

Construction d'une caméra IP de surveillance et d'un serveur d'enregistrement en boucle.

Un projet HUB Epitech de Maxence Abela, étudiant en 5e année,
Cycle Expertise et Innovation.

15 janvier 2023

Introduction

L'objectif de ce projet est de produire un système de vidéo-surveillance à partir du matériel inutilisé à ma disposition, à savoir une raspberry Pi 3B+, une webcam Full HD que j'ai acheté pour moins de 15€ pour les besoins de mon projet EIP maintenant terminé, et un PC qui me sert de serveur multi-usages avec des capacités de calcul non utilisées.

Mes critères pour ce projet sont les suivants:

- Le flux vidéo issu de la caméra doit être enregistré en continu sur un serveur et doit pouvoir être consulté en temps-réel depuis une machine sur le même réseau.
- Les enregistrements doivent être supprimés au bout de 48h, et les 48 heures conservées doivent occuper un espace réduit sur le disque afin de limiter son usure et de conserver de l'espace pour d'autres usages.
- Le système doit être facile à déployer à l'aide de scripts prévus à cet effet, et il doit redémarrer automatiquement en cas de redémarrage du serveur ou de la caméra.
- Le système n'a pas besoin d'être publiquement accessible.



Capture de la vidéo via la webcam

Métdode de capture

Plusieurs solutions existent pour capturer un flux vidéo depuis une Raspberry Pi et le transférer via le réseau. J'ai choisi **v4l2rtspserver**, mais je vais rapidement couvrir les autres pour justifier mon choix.

J'ai exploré les options listées sur ce site:

<https://www.magdiblog.fr/divers/raspberry-pi-camera-5-facons-de-faire-du-streaming/>

- Netcat avec MPlayer a bien fonctionné chez moi mais en FullHD à 20fps, j'avais une latence de plus de 10 secondes, même problème avec VLC. Ces solutions ne me permettaient pas non plus de facilement enregistrer la vidéo sous forme de fichiers, et je n'ai pas trouvé d'options pour faire de l'encodage en temps-réel, mais comme j'avais déjà 10 secondes de délai, je n'ai pas cherché très longtemps.
- Je ne souhaite pas passer par des chaînes de jpg avec mjpeg-streamer, car je veux obtenir à la fin des vidéos fluides, et me retrouver avec des dossiers contenant des milliers d'images m'embêterait plus qu'autre chose. Mais je note que ça serait une option correcte si l'objectif du projet avait été de construire une interface web présentant le flux "streaming", car des images qui se rafraîchissent à haute fréquence sont bien plus faciles à intégrer qu'un véritable lecteur vidéo dans une page web. Si je décide de faire ça un jour, je me contenterai de "piocher" des images dans le flux vidéo pour les afficher sur le serveur.

Après des recherches additionnelles, j'ai découvert que la plupart des caméras de surveillance du marché utilisent le protocole RTSP (Real Time Streaming Protocol) pour envoyer leur flux vidéo à un enregistreur ou un lecteur. Ce protocole permet l'envoi de différents formats vidéo, il permet d'envoyer des commandes PTZ (pan, tilt, zoom) aux caméras qui sont équipées de moteurs. Il consiste en un serveur auquel plusieurs machines (authentifiées ou non) peuvent se connecter pour accéder au flux vidéo, VLC et d'autres lecteurs vidéo libres sont compatibles avec RTSP et peuvent se connecter à de tels serveurs. J'ai décidé d'utiliser cette technologie pour ce projet, car si plus tard je souhaite ajouter de vraies caméras au système, j'aurai déjà un enregistreur compatible sous la main. Ces caméras, en passant, me coûteraient moins cher que le couple Raspberry + Webcam que je n'utilise que parce que je les ai déjà chez moi.

Sous linux, la meilleure solution pour capturer la sortie d'une caméra et mettre en place un serveur RTSP est de passer par le logiciel **v4l2rtspserver**, qui utilise la collection de drivers **video4linux** (d'où "v4l2").

Installation et configuration de v4l2rtspserver

Le script **install-camera.sh** est disponible sur le dépôt github lié à ce projet pour installer les dépendances et démarrer le service nécessaire au fonctionnement de la caméra.

La compilation pour moi a pris beaucoup de temps la première fois, j'ai dû attendre 10 minutes sur ma Raspberry Pi 3 la première fois, ça serait allé plus vite sur une Pi 4, et j'aurais pu compiler en exécutant make -j4 afin de compiler en utilisant les 4 coeurs disponibles du processeur pour diviser par 4 le temps de compilation, ce que j'ai fait la fois suivante quand j'ai dû recompiler v4l2rtspserver avec les bibliothèques audio pour que le son soit enregistré en même temps que la vidéo.

En effet, je me suis rendu compte que je n'enregistrais pas de son alors qu'un micro était présent sur la webcam dont je me servais. J'ai fini par remonter à l'origine du problème, qui était que j'avais compilé v4l2rtspserver sans les bibliothèques audio nécessaires (que j'ai ajouté par la suite dans mon script d'installation: **libalsaplayer-dev**, **libclalsadrv-dev**, et **libdssialsacompat-dev**)

Solution pour le son ici :

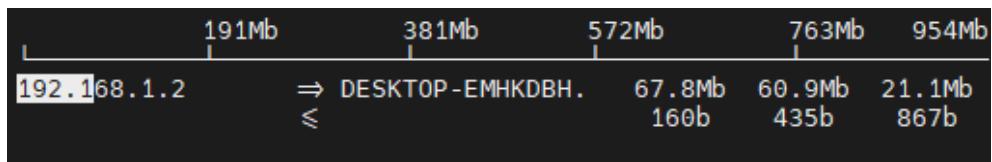
<https://github.com/mpromonet/v4l2rtspserver/issues/100#issuecomment-392247426>

v4l2rtspserver propose beaucoup d'options pour configurer son fonctionnement, mais celles qui nous intéressent sont -W, -H, et -F, respectivement pour configurer la largeur et la hauteur en pixels de l'image capturée, ainsi que le nombre d'images par secondes. Les autres options peuvent-être laissées à leur valeur par défaut pour ce projet.

Je le lance donc avec les options -W 1920 -H 1080 -F 30 pour un largeur d'image de 1920 pixels, une hauteur de 1080 pixels, et 24 images par seconde.

```
v4l2rtspserver -W 1920 -H 1080 -F 30
```

En le laissant tourner pendant 10 minutes et en bougeant la caméra, iftop m'indique une vitesse de transfert maximale ne dépassant pas les 80Mbps. Je peux le réduire à 68 Mbps (max) en changeant le framerate à 24, ce qui reste correct visuellement. C'est beaucoup mais ça reste restreint pour un réseau local, et on peut sans problème le faire transiter via WiFi avec un routeur récent et un Raspberry pi 4. La Pi 3 n'étant pas capable d'atteindre un tel débit en Wifi.



Capture d'écran de l'outil iftop pendant un stream vidéo 24fps sur la Raspberry Pi 3B+.

La première ligne comporte les vitesses d'upload maximales, moyenne, et actuelle, dans cet ordre.

La commande que j'ai donc retenu pour démarrer le flux vidéo après mes essais fut la suivante:

```
v4l2rtspserver -W 1920 -H 1080 -F 24
```

On peut alors d'ores et déjà accéder au flux vidéo depuis une machine connectée au même réseau en accédant à rtsp://IPAddress:8554/unicast via VLC dans Media->Ouvrir un flux réseau...

C'est comme ça que je compte accéder au flux vidéo en temps réel par la suite si j'en ai besoin car plusieurs machines peuvent écouter simultanément sur ce flux vidéo. Il faut noter cependant que des images seront perdues avec les paramètres actuels si 2 machines y accèdent en même temps, car le débit ethernet maximal de la Raspberry Pi 3 B+ est de 100 Mbps et le débit souhaité avec 2 machines écoutant le stream peut atteindre les 120 Mbps.

Encodage de la vidéo à bord de la Raspberry

Vous noterez que je n'ai pas encore parlé d'encodage jusque là si vous êtes familier avec le sujet, cela peut sembler étrange. Ne vous en faites pas j'y viens, justement ça aurait pu aider avec le problème de débit que j'ai mentionné plus haut, mais mes recherches m'ont appris que ça n'était pas envisageable sur ma Raspberry, qui serait tout juste capable de décoder un flux h264 en 1080p grâce à son décodeur hardware, mais pas de l'encoder. Le CPU, lui, est loin d'être capable de le faire à plus de 2 images par seconde. Vous pouvez jeter un œil à cette réponse dans les forums officiels Raspberry pi pour plus d'informations à ce sujet.

<https://forums.raspberrypi.com/viewtopic.php?p=1245719#p1245719>

En revanche, si vous souhaitez reproduire ce projet et que vous avez à disposition une Raspberry Pi 4, l'encodage devient tout à fait abordable, et je vous recommande chaudement de modifier les scripts pour le faire à bord de la Raspberry plutôt qu'à l'enregistrement.

Enregistrement de la vidéo

Ici aussi, plusieurs solutions semblent exister, VLC est capable d'enregistrer le flux vidéo, mais il existe des solutions plus efficaces. Si possible, j'aimerais éviter de monopoliser le processeur de mon serveur, car il a d'autres travaux à accomplir, et j'aimerais éviter de remplir mon disque trop vite pour seulement 48 heures d'enregistrement.

Comme je le disais, je peux atteindre des vitesses de transfert de 80Mbps, c'est un trafic que je suis prêt à accepter sur mon réseau, mais pas s'il faut le cumuler sur mon disque dur. Car actuellement, l'espace nécessaire pour 48 heures de vidéo serait de $80 * 3600 * 48 = 1.38\text{To}$ ce qui est beaucoup trop, j'aimerais si possible diviser au moins par 100 l'espace nécessaire pour stocker les vidéos. Donc je vais chercher un moyen de capturer le flux RTSP, de l'encoder sans trop charger mon CPU, et de le stocker sous la forme de fichiers de 10 minutes de tailles réduites. Le meilleur outil pour accomplir cette tâche semble être ffmpeg, un véritable couteau suisse des flux vidéos, capable de traiter toutes sortes de flux vidéos et de le convertir en toutes autres sortes de flux vidéos.

```
sudo apt install ffmpeg # Installation de ffmpeg
```

```
ffmpeg \
-i rtsp://192.168.1.2:8554/unicast \
-c copy -map 0 \
-c:a aac \
-f segment -segment_time 15 -segment_format mp4 \
record/capture-%03d.mp4
```

Cette commande m'a généré comme prévu de gros fichiers d'une centaine de Mo pour seulement 15 secondes de contenu, car elle se contente de copier sans le modifier le contenu vidéo dans des fichiers mp4 de 15 secondes numérotés à partir de 0 depuis le lancement de la commande. Seul le son est encodé en utilisant le codec aac, mais il n'y a pas grand chose à dire dessus car l'encodage audio a un impact négligeable sur le CPU par rapport à la vidéo.

J'ai aussi relevé des erreurs dans le log de ffmpeg qui étaient dues à mon utilisation de mp4 comme format de sortie. En effet, les fichiers MP4 ont un header qui décrit leur contenu et nécessite des calculs supplémentaires lors de la création du fichier pour renseigner sa taille, or comme on traite un flux continu, on ne connaît pas sa taille à l'avance, et ffmpeg doit régulièrement s'arrêter pour finaliser la création du fichier avant de passer au suivant. Ma solution fut d'utiliser le format flv à la place, qui n'a pas ce problème de header et est donc plus adapté à de l'enregistrement continu. J'en ai profité pour changer le format de sortie et indiquer la date et l'heure de création de chaque fichier dans leur nom.

```
ffmpeg \
-i rtsp://192.168.1.2:8554/unicast \
-c copy -map 0 \
-c:a aac \
-f segment -segment_time 15 -segment_format flv \
record/enregistrement_%d-%m-%Y_%H-%M-%S.flv
```

Encodage h264 à la volée

Dans le domaine de la vidéo (pour la musique aussi), lorsqu'on stocke du contenu, on cherche souvent à le compresser, car la vidéo peut vite demander des Téraoctets de mémoire. Pour ça on utilise des codecs (mot-valise pour désigner encoder/decoder en anglais). Plusieurs codecs vidéo existent, les plus communs sont h264, et plus récemment h265 (qui n'est pas encore supporté par la majorité des appareils). L'objectif de ces codecs est de compresser des vidéos plus ou moins efficacement avec ou sans pertes de qualité en fonction des paramètres qui leur sont fournis et à l'aide de puissants algorithmes de compression. Ces codecs exploitent notamment la présence d'un fond immobile pour gagner de l'espace en ne ré-encodant pas le fond dans chaque image, et en ne codant que les pixels qui changent dans le fichier résultant. C'est évidemment plus complexe que ça, mais ça permet de se faire une idée de leur efficacité. En acceptant des pertes et avec une vidéo peu mobile, ces codecs peuvent diviser par 100 ou 200 l'espace ou la bande passante occupée par une vidéo brute.

Mes critères pour configurer l'encodage sont les suivants:

- Je peux accepter une compression avec pertes, mais je souhaite les limiter au minimum.
- Je veux limiter l'impact du processus de compression sur le CPU de mon serveur, ses 4 cœurs doivent pouvoir entamer des tâches intensives sans impacter l'encodage. Je surveillerai leur état avec **htop**.
- J'aimerais limiter l'espace occupé par 48h de vidéo à moins de 200 Go si possible.
- L'encodage peut créer un délai lors de l'enregistrement, mais il doit toujours se faire plus vite que la vidéo. J'entends par là que je veux une vitesse de traitement supérieure à la vitesse réelle de la vidéo.
- Je ne peux pas utiliser l'encodeur hardware embarqué sur la carte graphique intégrée au i5 du serveur, car il est déjà utilisé pour un autre service dans mon cas.

```
ffmpeg
-i rtsp://192.168.1.2:8554/unicast \
-c:v libx264 -preset ultrafast -crf 17 \
-c:a aac \
-f segment -segment_time 15 -segment_format flv \
records/enregistrement_%d-%m-%Y_%H-%M-%S.flv
```

Comme je l'ai dit, ffmpeg est un véritable couteau suisse, on peut sélectionner plusieurs codecs installés par défaut, et des centaines de codecs peuvent-être utilisés comme plug-in avec ffmpeg. H264 propose plusieurs options, dont le détail peut être retrouvé dans la documentation de ffmpeg [ici](#). Les paramètres qui nous intéressent ici pour ajuster la qualité de la compression, la taille des fichiers résultats (bitrate), et la vitesse d'encodage sont **-crf** et **-preset**.

- **-crf** (Constant Rate Factor) correspond à un bitrate cible dont ffmpeg va chercher à se rapprocher en sacrifiant la qualité de l'image si nécessaire. Sa valeur est comprise **entre 0 et 51**, où 0 correspond à un encodage sans pertes, et 51 correspond à la pire qualité possible mais un volume très réduit de données. Les

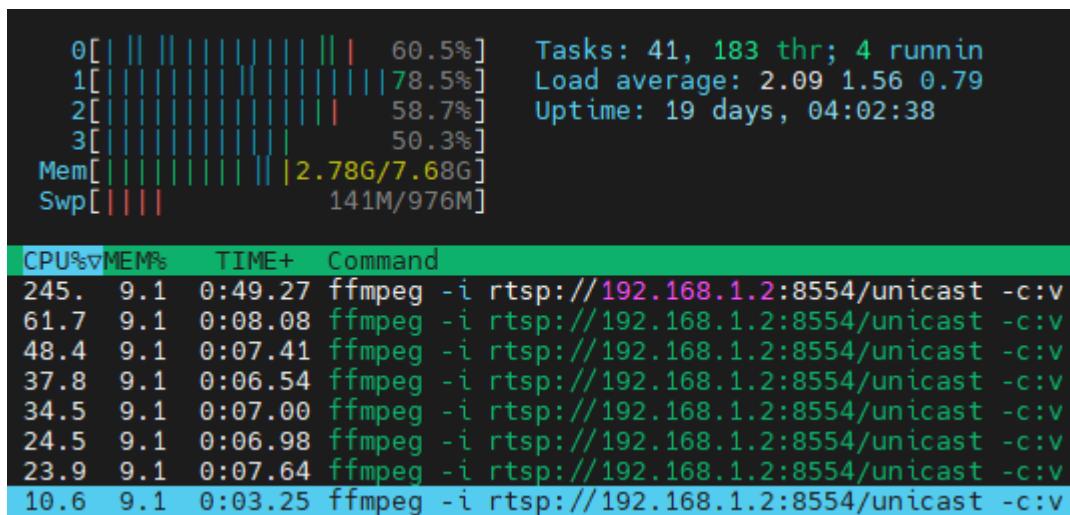
développeurs considèrent qu'une valeur comprise entre 17 et 28 permet de conserver une bonne qualité de visionnage. Les forums qui parlent d'encodage pour des films estiment qu'il faut éviter de monter au dessus d'un crf de 19 pour conserver une bonne qualité visuelle. Dans le cas de la vidéo-surveillance, je n'ai pas besoin d'une qualité digne d'un film, j'ai juste besoin que la perte de qualité soit faible en comparaison avec la qualité originale. Donc j'ai procédé à des tests en modifiant la valeur de crf jusqu'à ce que j'observe des défauts de qualité dans l'image par rapport à la vidéo brute obtenue en temps réel via VLC. Dans mon cas, j'ai pu pousser le CRF jusqu'à 30, mais je vais le laisser à **25**, car le réglage du preset aura un impact sur la qualité visuelle aussi.

- **-preset** correspond à des ensembles de paramètres prédéfinis par les développeurs du codec h264 qui permettent de régler la vitesse d'encodage en sacrifiant le bitrate final. Preset prend 9 valeurs de **veryslow** à **ultrafast**. Ainsi, un encodage avec un preset **slow** permettra d'obtenir des fichiers plus légers au prix d'un encodage plus lent et probablement d'une grande charge du CPU. Et un encodage avec un preset **fast** permettra d'alléger la charge du CPU (c'est mon hypothèse) et d'atteindre une vitesse supérieure à la vitesse de la vidéo. Mais le système n'est pas parfait, et il peut y avoir un impact sur la qualité visuelle. C'est pourquoi j'ai gardé de la marge avec la valeur du **crf** plus tôt. J'ai testé les presets à partir de **slow** en allant vers les valeurs plus rapide, en surveillant la vitesse de l'encodage, la charge du CPU avec **htop**, et le rendu vidéo qui peut être saccadé sans que le log de ffmpeg n'indique de vitesse trop lente.

Tests de presets

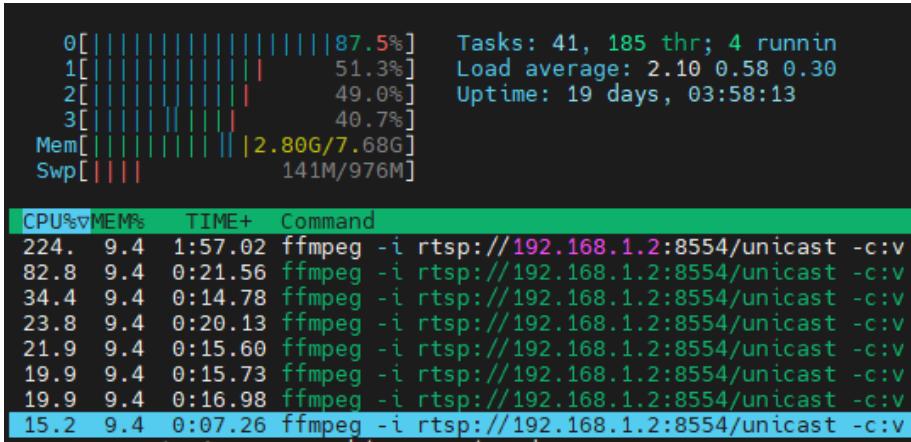
Comme dit précédemment, l'objectif de ces tests est de trouver un preset le plus lent possible à partir duquel la charge du CPU ne dépasse que rarement les 50%. Je souhaite le preset le plus lent, car plus il est rapide, plus les fichiers générés seront larges.

1) Crf 25 et preset Slow



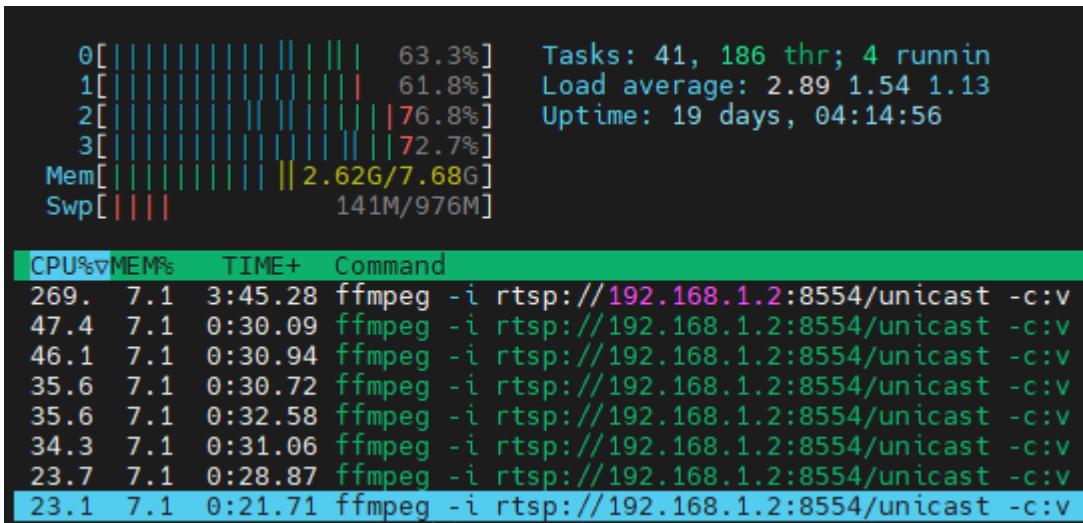
La vitesse de l'encodage descend par moment à 0.79, pas besoin d'aller vérifier la vidéo, je sais déjà que ça sera saccadé.

2) Crf 25 et preset Medium



La vitesse de l'encodage est descendue au pire à 1.01x d'après ffmpeg, et la charge du CPU reste trop grande à mon goût. J'ai quand même consulté les vidéos générées, et j'ai constaté des arrêts de l'image par moment. Il faut donc augmenter le preset.

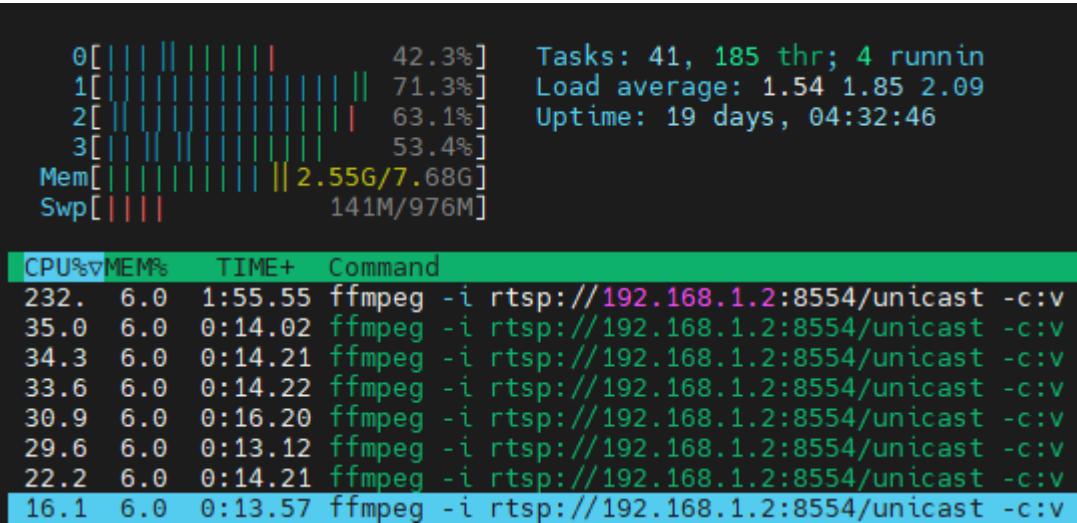
3) Crf 25 et preset Fast



Je n'ai pas observé de saut d'image cette fois. Mais étrangement, le CPU est aussi chargé, voire plus que quand j'étais en preset slow ou medium, mon hypothèse de décharge du CPU n'est pas vraiment correcte finalement. En revanche, je note que la charge du CPU redescend quand il ne se passe rien en face de la caméra. En revenant sur les autres presets, j'ai observé la même chose. D'ailleurs, le volume des fichiers générés varie et peut aller jusqu'à tripler entre un fichier où il ne se passe rien devant la caméra, et un autre où je souffle la fumée de ma cigarette électronique devant (grande **entropie** des pixels capturés par la caméra due à la fumée et donc difficulté pour le compresser efficacement).

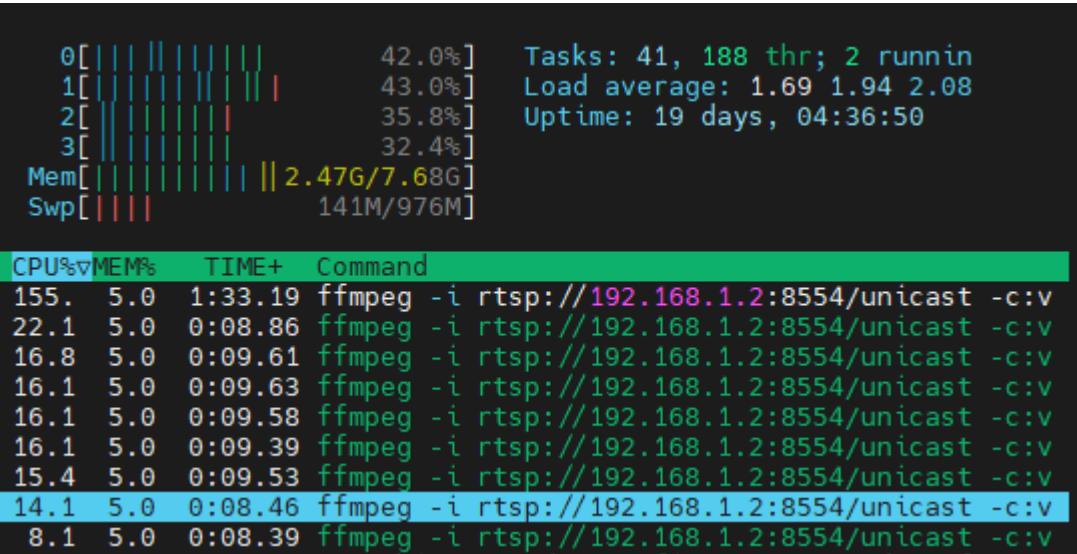
Remarque sur la compression et l'entropie: L'entropie du contenu que l'on cherche à compresser a un grand impact sur le ratio de compression atteignable. Une suite d'octets parfaitement aléatoire et quasiment incompréhensible, car on ne peut pas identifier de relation entre eux pour les exprimer sous la forme d'une fonction, et on ne peut pas les factoriser pour les regrouper. Alors qu'une vidéo avec des pixels qui ne varient pas ou peu et des mouvements relativement prévisibles est tout à fait compressible.

4) Crf 25 et preset faster



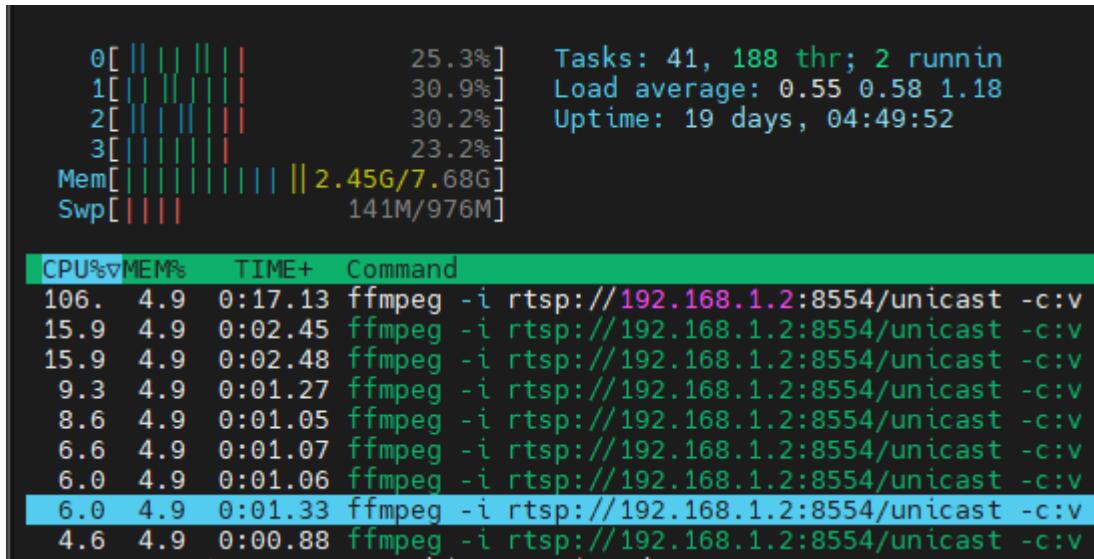
Je note enfin ici une utilisation du CPU plus réduite qu'avec les presets essayés avant. Mais ça reste trop pour moi. À ce niveau, la taille des fichiers pour 15 secondes de vidéo varie entre 4 et 8 Mo.

5) Crf 25 et preset veryfast



Enfin ici, je rencontre une utilisation réduite du CPU, avec tous les coeurs entre 30 et 40% d'utilisation en moyenne, ce que je suis tout à fait prêt à accepter. La taille des fichiers générés pour 15 secondes varie entre 9 et 20 Mo si ça se maintient pour des fichiers de 10 minutes, on arrivera **pour 48h à 230 Go** au maximum et plus probablement autour de la moitié. Mais maintenant qu'on est arrivé là et que je n'ai pas constaté de pertes de qualité, je vais effectuer un dernier test en augmentant le **crf** à 30.

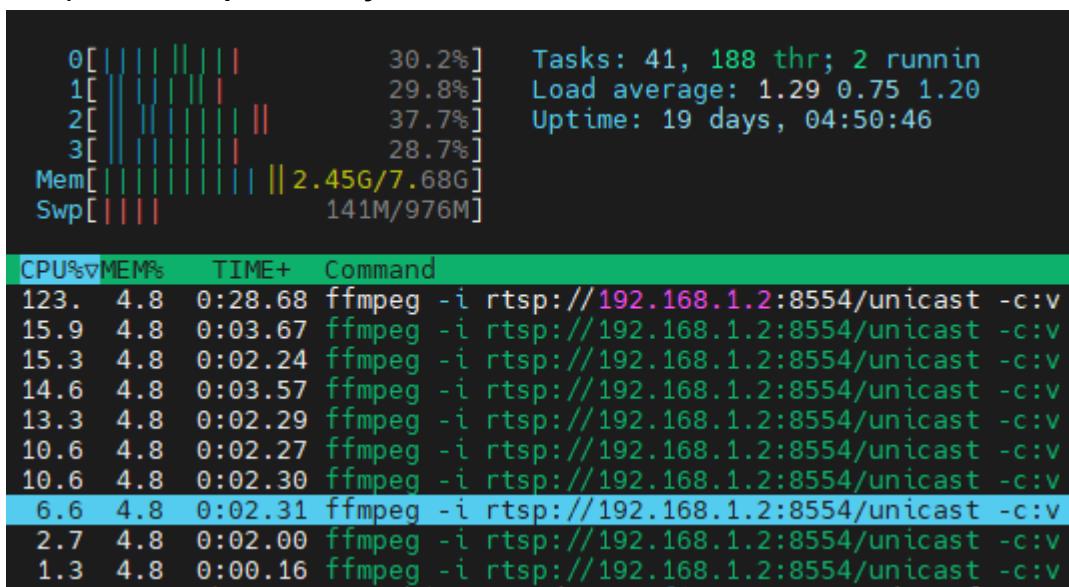
6) Crf 30 et preset veryfast



On observe encore une fois une nette baisse de l'utilisation du CPU en diminuant la qualité avec l'option crf. La taille des fichiers elle aussi est drastiquement réduite entre crf 25 et crf 30. (20 Mo au pire contre 10 Mo maintenant) Cela s'explique par le fait que les valeurs de crf sont établies sur une échelle exponentielle. La documentation dit : "The range is exponential, so increasing the CRF value +6 results in roughly half the bitrate / file size, while -6 leads to roughly twice the bitrate."

En revanche, la qualité avec ces paramètres est trop mauvaise à mon goût, j'ai donc baissé le crf à 28.

7) Crf 28 et preset veryfast



Le CPU est légèrement plus utilisé, mais la qualité de la vidéo est maintenant correcte. Avec ces paramètres, la taille des fichiers de 15 secondes varie entre 3,8 Mo et 6,9 Mo. Pour 48h, ça donnera dans le pire des cas **79 Go pour 48h**.

8) Résultat avec les nouveaux paramètres d'encodage

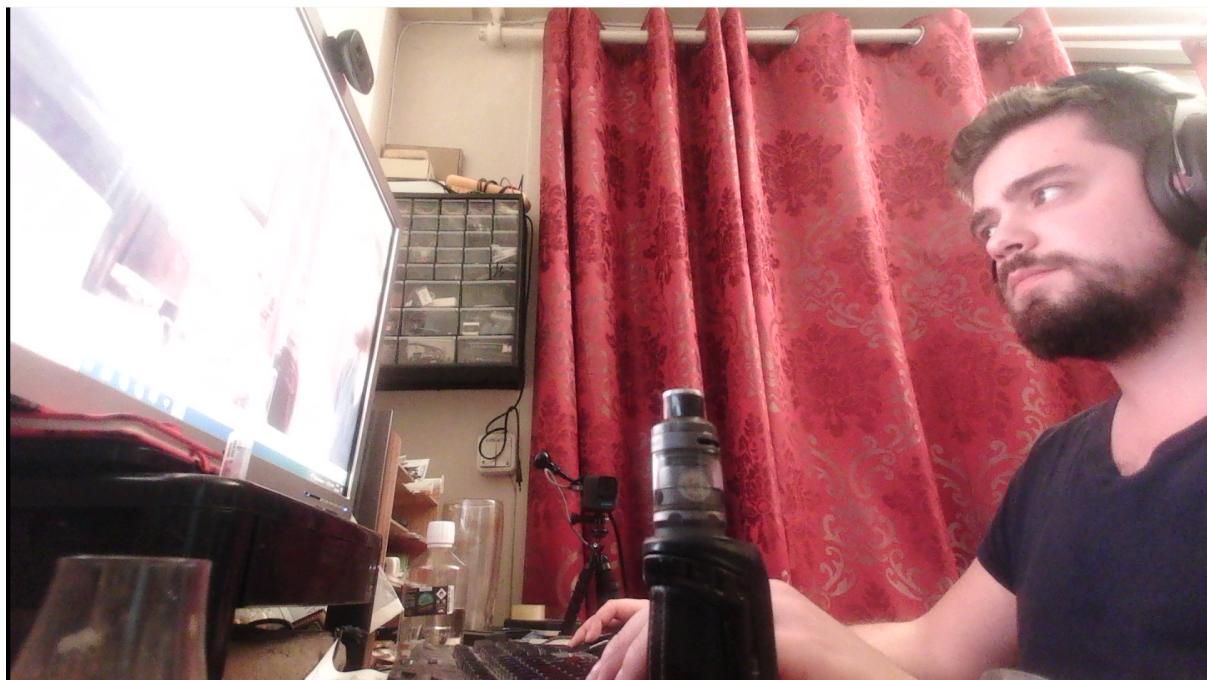


Image brute tirée du flux vidéo dans VLC.

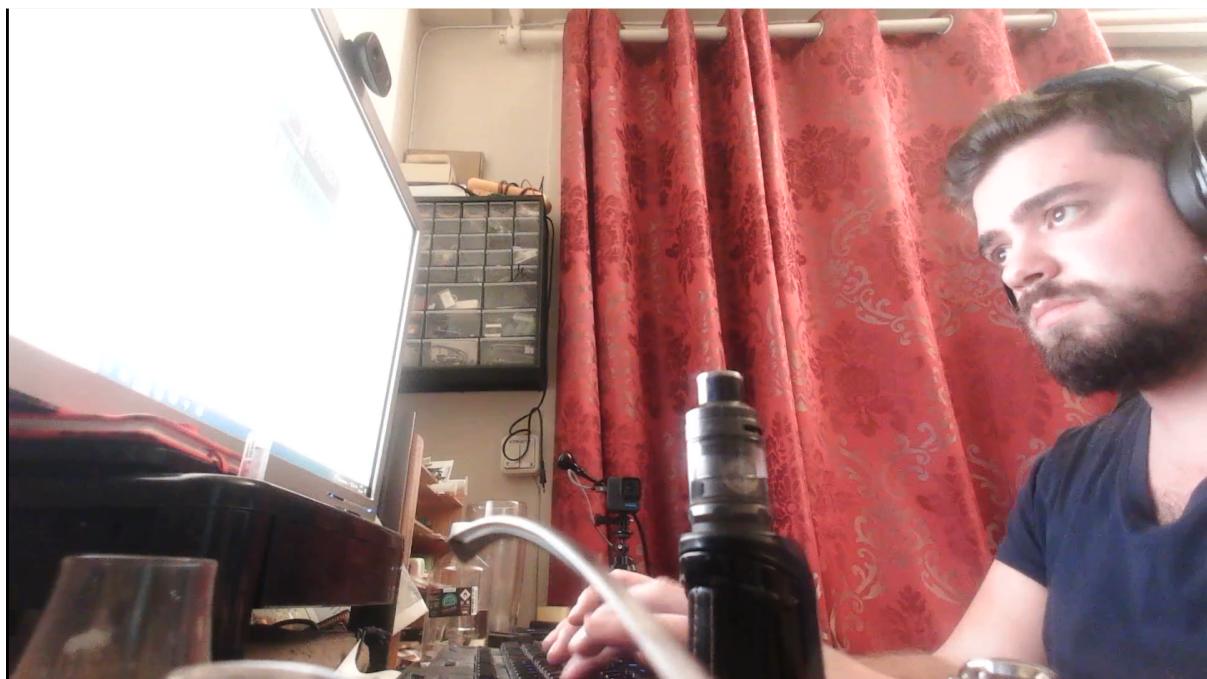


Image tirée d'une vidéo encodée avec les paramètres `-crf 28` et `-preset veryfast`.

Ces essais ont permis d'arriver à un réglage optimal de la caméra en compressant le flux vidéo assez pour arriver à 79 Go au plus pour 48h de contenu.

Conclusion

J'ai atteint les objectifs que je me suis fixés, mais j'ai passé beaucoup plus de temps que prévu sur ce projet, notamment lors de mes expérimentations pour la capture vidéo et à cause des réglages et des recherches nécessaires pour effectuer un encodage efficace du flux vidéo. Comme je l'ai décrit, j'ai d'abord enregistré la vidéo sans l'encoder, ce qui m'aurait demandé 1,7 To de stockage pour 48h, puis j'ai lancé un encodage avec les paramètres par défaut de ffmpeg, en me renseignant et en modifiant les paramètres d'encodage tout en surveillant la vitesse d'encodage indiquée dans les logs de ffmpeg pour qu'elle reste supérieure à "1x" (donc en temps réel). J'ai fini par atteindre un espace occupé pour 48h de vidéo inférieur à 79Go contre 1,7To, soit un taux de compression de 0,046 seulement avec une utilisation du CPU inférieure à 30%.

Au cours de ce projet, j'ai appris à régler finement les paramètres d'encodage h264 d'un flux vidéo pour en obtenir des performances optimales. Ces compétences s'appliqueront aussi au paramétrage du codec h265 et elles me seront utiles pour tout autre projet touchant à la gestion d'un flux vidéo ou audio. J'ai étudié différents systèmes pour transmettre un flux vidéo à travers un réseau, et j'ai pris connaissance plus en détail de RTSP. J'ai aussi découvert l'existence et le fonctionnement de l'outil iftop, que j'utiliserai sûrement encore à l'avenir.

J'ai investi un peu plus de 30 heures dans ce projet pour le réaliser, le documenter, et produire les scripts et services nécessaires à son fonctionnement. Je demande donc au jury de m'attribuer des XP pour 4 jours de travail, donc 8 XP au total s'ils estiment que c'est bien justifié.

Maxence Abela

maxence.abela@epitech.eu

maxence.abela@gmail.com