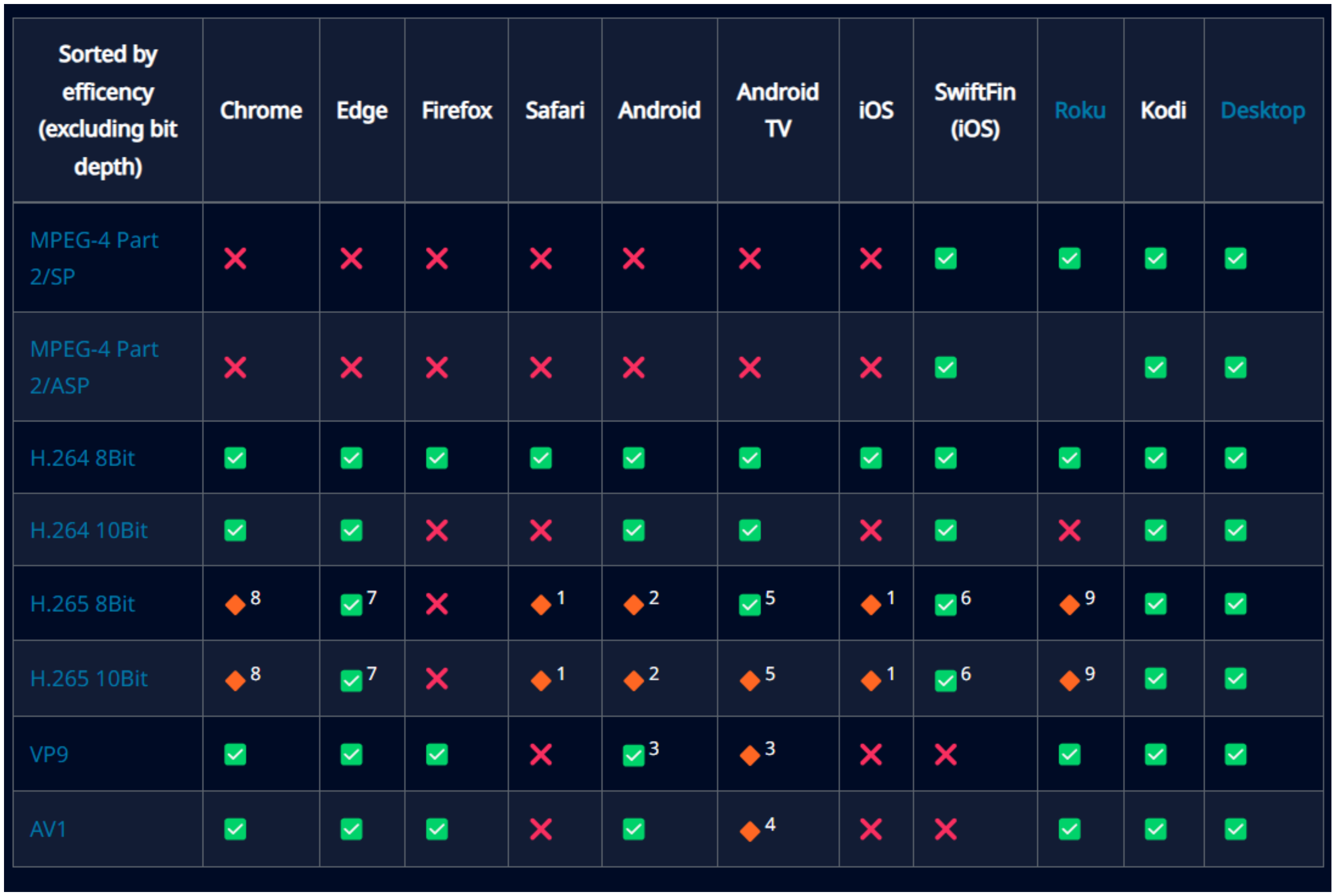
3rd Project Update (2023-11-19)

in this update I will show some of the benchmark results and talk more about the problem we are facing.

Jellyfin codec support



Sorted by efficency (excluding bit depth)	Chrome	Edge	Firefox	Safari	Android	Android TV	iOS	SwiftFin (iOS)	Roku	Kodi	Desktop
MPEG-4 Part 2/SP	✗	✗	✗	✗	✗	✗	✗	✓	✓	✓	✓
MPEG-4 Part 2/ASP	✗	✗	✗	✗	✗	✗	✗	✓		✓	✓
H.264 8Bit	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
H.264 10Bit	✓	✓	✗	✗	✓	✓	✗	✓	✗	✓	✓
H.265 8Bit	◆ [8]	✓ [7]	✗	◆ [1]	◆ [2]	✓ [5]	◆ [1]	✓ [6]	◆ [9]	✓	✓
H.265 10Bit	◆ [8]	✓ [7]	✗	◆ [1]	◆ [2]	◆ [5]	◆ [1]	✓ [6]	◆ [9]	✓	✓
VP9	✓	✓	✓	✗	✓ [3]	◆ [3]	✗	✗	✓	✓	✓
AV1	✓	✓	✓	✗	✓	◆ [4]	✗	✗	✓	✓	✓

The table above shows us the codecs supported in each of the platforms, as we can see H.264 8bit format is supported on all platforms but H.265 has limited support. So it is safe to assume that there will be a transcoding from H.265 to H.264 every time we play a video on a device that does not support H.265 decoding; this approach works well as long as the server has powerful GPU and it is not being used for other demanding services at the same time; but most home servers are either using integrated graphics or have very low-end and old GPUs that probably does not support H.265 decoding and has to rely on software decoding. This means some compromisation has to be done during the transcoding to keep up with the vide playback. In my server this is the difference between direct playback and transcoded playback:

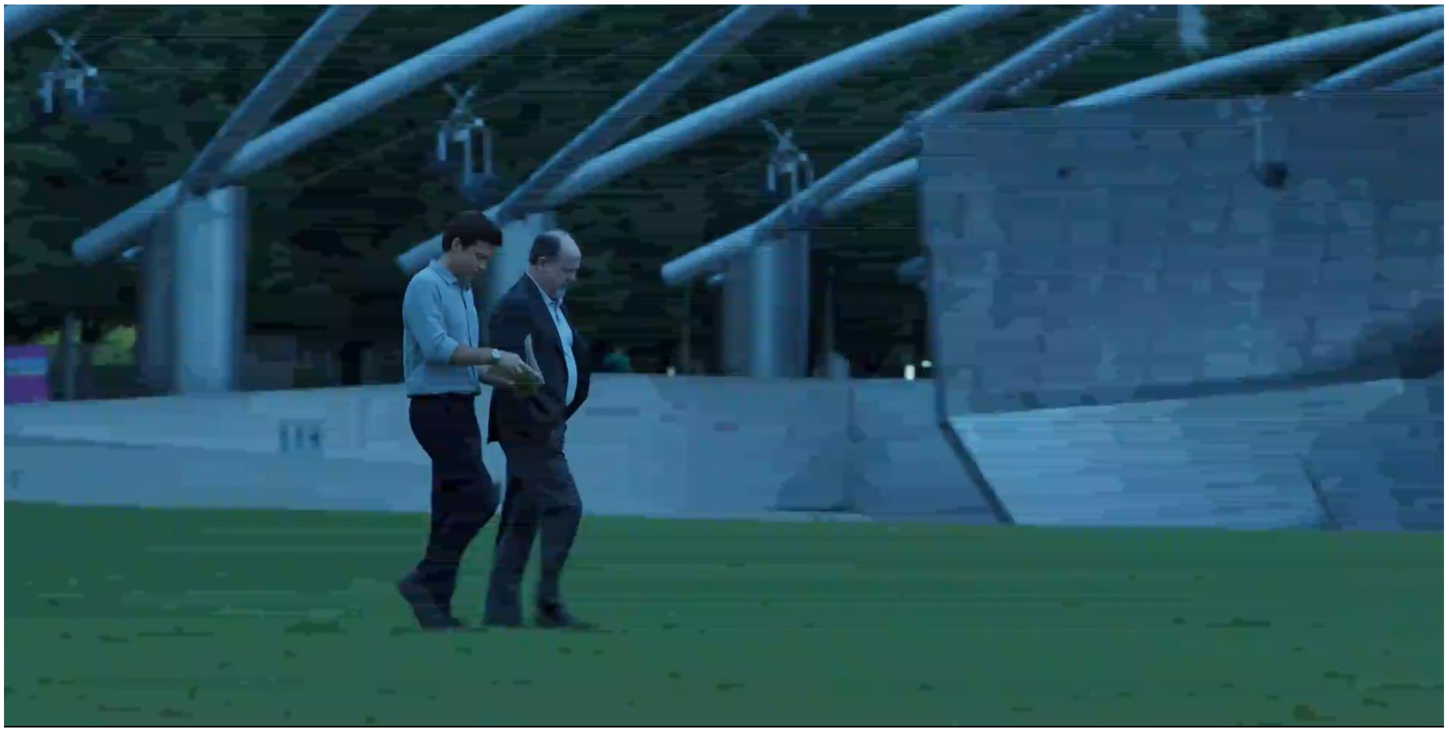


Figure 1 - Transcoded Video Playing on Firefox

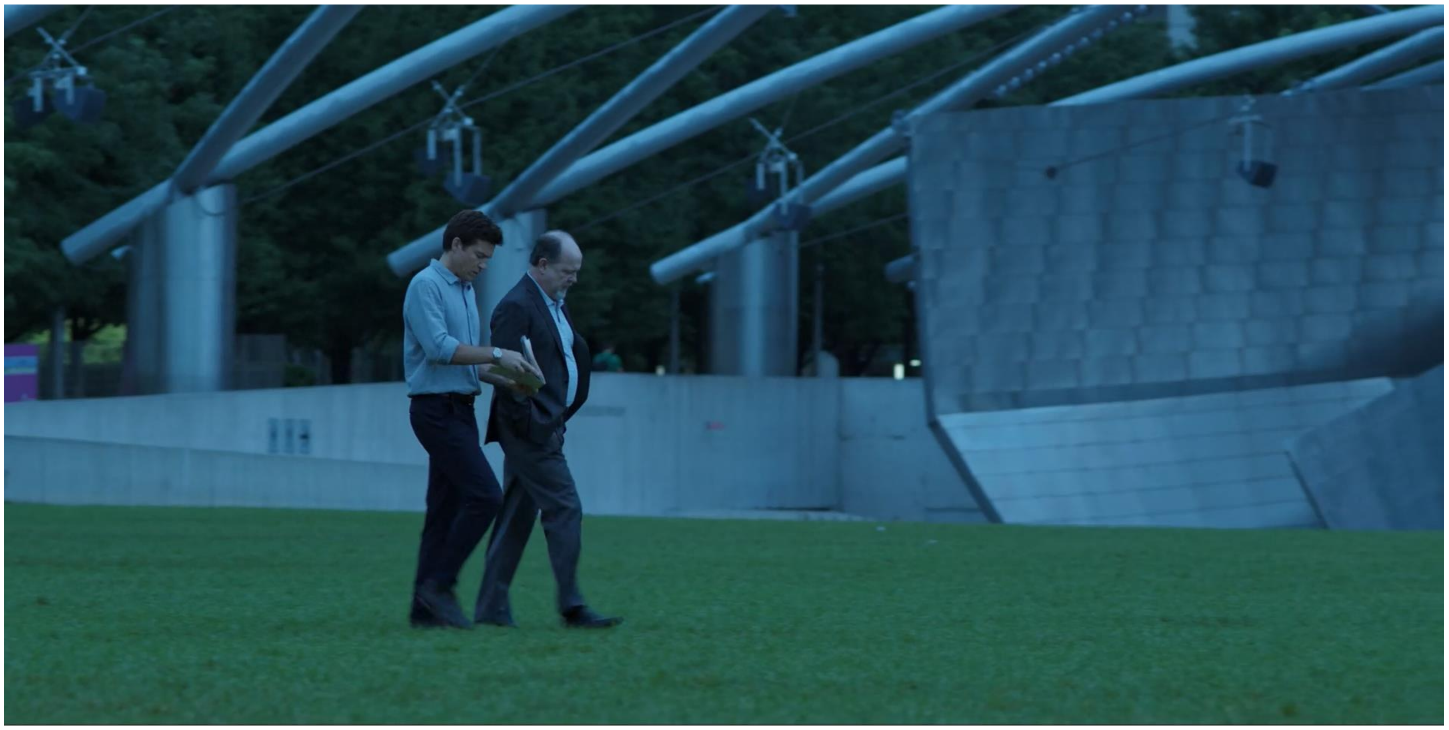


Figure 2 - Direct Play

Both videos are hosted on the same server, the transcoding is done by Intel Sandybridge GPU using VAAPI on Linux.

Now compare this to the ffmpeg output I got from following command:

```
ffmpeg -vaapi_device /dev/dri/renderD128 -i test-1080p-h265.mkv -vf 'format=nv12,hwupload' -c:v
h264_vaapi -preset ultrafast -b:v 5000k -c:a copy test-1080p-h264.mkv
```

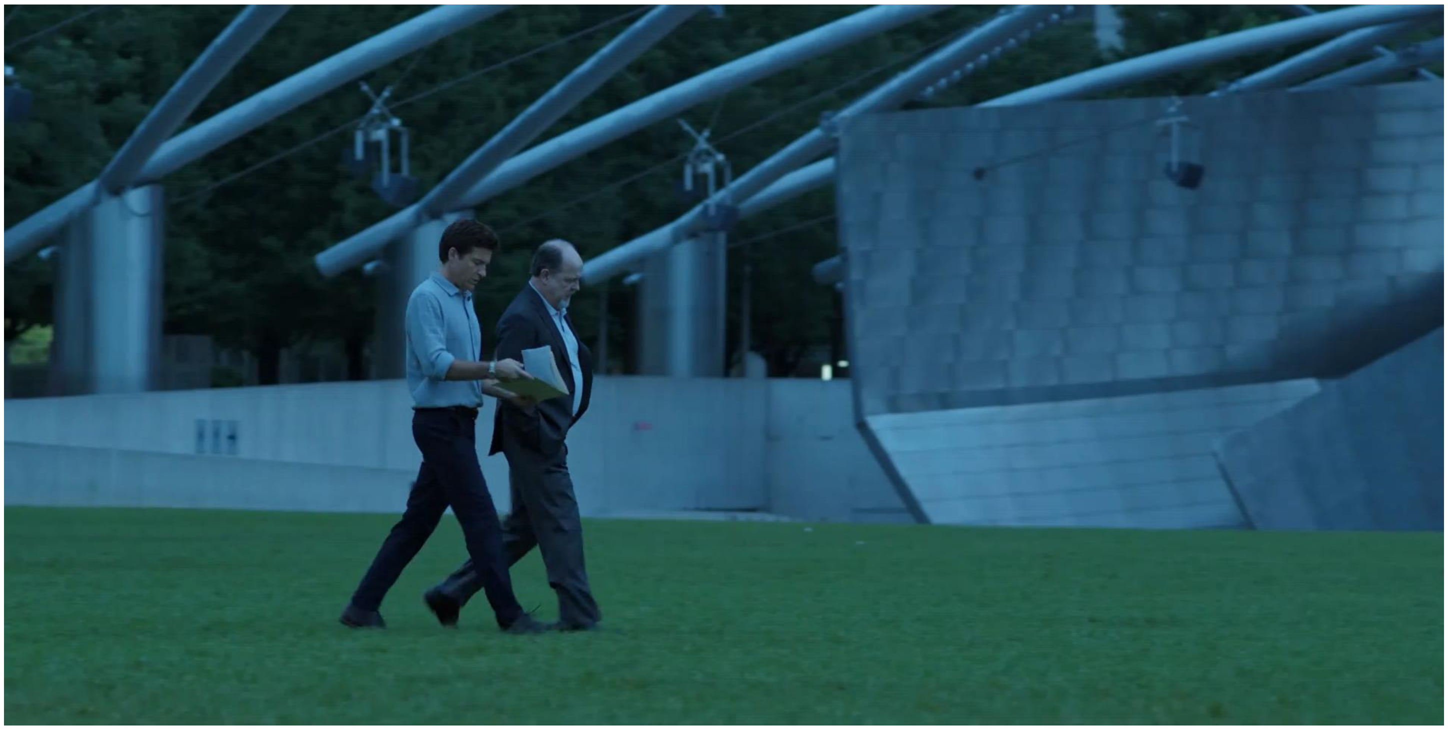
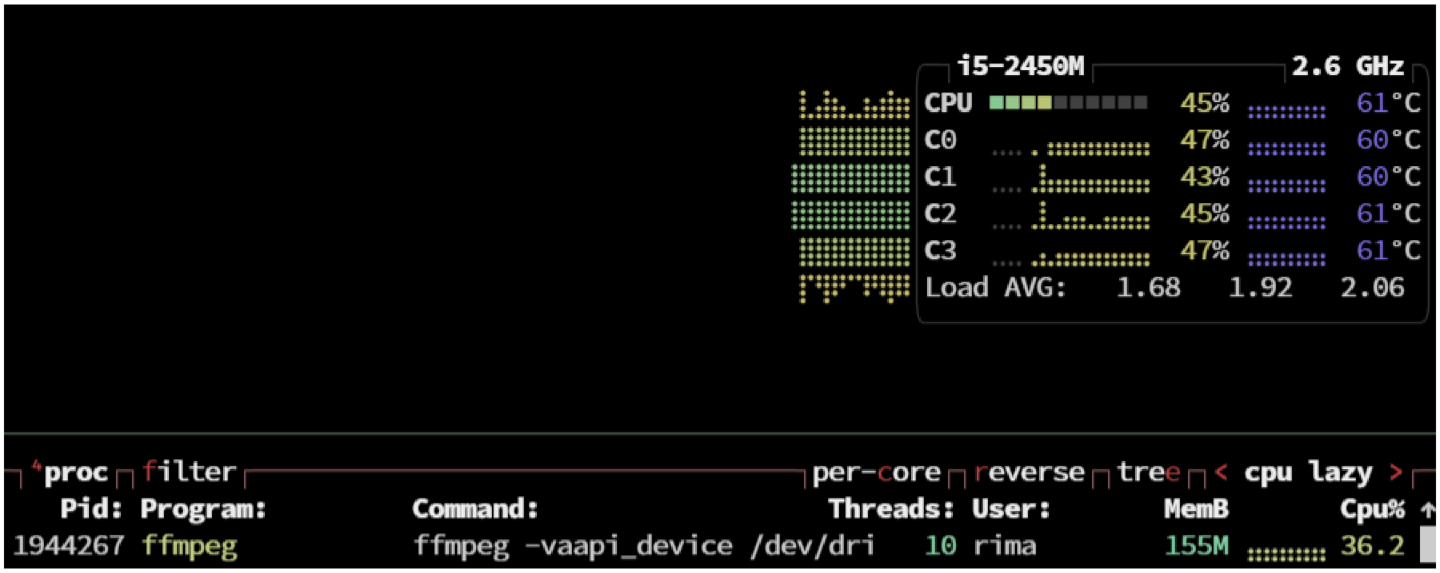


Figure 3 - Manually Transcoded

Not as good as direct play but much better than Jellyfin transcoding, lets look at the resource usage:

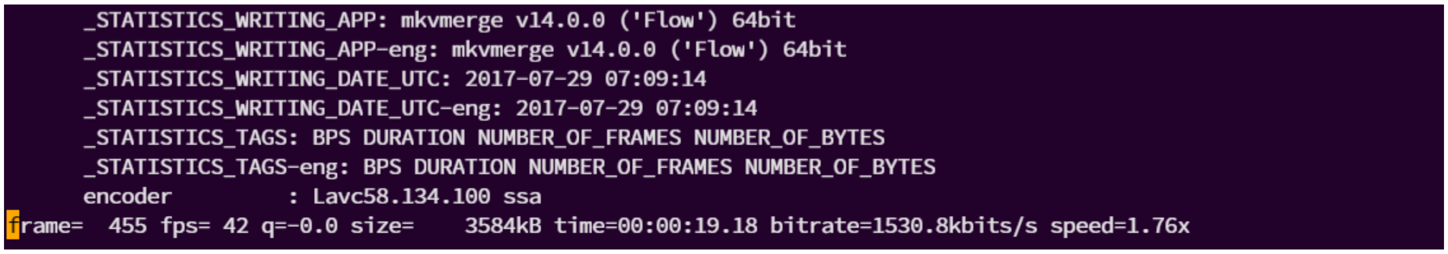


```
                               i5-2450M                 2.6 GHz
                         CPU ▮▮▮▮▮▮▮▮▮▮      45%            61°C
                         C0                  47%            60°C
                         C1                  43%            60°C
                         C2                  45%            61°C
                         C3                  47%            61°C
                         Load AVG:    1.68    1.92    2.06

 ⁴proc  filter                               per-core  reverse  tree  < cpu lazy >
  Pid: Program:       Command:                   Threads: User:     MemB        Cpu%
1944267 ffmpeg        ffmpeg -vaapi_device /dev/dri   10  rima      155M        36.2
```

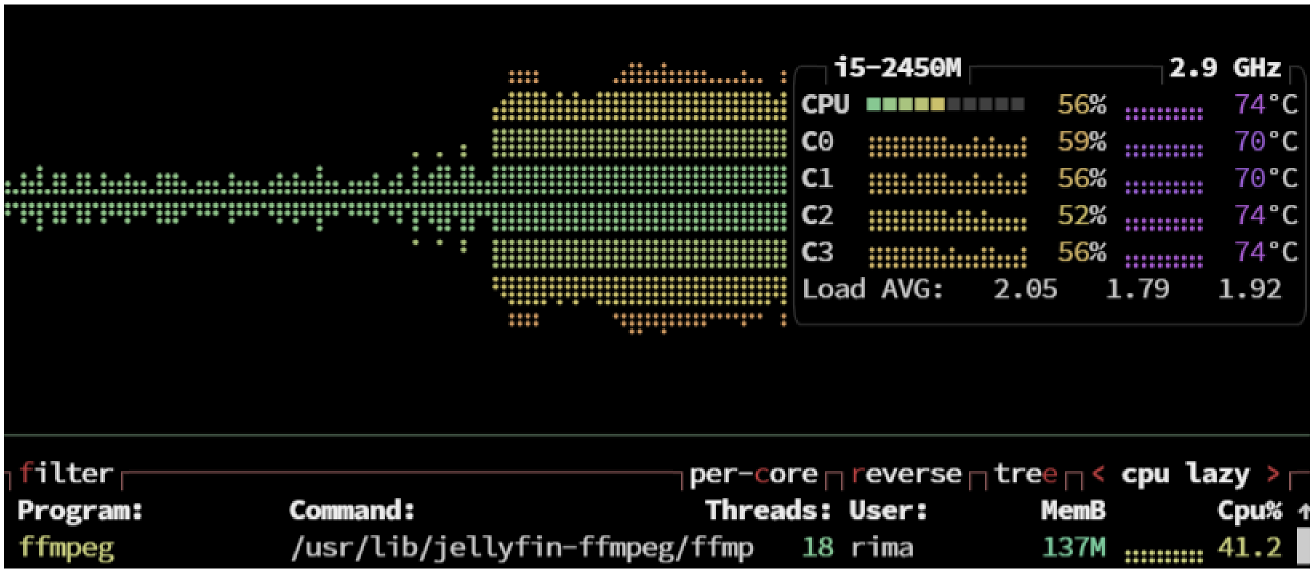
Figure 4 - Resource monitor during manual transcoding

And transcoding speed:



```
      _STATISTICS_WRITING_APP: mkvmerge v14.0.0 ('Flow') 64bit
      _STATISTICS_WRITING_APP-eng: mkvmerge v14.0.0 ('Flow') 64bit
      _STATISTICS_WRITING_DATE_UTC: 2017-07-29 07:09:14
      _STATISTICS_WRITING_DATE_UTC-eng: 2017-07-29 07:09:14
      _STATISTICS_TAGS: BPS DURATION NUMBER_OF_FRAMES NUMBER_OF_BYTES
      _STATISTICS_TAGS-eng: BPS DURATION NUMBER_OF_FRAMES NUMBER_OF_BYTES
      encoder         : Lavc58.134.100 ssa
frame=  455 fps= 42 q=-0.0 size=    3584kB time=00:00:19.18 bitrate=1530.8kbits/s speed=1.76x
```

Compare it to Jellyfin transcoding:

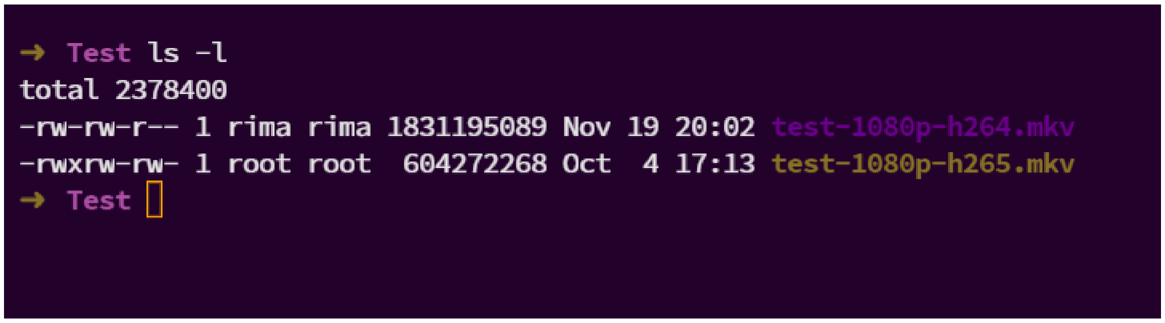


```
                                              i5-2450M              2.9 GHz
                                              CPU              56%           74°C
                                              C0               59%           70°C
                                              C1               56%           70°C
                                              C2               52%           74°C
                                              C3               56%           74°C
                                              Load AVG:   2.05   1.79   1.92

filter                                  per-core reverse tree < cpu lazy >
Program:        Command:                    Threads: User:      MemB       Cpu%
ffmpeg          /usr/lib/jellyfin-ffmpeg/ffmp  18 rima          137M       41.2
```

Figure 5 - Resource monitor during Jellyfin transcoding

The results are clearly indicating that Jellyfin is not aiming for efficiency or quality while transcoding, but it does transcode much faster. This makes sense since a playing movie is better than interrupted one. But, this is also raises the question, why not just transcode beforehand? Let's see the size difference between two files:



```
➜  Test ls -l
total 2378400
-rw-rw-r-- 1 rima rima 1831195089 Nov 19 20:02 test-1080p-h264.mkv
-rwxrw-rw- 1 root root  604272268 Oct  4 17:13 test-1080p-h265.mkv
➜  Test 
```

Figure 6 - Original and converted file size

600Mb vs 1.8 Gb and I should point out that H.264 file is only half the length of the full movie which is in h265 file (I stopped the transcoding halfway since it was taking too long).

So, we are forced to choose between compatibility and space, at least until all devices start supporting h265.

What is the solution, what is next:

There is no straight forward solution, this project intended to provide an improvement idea to Jellyfin. This can be an integrated on Jellyfin or developed as an add-on. The final update will be all about this solution but I will briefly mention it again.

Since we can achieve a better quality video with less pinning on resource, why not covert before hand, this does not have to be always h265 to h264 transcode, users can have the free will to choose what they want to optimize their content for (space or compatibility, quality or responsiveness). An even smarter solution would be smart choice of content to transcode (more on this later).