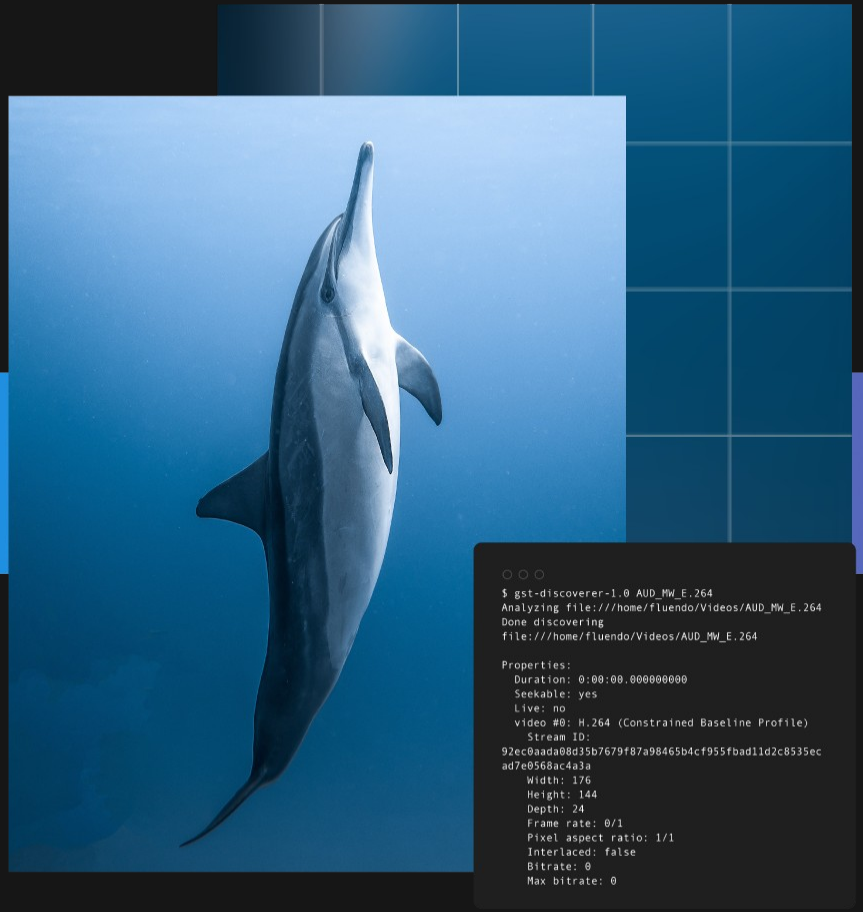
fluendo

Tools to profile a video encoder



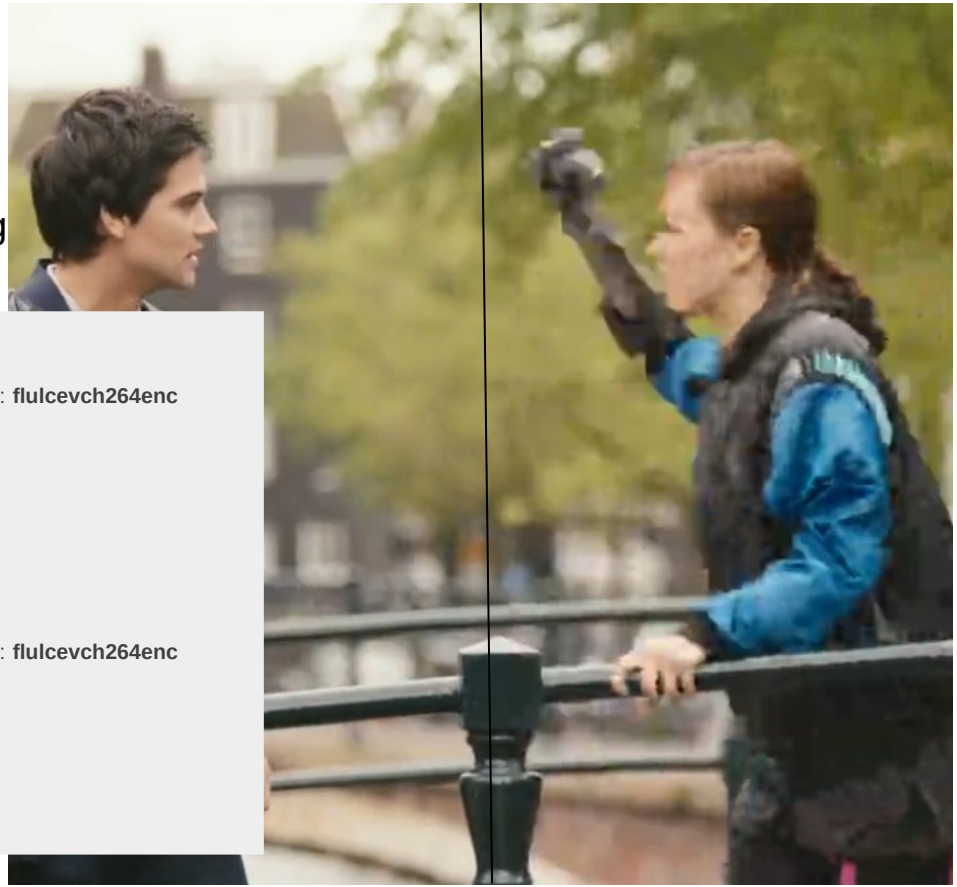
Oct 23rd, 2025 - London – Diego Nieto & Rubén González

Index

Why profiling our encoder?

Profiling

Profiling: the activity of collecting important and useful details about someone or something



From:

/GstPipeline:pipeline0/GstEncoderStats:encoderstats0: last-message = Encoder: **flulcevch264enc**
Output size: 597 KB
Bitrate: 1839.997 kbps
Processing time: 168 ms
CPU: 0.01 s
VMAF: 97.432
Encode latency: **200.511 ms**

To:

/GstPipeline:pipeline0/GstEncoderStats:encoderstats0: last-message = Encoder: **flulcevch264enc**
Output size: 549 KB
Bitrate: 1831.930 kbps
Processing time: 14 ms
CPU: 0 s
VMAF: 95.001
Encode latency: **13.561 ms**

Encoding stats

Main metrics

- Bitrate
- Encode time: Speed/Time/Resources: CPU/GPU
- Latency
- Quality

Encoding stats

Encoding time

- CLI tools:
  - htop
  - nvtop
  - Time (User time + System time) / rusage
  - Measure-command (Windows)

- GStreamer
  - GST_DEBUG="GST_TRACER:7" GST_TRACERS="stats;rusage" \
    GST_DEBUG_FILE=trace.log …
  - Problem: What if we need to aggregate external threads? E.g. lcevc
  - Proposal: An element that aggregates these timings in an isolated thread

Encoding stats

Encode latency: The delay between input and output during encoding, often critical in real-time applications.

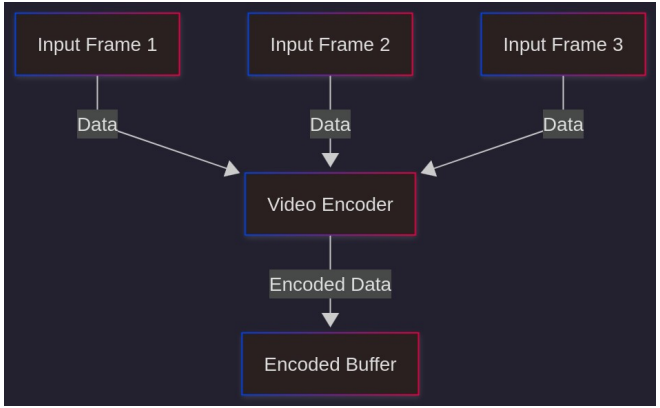
There are two halves to the latency problem. There is latency at the decoder and latency at the encoder. --tune *zerolatency* removes latency from both sides.

x264

```
else if( len == 11 && !strncasecmp( tune, "zerolatency", 11
{
    param->rc.i_lookahead = 0;
    param->i_sync_lookahead = 0;
    param->i_bframe = 0;
    param->b_sliced_threads = 1;
    param->b_vfr_input = 0;
    param->rc.b_mb_tree = 0;
}
```

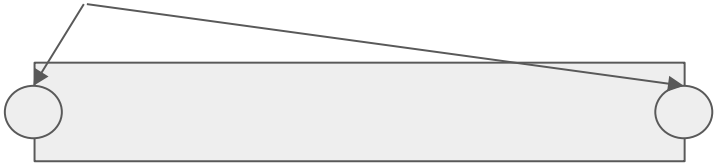
```
else if (!strcmp(tune, "zerolatency") ||
         !strcmp(tune, "zero-latency"))
{
    param->bFrameAdaptive = 0;
    param->bframes = 0;
    param->lookaheadDepth = 0;
    param->scenecutThreshold = 0;
    param->bHistBasedSceneCut = 0;
    param->rc.cuTree = 0;
    param->frameNumThreads = 1;
}
```

Encoding stats



Encode latency

- What happens whether the encoder needs to receive several frames before to output a buffer?
  - GST_TRACERS
    - Problem: not accurate results (when queues inside and not 1-input 1-output)
  - Proposal: add probes in both sink and src pads to measure it



Encoding stats

What about quality?

Encoding stats

Quality metrics

Reference metrics

- PSNR: old well know. Good for compression
- SSIM: structural, luminance and contrast
- VMAF: machine learning based. Extracts both temporal and spatial features

Non reference metrics

- Brisque

Applying reference metrics, e.g. vmaf, requires to decode the data beforehand. Proposal: what if we use the reconstructed frame from the encoding process?

Encoding stats

How to gather all these stats to compare videos in real time encoding?

Comparing encoders

github.com/fluendo/gst-plugins-rs/pull/4 → WIP

```
Plugin Details:
  Name                     videostats
  Description              GStreamer Video Stats Plugin
  Filename                 /home/dnieto/workspace/gst-plugins-rs/target/debug/libgstvideostats.so
  Version                  0.1.0-commit-id
  License                  MPL
  Source module            gst-plugin-videostats
  Documentation            https://gstreamer.freedesktop.org/documentation/videostats/
  Source release date      2025-09-28
  Binary package           gst-plugin-videostats
  Origin URL               https://gitlab.freedesktop.org/gstreamer/gst-plugins-rs

  video-compare-mixer: VideoCompareMixer
  video-encoder-stats: EncoderStats

  2 features:
  +-- 2 elements
```

videostats plugin

DEMO!

```
gst-launch-1.0 \
    filesrc location="/home/dnieto/Downloads/tears_of_steel_1080p.mov" ! \
    qtdemux name=demux demux.video_0 ! \
    queue ! \
    decodebin3 ! \
    videoconvertscale ! \
    capsfilter caps=\"video/x-raw,aspect-ratio=1/1,framerate=24/1\" ! identity sync=true ! \
    tee name=tee \
    tee.src_0  ! video-encoder-stats encoder=\"x264enc bitrate=1024 tune=zerolatency speed-preset=ultrafast threads=4 key-int-max=256 b-adapt=false vbv-buf-capacity=120\" name=vs0 vmaf-stats=false \
    tee.src_1  ! video-encoder-stats encoder=\"openh264enc bitrate=1024000 complexity=low rate-control=bitrate background-detection=false scene-change-detection=false gop-size=256\" name=vs1 decoder="h264parse ! avdec_h264" ! fakesink \
    video-compare-mixer split-screen=true backend=CPU name=mixer \
    vs0.src ! h264parse ! avdec_h264 ! mixer.sink_0  \
    vs1.decoder_src ! mixer.sink_1  \
    mixer. ! autovideosink -v
```

vmaf (MR #9757)

```
GObject
 +----GInitiallyUnowned
       +----GstObject
             +----GstElement
                   +----GstAggregator
                         +----GstVideoAggregator
                               +----GstVmaf


Element Properties:

  signal-scores       : Send a signal after calculating a frame score
                        flags: readable, writable
                        Boolean. Default: false

Element Signals:

  "score" :  void user_function (GstElement * object,
                                 gdouble arg0,
                                 gpointer user_data);
```

What to do with all these data?

Multimedia-benchmark

- Supports multi-video benchmarking:
  - Local source
  - Videotestsrc
  - Google Drive sources
- Runs the following encoders in GStreamer:
  - x264, x265, openh264, fluendo LCEVC H.264
- Gathers FFmpeg VMAF results of the previous encoding
- Extracts the bitrate from the encoded file
- Process all results combining the data of all codecs to be analyzed and plotted
- Provides a Jupyter notebook to easily analyze the data

Input yml setup

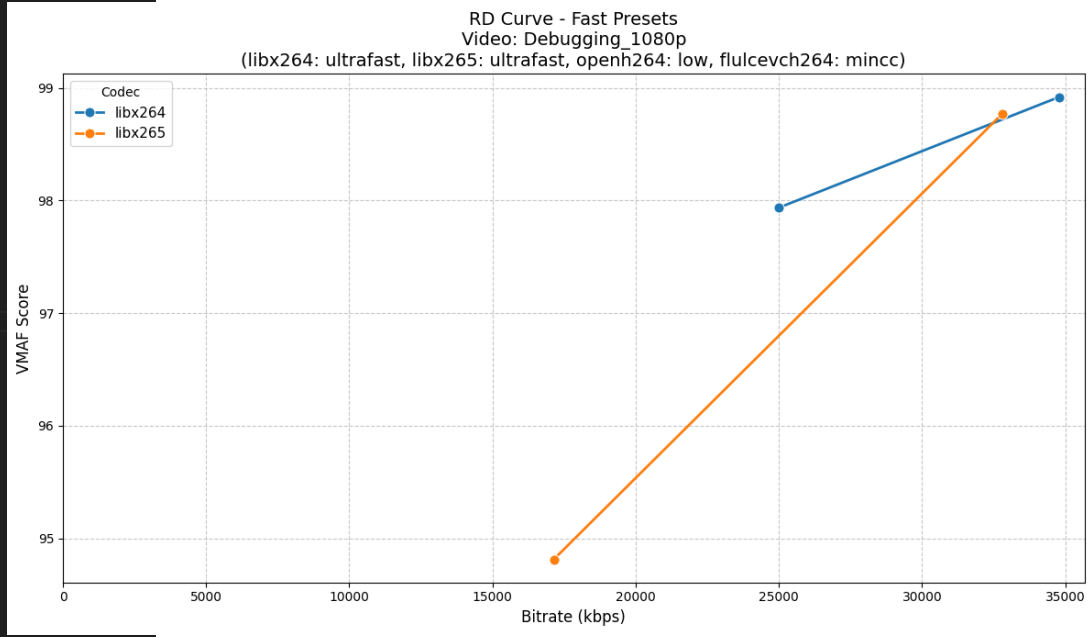
```yaml
experiment_name: "grant_reporting_minimum"
output_path: "results"
credentials_file_path: "/app/data/rdi-2717-dbfe965656a1.json"

source_videos:
  - name: "Wikipedia_1920x1080p30"
    path: "<>"
    resolution: "1920x1080"
    resolution_name: "1080p"
    bit_depth: 8
    color_format: "4:2:0"
    frame_rate: 30
    content_type: "screen"
    owner: "aomctc_v5"
    type: "gdrive"
    description: "https://aomedia.org/docs/CWG-D103o_AV2_CTC_v5.pdf"

  - name: "WalkingInStreet_1920x1080_30fps"
    path: "<>"
    resolution: "1920x1080"
    resolution_name: "1080p"
    bit_depth: 8
    color_format: "4:2:0"
    frame_rate: 30
    content_type: "natural"
    owner: "aomctc_v5"
    description: "https://aomedia.org/docs/CWG-D103o_AV2_CTC_v5.pdf"

tasks:
  - codec: "libx265"
    crf_values: [10, 28, 51]
    preset: ["ultrafast", "medium", "veryslow"]
    config_file: "configs/codecs/x265_settings.cfg"

  - codec: "libx264"
    crf_values: [10, 28, 51]
    preset: ["ultrafast", "medium", "veryslow"]
    config_file: "configs/codecs/x264_settings.cfg"
```



End2end latency

End to end latency

How to measure the end two end latency?

Screen capture device to compare clocks

- Easy way for embedded environments

Problem:

- expensive external devices or manual capture which is setup not very accurate
- only works on environments where network is not involved

End to end latency

How to measure the end two end latency?

Burn QR timestamps

- https://github.com/SNS-JU/6gxr-latency-clock
- There are six timestamps recorded. The relevant one is the sixth. This is the time the frame will be displayed as 64-bit nanoseconds since the unix epoch (realtime).
- Challenges to get those stats
  - Overwrite in the frame the encoded timestamps
  - Signal processing
  - Artifacts injection affects quality metrics

End to end latency

How to measure the end two end latency?

SEI Injection:

- NTP synchronization: GstNTPClock
- Custom metadata structure

```
static const guint8 FLU_TIMING_UUID[] = {0xCF, 0x84, 0x82, 0x78, 0xEE, 0x23, 0x30, 0x6C,0x92, 0x65, 0xE8, 0xFE,
0xF2, 0x00, 0x01, 0x02};

typedef struct
{
   GstClockTime source_time_ns;
   GstClockTime pre_encode_time_ns;
   GstClockTime post_encode_time_ns;
   GstClockTime pre_decode_time_ns;
   GstClockTime post_decode_time_ns;
} FluTimingMetadata;

GstVideoSEIUserDataUnregisteredMeta sei_meta =
gst_buffer_add_video_sei_user_data_unregistered_meta(buffer, (guint8*)FLU_TIMING_UUID,
```

End to end latency

How to measure the end two end latency?

SEI Injection

- Add pad probes for each element to fill the corresponding timestamp
  ((FluTimingMetadata*)sei_meta->data)->pre_encode_time_ns = gst_clock_get_time(m_clock);

- Inject the SEI GST_H264_SEI_USER_DATA_UNREGISTERED

```
sei_data = g_array_new(FALSE, FALSE, sizeof(sei_msg));
g_array_append_vals(sei_data, &sei_msg, 1);
sei_memory = gst_h264_create_sei_memory_avc(4, sei_data);
g_array_unref(sei_data);
g_assert(sei_memory);

new_buffer = gst_h264_parser_insert_sei_avc(m_parser, 4, buffer, sei_memory);
```

End to end latency

How to measure the end two end latency?

SEI Injection

- In the server side: send the encoded data through the network
- In the client side: get the metas after the parser
- Calculate the difference between the timestamps

gst.wasm SEI injection to measure end2end latency

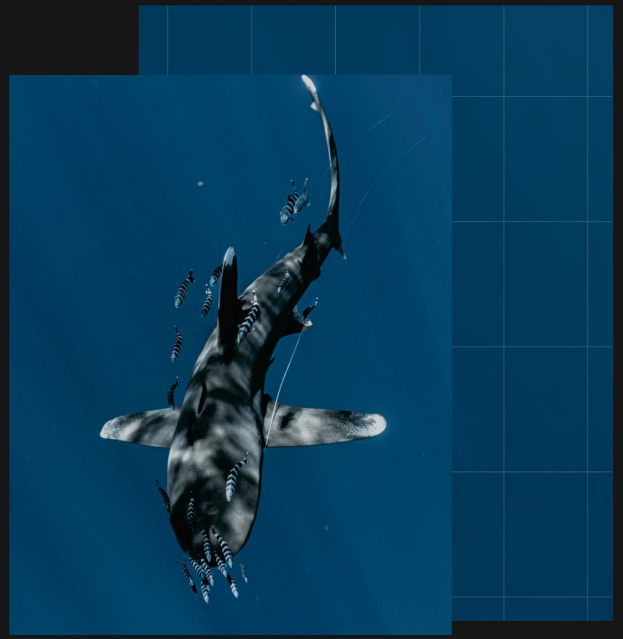
Funded by
the European Union

Elements Console Sources Network Performance Memory

top Filter Default levels

```
end-to-end avg latency       2765 ms                                    app-exa
src_to_pre_encode avg latency    2626 ms                                app-exa
pre_encode_to_post_encode avg latency    37 ms                          app-exa
post_encode_to_pre_decode avg latency    49 ms                          app-exa
pre_decode_to_post_decode avg latency    51 ms                          app-exa
========================================                                app-exa
SEI found: FluTimingMeta{ source_time_ns=1758035400294982735,           app-exa
pre_encode_time_ns=1758035402400405620, post_encode_time_ns=1758035402435930180,
pre_decode_time_ns=1758035402476999936, post_decode_time_ns=1758035402528000000 }
end-to-end avg latency       2741 ms                                    app-exa
src_to_pre_encode avg latency    2603 ms                                app-exa
pre_encode_to_post_encode avg latency    37 ms                          app-exa
post_encode_to_pre_decode avg latency    49 ms                          app-exa
pre_decode_to_post_decode avg latency    51 ms                          app-exa
========================================                                app-exa
SEI found: FluTimingMeta{ source_time_ns=1758035400294982735,           app-exa
pre_encode_time_ns=1758035402436002024, post_encode_time_ns=1758035402465343742,
pre_decode_time_ns=1758035402511000064, post_decode_time_ns=1758035402560000000 }
end-to-end avg latency       2717 ms                                    app-exa
src_to_pre_encode avg latency    2579 ms                                app-exa
pre_encode_to_post_encode avg latency    37 ms                          app-exa
post_encode_to_pre_decode avg latency    49 ms                          app-exa
pre_decode_to_post_decode avg latency    51 ms                          app-exa
========================================                                app-exa
SEI found: FluTimingMeta{ source_time_ns=1758035400294982735,           app-exa
pre_encode_time_ns=1758035402465417525, post_encode_time_ns=1758035402499817104,
pre_decode_time_ns=1758035402547000064, post_decode_time_ns=1758035402593999872 }
```

Next steps

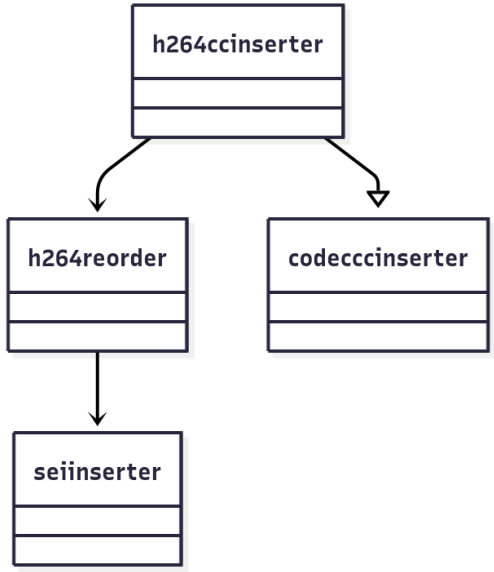


How to evolve?

- SEIs

SEI injection stats to allow network pipelines for metas as UDUS (from manual to SEI Injection: https://gitlab.freedesktop.org/gstreamer/gstreamer/-/issues/3059

  - seiinjector: element to inject UDU from metas in an abstract way. TBD



How to evolve?

- Videostats
  - Use reconstructed frame for VMAF inputs
  - Use SEIs to work within the network

- Multimedia benchmark
  - Integrate and use videostats deeply
  - Compare different setups in a more automatic way

Questions?