_r3.264 city_r3_dec.y4m>decoded_city_r3.txt
H264AVCDecoderLibTestStatic city_r2.264 city_r2_dec.y4m>decoded_city_r2.txt
H264AVCDecoderLibTestStatic city_r1.264 city_r1_dec.y4m>decoded_city_r1.txt

PSNRStatic 352 288 city_cif.y4m  city_r3_dec.y4m 0 0 city_r3.264 30 >>PSNR_city_r3.txt
PSNRStatic 704 576  city_4cif.y4m city_r2_dec.y4m 0 0 city_r2.264 60 >>PSNR_city_r2.txt
PSNRStatic 704 576  city_4cif.y4m  city_r1_dec.y4m 0 0 city_r1.264 60 >>PSNR_city_r1.txt

BitStreamExtractorStatic.exe soccer_4cif_3L.264 soccer_r3.264 -sl 3> scalable_bsextractor_soccer_r3.txt
BitStreamExtractorStatic.exe soccer_4cif_3L.264 soccer_r2.264 -sl 8> scalable_bsextractor_soccer_r2.txt
BitStreamExtractorStatic.exe soccer_4cif_3L.264 soccer_r1.264 -sl 13> scalable_bsextractor_soccer_r1.txt

H264AVCDecoderLibTestStatic soccer_r3.264 soccer_r3_dec.y4m>decoded_soccer_r3.txt
H264AVCDecoderLibTestStatic soccer_r2.264 soccer_r2_dec.y4m>decoded_soccer_r2.txt
H264AVCDecoderLibTestStatic soccer_r1.264 soccer_r1_dec.y4m>decoded_soccer_r1.txt

PSNRStatic 352 288 soccer_cif.y4m  soccer_r3_dec.y4m 0 0 soccer_r3.264 30 >>PSNR_city_r3.txt
PSNRStatic 704 576  soccer_4cif.y4m soccer_r2_dec.y4m 0 0 soccer_r2.264 60 >>PSNR_city_r2.txt
PSNRStatic 704 576  soccer_4cif.y4m  soccer_r1_dec.y4m 0 0 soccer_r1.264 60 >>PSNR_city_r1.txt




pause
```

Figure 16: Extract Bitstream according layer, decode and get PSNR for sequences

As an example, it can be delivered the following image showing the scalable output:

```
SUMMARY:
                       bitrate    Min-bitr   Y-PSNR   U-PSNR   V-PSNR
                      --------- ---------- -------- -------- --------
   352x288 @  3.7500   682.1614   682.1614  44.3512  47.0017  48.4615
   352x288 @  7.5000   931.2138   931.2138  42.8016  45.3716  46.8623
   352x288 @ 15.0000  1198.8768  1198.8768  41.5263  43.9195  45.4674
   352x288 @ 30.0000  1526.1792  1526.1792  40.6877  43.0774  44.7024
   704x576 @  3.7500  4516.0929  2756.8071  47.9988  49.7004  50.6417
   704x576 @  7.5000  6551.6631  3653.0677  45.4885  47.7173  48.9682
   704x576 @ 15.0000  9270.4848  4646.7936  43.6720  46.2991  47.7667
   704x576 @ 30.0000 12923.2416  5741.0160  42.3470  45.3733  46.9880
   704x576 @ 60.0000 16151.4144  6418.1136  41.0960  44.6213  46.3403

Encoding speed: 12365.690 ms/frame, Time:1236569.000 ms, Frames: 100
```

Figure 17: Output for Encoding using SVC

Results

Bitrate vs PSNR

It's a must to obtain the curves Bitrate vs PSNR for all the 3 cases, because we'll gain insight. After processing all the data (for simulcast, transcoding and combined scalability with 3 layers), for 2 sequences (in this case, Soccer and City), with 100 (or 50 for CIF for 352x288) as number of frames for every case, it will be shown 6 figures, corresponding to every stream (R1, R2 and R3) for every sequence, i.e., **R1-Soccer**, **R1-City**, **R2-Soccer**, **R2-City**, **R3-Soccer**, **R3-City**.

A side note is that the values for QP were: 20,22,24,26 (for R1 and R3) and 24,26,28,30 for R2 in all cases. The lower QP the greater PSNR.

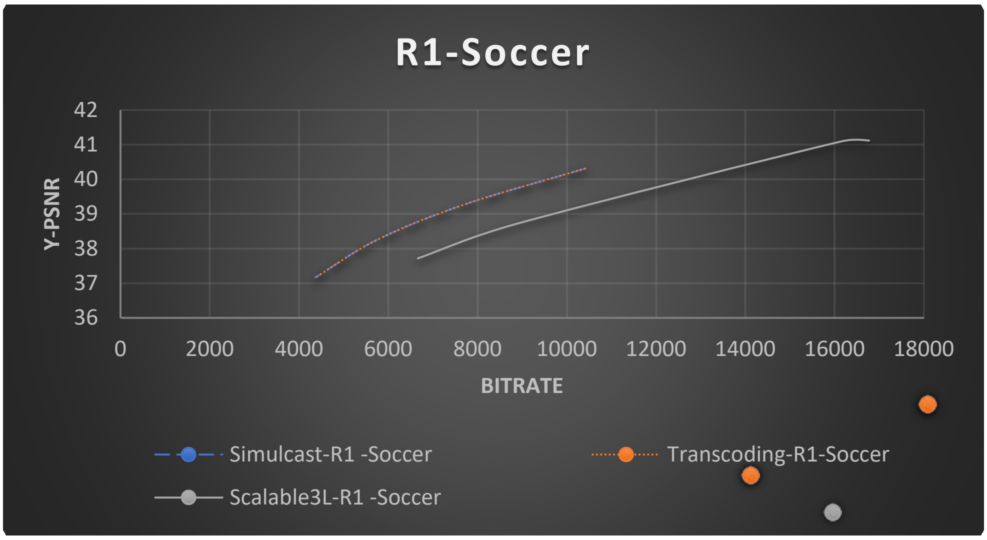

Figure 18: R1 Stream. Note that Simulcast and Transcoding Solutions have the same points

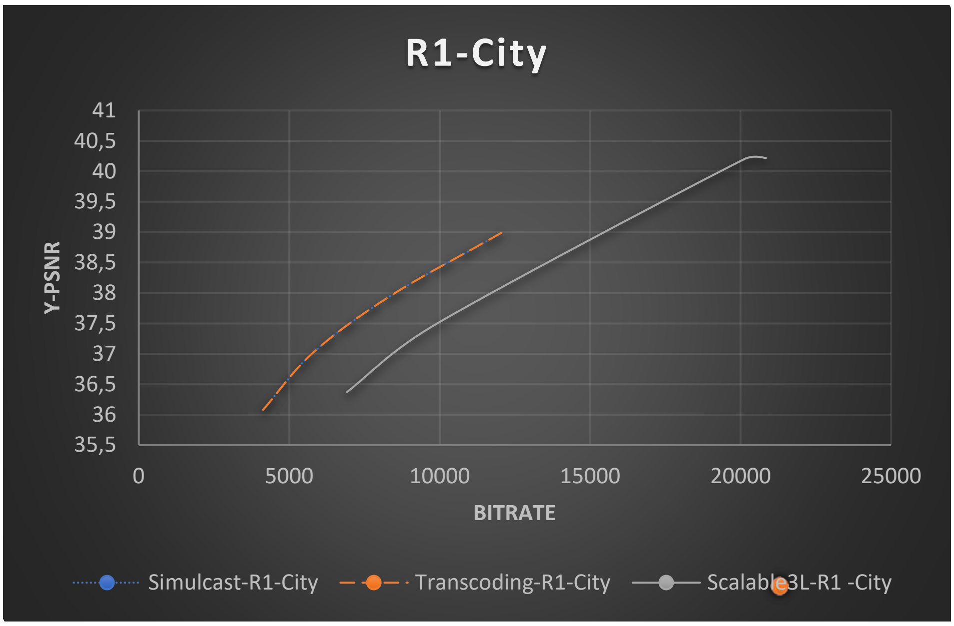

Figure 19: Similarly, Simulcast and Transcoding use the same points

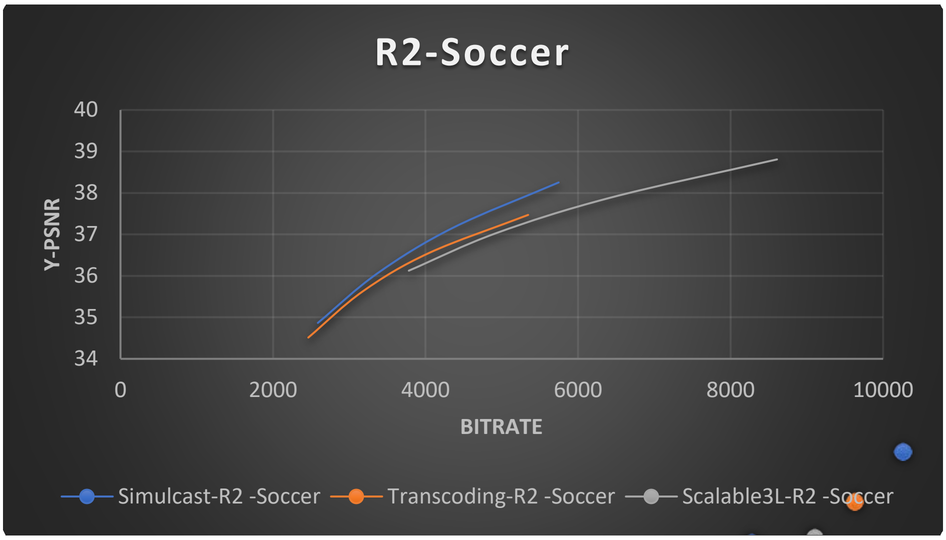

Figure 20: R2 Stream for Soccer Sequence

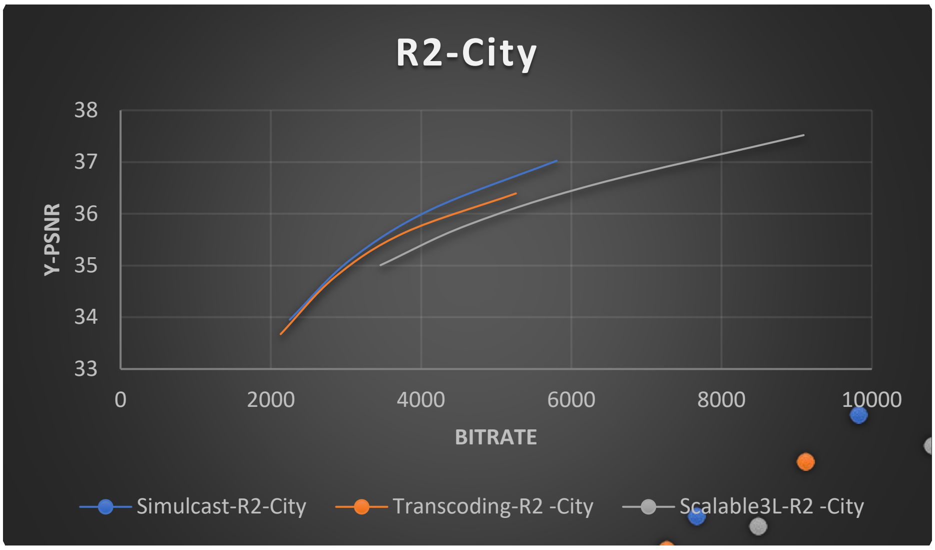

Figure 21: R2 Stream for City Sequence

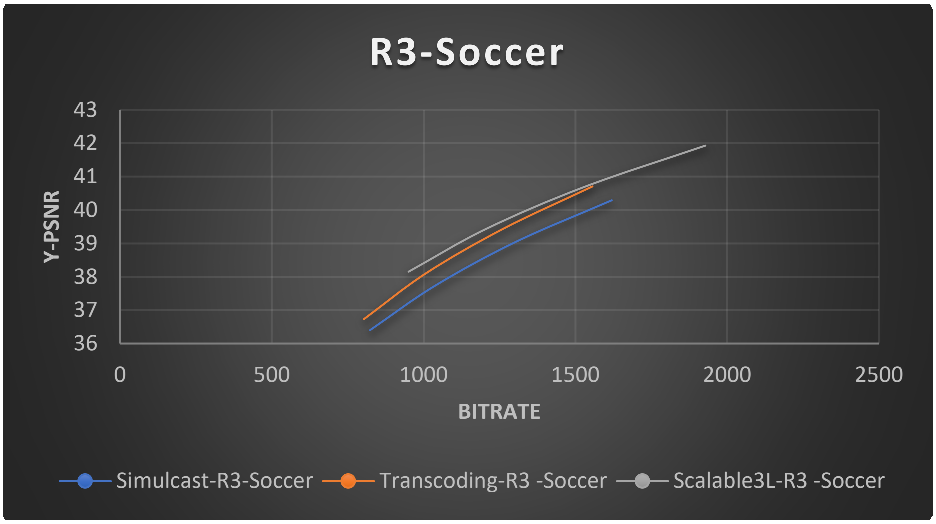

Figure 22: R3 Stream for Soccer Sequence

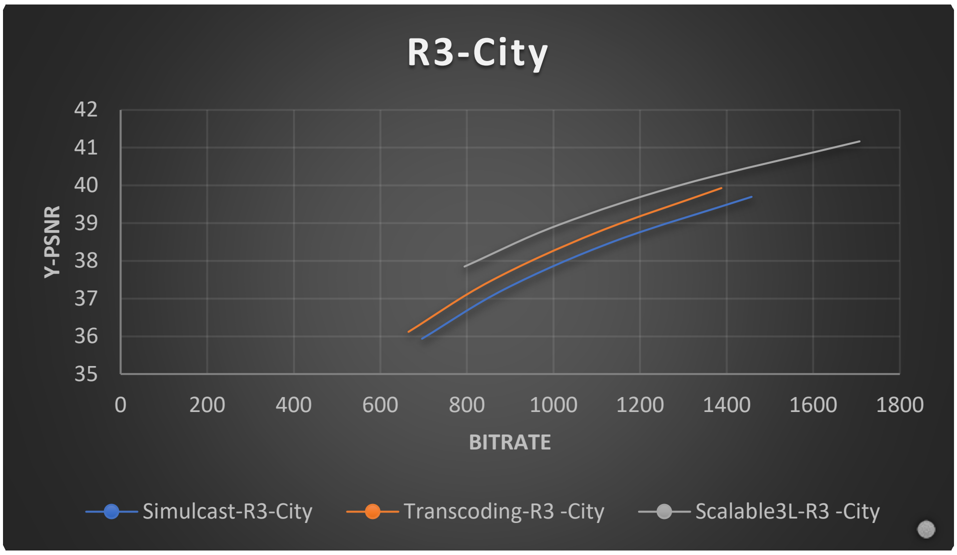

Figure 23:R3 Stream for City Sequence

Analysis

As a first note, for R1, simulcast and transcoding streams uses the same points, i.e.,the curves matches on the same points (overlapped). R2 (less quality) matches with points of less PSNR and higher QP (if plot more QP values, they'll match with values in the R1 curve but with less bitrate and PSNR, i.e., tending to lower left in the curves).

As a simple observation made, if the values accumulate to upper left, this will mean that for lower bitrates you can get good quality or higher PSNR. Conversely, happens when upper right but for higher bitrates. In all the cases, the scalable solution presents the highest quality for all the solutions but ultimately at a cost of higher bitrate.

Simulcast solutions aims to less bitrate but still acceptable keeping values higher than 30 dB of PSNR. As a compromise, there's the need to save all the streams. For larger streams and an enormous increase of devices, this solution will be insufficient and at some point, will be impractical to escalate for multiple resolutions (devices) due to increase of complexity. If the goal is to have a small system or a private one (business intended), then this could suffice.

The storage capacity is an important metric to analyze and cover. At first sight and explained previously, simulcast solutions need to save all the encoded streams, but transcoding and scalability just saves one instead.

The encoding time and decoding process can be then a trade-off to consider for Transcoding and Scalability Solutions. In real situations, these solutions are combinable for best approach to end-user experience (it's not the goal of this document to analyze a combined solution). If the storage is the priority and a ***fast*** solution should be provided, i.e., deploying a solution for clients, transcoding could suffice with the cost of considering the transcoding processes involved. Otherwise, scalability will be more than enough.

The scalability solution is desirable when massive number of target devices are expected. The scalability keeps higher quality for lower resolutions (it was the intention of the work) while "watching something" if network conditions deteriorate. For escalated solutions and long-term, scalability could suffice. If not, then the selection should be based on transcoding one with storage-awareness and encoding-time-awareness.

Scalability is more suitable for changes or alterations in some elements of the network as well, allowing upgrades of certain elements. As an example, if a fallback is needed, transcoding and scalability solutions are the preferrable ones for less space.

Similarly, to combining scalability and transcoding, it's possible to combine these with simulcast, providing a broader solution. The separated analysis is given to see the effects and behavior, but the solutions can be combined for better experience (as explained earlier).

It's important to remark that these PSNR-Bitrate curves were important for analysis, but not exclusive meaning that are other parameters/metrics to be aware. The group of software created by JSVM also provides parameters as ***size [bytes]*** and ***encoding speed[ms/frames], time-to-encode[ms].*** They could be used for further post-processing analysis.

Storage

As a rule of thumb, higher resolutions mean increased sized. The same encompasses quality, i.e., better quality implies larger files. Because the encoder provides a metric related with size, we'll analyze all the solutions around this metric. The following image shows the behaviour encoded filesize – QP. It will be shown that scalable cases will require larger storage capabilities, but it's important to mention that in the configuration files the ***MeQPX (QP value for Motion Estimation***) was very small and close to the Quantization Parameter Value,i.e., if QP = 20 then MeQPX = 26, QP = 22 -> MeQPX = 28 and so on. This led to large filesizes, but this value can be increased so smaller filesizes will be obtained. The same logic can be applied for AVC (Single Layer case) in the configuration file for the DeltaLayerQ values.

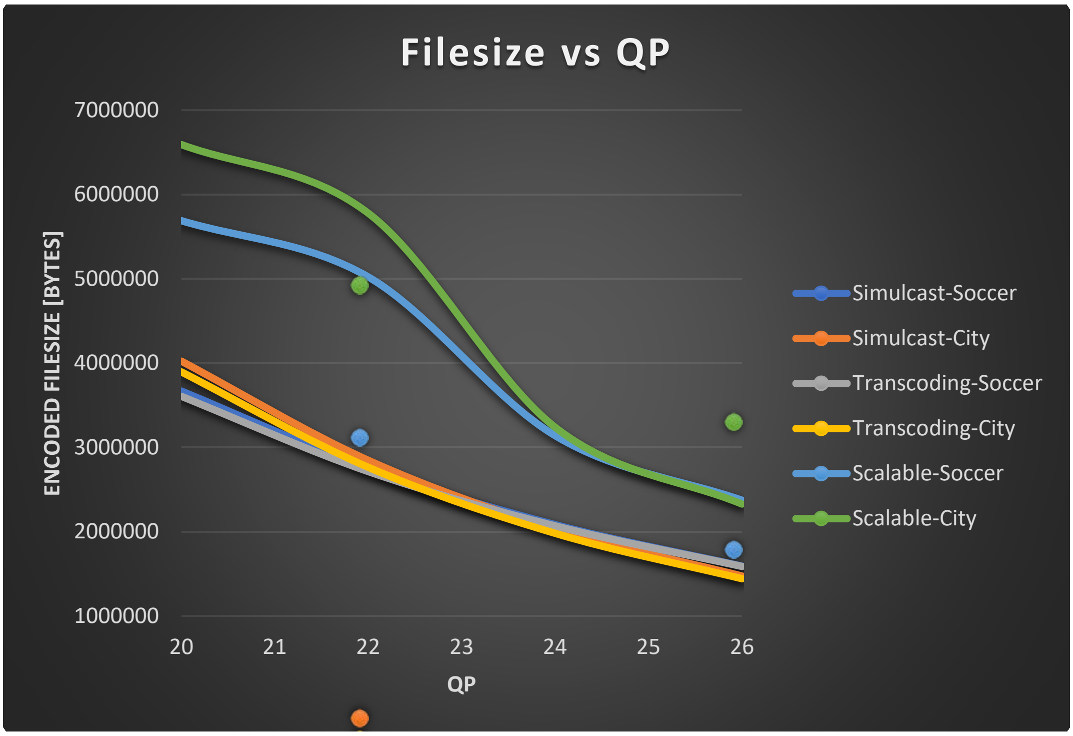

Figure 24: Filesize vs QP for different streams

As a result, Scalable Solution for good PSNR leads larger filesizes. Remark again, good PSNR . Interesting though, it's the result obtained for transcoding and simulcast solutions which are closely in values between each other. If the processing speed was sufficient, then transcoding could be a match since we won't have to worry to much about storage capabilities in nodes and servers. Otherwise, simulcast will be the selected case.

Timing Requirements

Requirements related with real-time and offline are dependent of the given solution. If you store all the streams, then the time to consider will be the latency of the network and decoding time. Otherwise, you will have to consider "transcoding-process [s]" for transcoding solution and "bitstream-extraction-time [s] + decoding-time at Rx [s]", also including the latency of the given network. Elements/Nodes in heterogenous networks are constrained to given circumstances and can vary over time, scalability and transcoding cases should be monitored for timing constraints. As a general rule, the transcoding process could be more demanding and may not suffice the end-solution if a larger increase latency occurs. The time of the down-conversion has to be considered as well (it was not considered in for this solution), but more important is the encoding *again* time and processes. Given an increase of resolutions, the total encoding time will be an important metric. In any case, this should be analyzed because could be more suitable than scalability.

References

Assunção, P. (2022). *SCM Classes.* Leiria: IPLeiria.

JSVM Software Manual. (June 14th, 2011). ISO/IEC MPEG.

xiph.org. (s.d.). *xiph.org*. The Xiph.Org Foundation: https://media.xiph.org/video/derf/

Appendix

Configuration Files

The firs appendix is related with the configuration files. First, it's presented a screenshot of the sequences obtained in the website and later on the configuration files required for the three solutions.

City Sequence:

Figure 25: City Sequence

Soccer Sequence:

Figure 26: Soccer Sequence

Outputs

All the files are available at

https://drive.google.com/drive/folders/1NsFWrKjhNVLfrX8aWwBsIUZXgQe8Z9ks?usp=sharing

https://github.com/alexnpz/video-adaptation-analysis