UNIVERSITÉ PARIS 8 VINCENNES-SAINT-DENIS

Smile OPEN SOURCE SOLUTIONS

Rapport de stage M2

Auteurs :
Nizar ABAK-KALI
11290569

Tuteur :
M. Pierre LAMOT
M. Fabien DUTUIT

31 août 2017

Résumé du stage

Ce stage s'est déroulé dans la société de service Smile et plus particulièrement le département Smile ECS, responsable de l'offre embarqué et IoT [1]. Ainsi, mon stage s'est d'abord déroulé sous la tutelle de Pierre Lamot, expert en traitement d'images et streaming chez Smile ECS, puis sous celle de Fabien Dutuit, expert traitement de signal.
Au cours de ce stage, j'ai eu l'opportunité de travailler sur une grandes variétés de technologie et ainsi d'avoir la chance de pouvoir engranger un maximum de connaissances et d'expériences dans le monde de l'embarqué que je compte utiliser plus tard.

Ce stage ce découpe en deux phases. Une première concentré sur l'étude du protocole MPEG-TS ainsi que de son implémentation dans Gstreamer [2] puis de l'écriture d'un patch décri plus tard dans le rapport, permettant le multipléxage de données non typé dans un flux vidéo. Puis dans un second temps, mon stage c'est plus concentré sur le projet SisselBox [3], où je devais dans un premier temps mettre à jour le système de provisionnement, puis implémenter une application afin de tester mon patch.

1. IoT : Internet of Things ou objets connectés
2. 4.1
3. 2.1

Table des matières

Table des figures

Remerciements

A l'issue de ce stage, je souhaite remercier l'équipe technique de Smile ECS [4] de m'avoir accordé une place dans leur département d'embarquer et de m'avoir permis d'effectuer mon stage de fin d'étude au sein d'une équipe compétente et conviviale. Je remercie également M. Pierre Lamot, expert technique ECS, pour m'avoir soutenu et motivé pendant le stage, et tant apporté.Je souhaite également remercier M. Fabien Dutuit, expert technique ECS, pour m'avoir aidé dans les difficultés que j'ai pu rencontrer. Je souhaite remercier également toute l'équipe de Smile ECS qui m'a supporté durant ces 6 mois, mais qui m'a avant tout apporté énormément aussi bien sûr le plan humain que technique.

4. Embdeed Computer Science

Introduction

Dans le cadre de mon master en informatique embarqué à l'université paris 8, il est nécessaire d'effectuer un stage de fin d'étude d'une durée de 6 mois en entreprise. Après avoir prospecté plusieurs offres de stage et avoir effectué plusieurs entretient, mon choix s'est finalement porté vers la société Smile et plus particulierement le département embarqué. J'ai donc rejoint l'équipe de Smile ECS, filiale spécialisée dans l'infor-matique embarquée et le traitement des images à destination des industriels. Mon stage s'est donc déroulé du 20 février au 18 aout 2017 au sein de Smile ECS, à Asnieres.

Chapitre 1

Présentation de l'entreprise

1.1 Smile

Fondé en 1991, Smile est devenu dès 1995 un acteur du web réputé, maîtrisant l'architecture, les technologies et les outils qui permettent de construire les plus grandes plateformes de l'Internet. Depuis 2001, Smile est intégrateur de solutions open source, c'est-à-dire que le cœur de métier de Smile est la construction de systèmes d'information et de plateformes web intégrant les meilleures solutions open source du marché. Smile mène une forte action de veille afin d'identifier les solutions open source les plus matures et les plus pérennes, qui apporteront un réel bénéfice de compétitivité pour les entreprises. Smile met à disposition un échantillon de cette expertise au travers de livres blancs, librement diffusés, qui sont devenus des références dans leurs domaines. Premier intégrateur spécialisé sur les solutions open source, Smile a été choisi à de nombreuses reprises par les plus grandes entreprises et administrations pour déployer ces solutions dans le cadre de projets stratégiques. Autour du cœur de métier qu'est l'ingénierie, Smile propose une palette de services étendue, qui lui permet de prendre en charge un projet dans sa globalité : consulting en amont et en accompagnement des projets, agence interactive intervenant tant en création et webdesign qu'en conseils éditoriaux, stratégiques et e-marketing, tierce-maintenance applicative (TMA), formation, support et maintien en conditions opérationnelles, et enfin hébergement et exploitation. Smile possède également un rayonnement international grâce à ses dix-sept agences réparties dans 9 pays.

Smile organise également une veille technologiques importantes sur l'open source, avec la rédaction de nombreux livres blancs par les différents consultants sur des sujets variés allant des solutions web d'e-commerce, en passant par les middlewares et finissant par l'embarqué. Il existe actuellement une vingtaine de livres blancs,

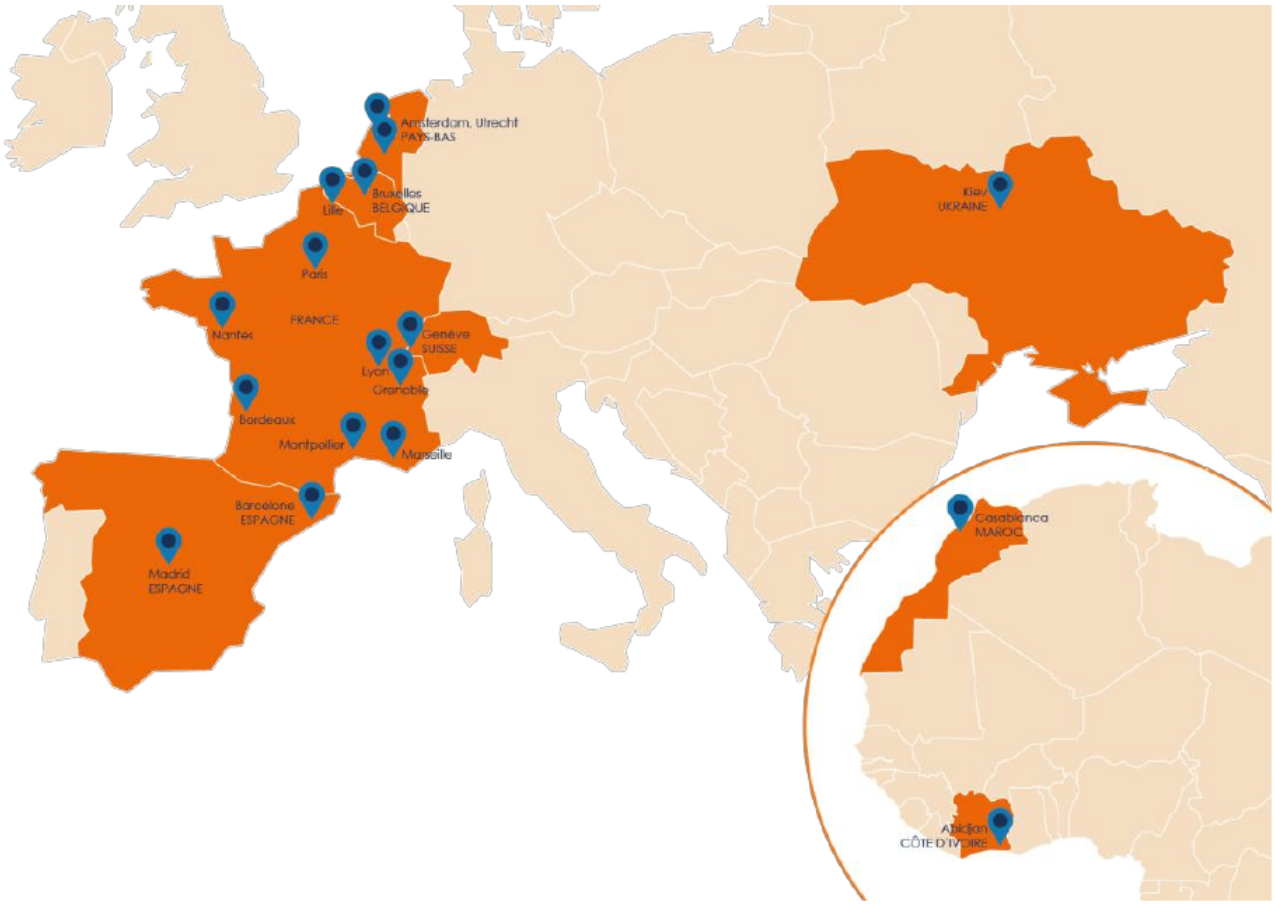

FIGURE 1.1 – Agences Smile

renouvelés tous les 2-3 ans pour inclure les nouveautés des différentes technologies. Dès 2001, Smile commence à construire son expertise des solutions open source : un choix d'avenir que beaucoup de ses concurrents n'osent pas alors entreprendre. A partir de 2004, les grandes entreprises adoptent de plus en plus souvent les solutions open source. Smile occupe une position solide de leader sur ce marché qui décolle, élargissant progressivement son offre vers de nouveaux domaines – CMS, Portails, e-Commerce, Décisionnel, Infrastructure, ERP - et créant des services associés : agence média, tierce maintenance, exploitation, système. En 2007, Smile affiche plus de 50% de croissance et inaugure trois nouvelles agences à Lyon, Nantes et Bordeaux. En 2008, Smile compte plus de 270 collaborateurs et poursuit sa croissance en se développant à l'international, notamment à Kiev en Ukraine. En 2009, Smile compte 320 collaborateurs dans le monde et poursuit son extension internationale. En juin 2009, Smile intègre Cometa Technologies S.L., basée à Barcelone et compte alors 8 agences dans le monde. En 2011, Smile regroupe 500

collaborateurs et ouvre ses agences à Bruxelles, Utrecht et Amsterdam.

En 2013, Smile compte 700 collaborateurs et 17 agences. 2014 est une année importante pour Smile, avec plus de 20% de croissance sur ces cinq dernières années et près de 700 collaborateurs dans le monde. Au fil de ces années, Smile est devenu un acteur incontournable de l'open source, leader en France et en Europe. En 2015, Smile annonce être en négociations exclusives avec la société Open Wide en vue d'un rapprochement. Ceci afin d'élargir son offre et renforcer son leadership.

1.2 Smile ECS

Smile ECS, ex-OpenWide est une société spécialisée dans le domaine de l'open source, racheté par Smile. J'ai décidé d'intégrer cette entreprise pour mon projet de fin d'étude à l'université Paris 8 afin d'approfondir mes connaissances dans le domaine du logiciel libre, et celui de l'embarqué. Les technologies Open source devenant de plus en plus prédominantes dans la paysage informatique moderne, intégrer une société spécialisée dans cette technologie était une façon pour moi de donner un cap sur l'orientation de ma carrière professionnelle. Ce rapport explicitera mon travail durant ce stage de fin d'étude, qui a duré six mois, dont le sujet a été d'améliorer le système de transmission des données, afin de pouvoir transmettre un flux vidéo, de caméras de sécurité, avec des métadonnées calculées depuis ce flux (exemple : positions des visages détectés). Puis dans un second temps, recupérer ce flux afin de le stocker pour un visionnage ultérieur. Après une explication du cadre du stage, j'essaierai de développer les technologies utilisées lors de ce stage.

Chapitre 2

Présentation du sujet

Mon sujet s'inscrit dans un projet developpé en interne, le projet SisellBox. Ansi, je vais présenter dans un premier temps le projet, puis dans un second temps expliquer ou je m'inscrit dans le projet et qu'est ce que j'y apporte.

2.1 Le projet SisellBox

FIGURE 2.1 – image de la SisellBox

La Sisell Box est un enregistreur développé par Open Wide permettant de déployer une solution complète de vidéo surveillance rapidement. Ce système peut être utilisé dans des contextes de mission très différents : surveillance longue durée, protection de sites sensibles, capture d'images hautes qualité, ainsi que surveillance ponctuelle. Fonctionnant sur batterie et créant un réseau wifi maillé de façon autonome, elle ne nécessite que peu d'intervention humaine pour l'utilisation. Il suffit de brancher la batterie, brancher les caméras et elle est autonome. L'utilisation d'algorithmes performants lui permet de détecter et d'analyser tout événement (intrusion, stationnarité, mouvement de foule) sur la base de critère précis et configurables. Le projet Sisell se découpe en partie, deux partie, d'une part le "streamer", prenant en entrée une source vidéo "live"(par exemple une caméras de sécurité) et fourni un flux de composé de vidéo et de métadonnées. La deuxième partie, le "recorder" récupère les flux vidéo et de méta- données généré par le streamer. Les méta-données sont enregistrées en base de donnée ainsi que divers informations (localisation des enregistrements vidéo etc.)

2.1.1 SisellBox Streamer

La partie streamer de Sisell Box gère la capture vidéo, l'analyse de l'image et son émission sur un réseau.

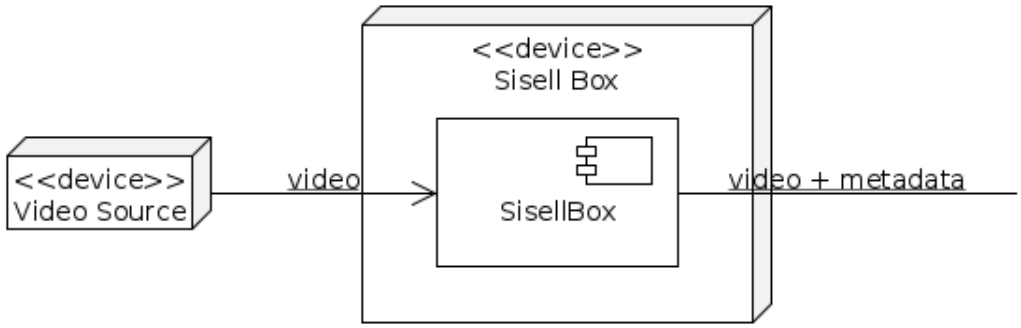

FIGURE 2.2 – Cas d'utilisation du streamer

Afin de faciliter l'inter-opérabilité de cette application, il nous faut se baser sur des protocoles de communications standard. La norme ONVIF qui a pour objectif d'uniformiser les services de vidéosurveillance préconise d'utiliser le protocole RTSP pour les émissions de flux vidéo. Le protocole RTSP (Real Time Streaming Protocol) est un protocole destiné aux systèmes de streaming multimédia. Il permet de contrôler le média à distance (jouer, mettre en pause, enregistrer, fin de

session etc.). Cela permet donc de ne pas se soucier de l'implémentation de ces fonctionnalités côté serveur et permettre le contrôle du flux via un lecteur standard type VLC sans développement supplémentaire. Le RTSP ne gère par contre pas la partie transport des données. Celles-ci sont donc émises via le protocole de transport RTP (Real-time Transport Protocol). Ainsi furent les choix technologiques pris avant mon arrivé sur le projet. Or, il s'est avéré que la norme ONVIF était trop compliqué à implémenté avec Gstreamer à des limitations technologiques. C'est ainsi qu'il a été choisi d'encapsuler le flux de données au travers de MPEG-TS puis de le transmettre via RTP.
L'image capturée par le streamer peut ensuite être analysée par une ou plusieurs brique(s) d'analyse. Une brique d'analyse est un élément interchangeable effectuant un traitement particulier sur l'image afin d'en extraire des informations (détection de visage, reconnaissance de forme, mouvement, etc..). Actuellement, seul quelques algorithmes d'analyse (tel que la détection de visage) ont été implémentés pour la nouvelle architecture.
Les informations extraites par la brique d'analyse sont mises sous la forme de métadonnées. Chaque trame vidéo se voit donc associé d'un trame de métadonnée. Celle-ci peut être vide si aucune information n'a été extraite par l'algorithme d'analyse vidéo mais est néanmoins présente. Le flux vidéo peut être encoder avec les formats H.264 et MJPEG, afin de les transmettre sur le réseau. Le streamer doit aussi gérer des sources non-live (fichiers vidéo) de différents formats (*.mkv, *.ts, *.avi).
Du côté de l'implémentation, les briques algorithmiques de traitement d'image sont développées en C++ et en Vala. Le reste est développé en Python et en utilisant les bindings Python pour GStreamer afin d'utiliser le framework et ses plugins développés en C.

2.1.2 SisellBox Recorder

Le recorder récupère les flux vidéo et de méta- données généré par le streamer. Les méta-données dont enregistrées en base de donnée ainsi que divers informations (localisation des enregistrements vidéo etc.). La vidéo est multiplexé avec les méta-données (en utilisant la piste de sous-titre) puis enregistrée sur le disque. Le recorder peut également récupérer des images (format jpeg) d'une zone de la vidéo récupérée (par exemple un visage détecté en amont) via un élément « Snapshoter ».

Le recorder est capable de :

— Recevoir des flux vidéo et de méta-données et compatible avec au minimum :
- protocoles réseau : RTSP/RTP
- codecs : MJPEG et H.264

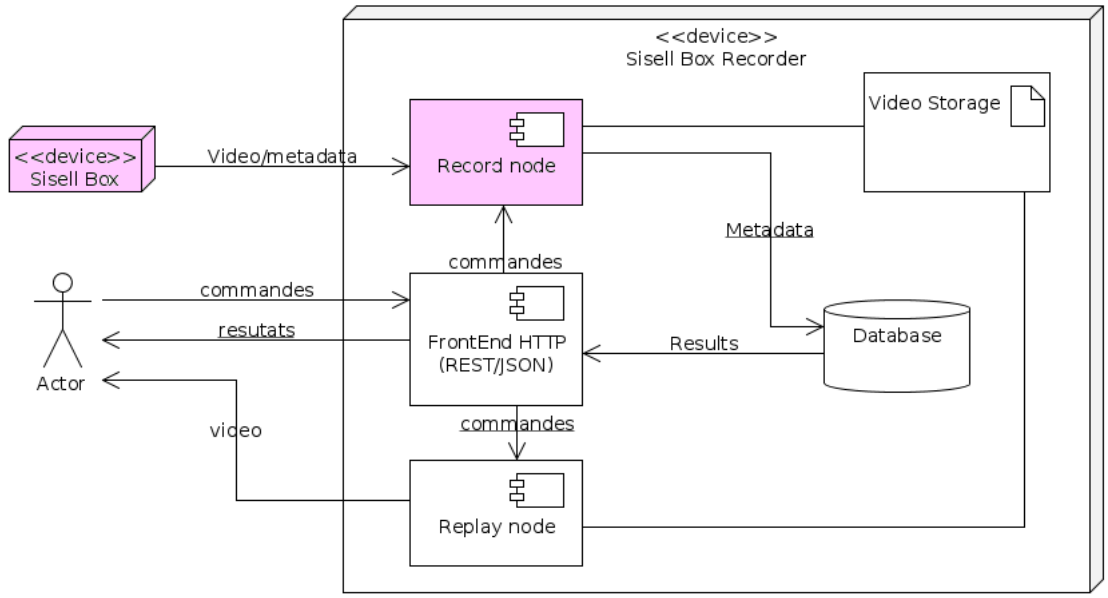

FIGURE 2.3 – Cas d'utilisation du recorder

— Enregistrer les vidéos sur disque dans un format permettant de les relire ainsi que les méta-données associés, dans notre cas, nous utilisons un conteneur MPEG-TS (fichier *.ts) avec les codecs H.264 et MJPEG pour la vidéo et du binaires formatés pour les méta-données

— Stocker les métadonnées en base de donnée

— Effectué des enregistrements sur des événements. Les événements pouvant être de type :

- Présence de méta-données
- événement de type particulier (présence de personne, ...)

Pour la base de donnée, nous utilisons PostgreSQL et Postgis (extension de PostgreSQL pour la manipulation d'informations géographique) sur une machine Debian distante. Pour la phase de développement, cette machine est virtualisée via une VM VirtualBox. Les lectures/écritures dans la base de donnée depuis le recorder se font grâce un mapping objet-relationnel (ORM) permettant de manipuler une base de donnée relationnelle de la même manière qu'une base de donnée orienté objet. Pour cela nous utilisons l'outil libre SQLAlchemy.

Chapitre 3

Les problématiques posées

Ainsi, après avoir explicité le contexte du projet, il est temps que je décrive les différentes problématique qui m'ont été posé durant ce stage. Dans un premier temps, le projet repose sur le framework multimédia Gstreamer [1], n'ayant jamais rencontré cet technologie auparavant, il m'as fallu un temps de d'auto-formation afin d'acquerir les bases pour pouvoir commencer le developpement de plugins simples. Ensuite, il m'as fallut étudier les plugins qui implémente le multiplexage et démultiplage des données dans les fichiers MPEG-TS afin de pourvoir patcher le code.

1. cet technologie est décrite plus en détaille ici 4.1

Chapitre 4

Présentation de l'environnement de travail

Dans le cadre de se stage j'ai eu l'occasion de travailler une plusieurs technologie, beaucoup d'entre-elles m'était inconnue. Ainsi , je vais explicité les différentes technologie sur lesquelles j'ai travailler.

4.1 Gstreamer

GStreamer est un framework multimédia sous licence libre publié pour la première fois le 31 octobre 1999. Il est écrit en C est initialement pensé pour être un solution concurrente à QuickTime et Direct Show sur GNU/Linux. Aujourd'hui, GStreamer est disponible pour de nombreux systèmes d'exploitations (GNU/Linux, Solaris, BSD, Android, OS X, iOS, Windows, OS/400). De plus, des bindings existent pour utiliser GStreamer dans une multitude de langages de programmation en plus du C (C++, Java, Python, Vala etc..)

4.1.1 Principes techniques

GStreamer est basé sur un système de pipeline. Différents éléments sont connectés entre eux par des tuyaux (pipe). Chaque élément possède des "pads" servant à les connecter aux autres. Ces pads peuvent être soit des entrées (sink) soit des sorties (source). Exemple de pipeline GStreamer : Cette pipeline lit un fichier *.ogg, en sépare les pistes et décode sa piste audio et la décode pour ensuite la jouer.

Une autre notion importante de GStreamer est celle de « capabilities » (ca-

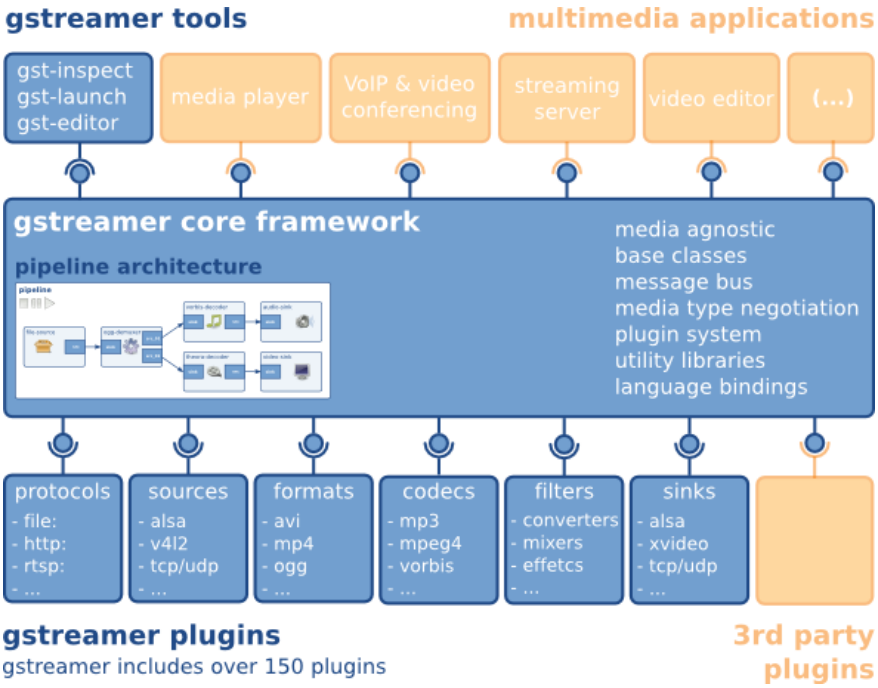

FIGURE 4.1 – schéma de l'architecture de Gstreamer

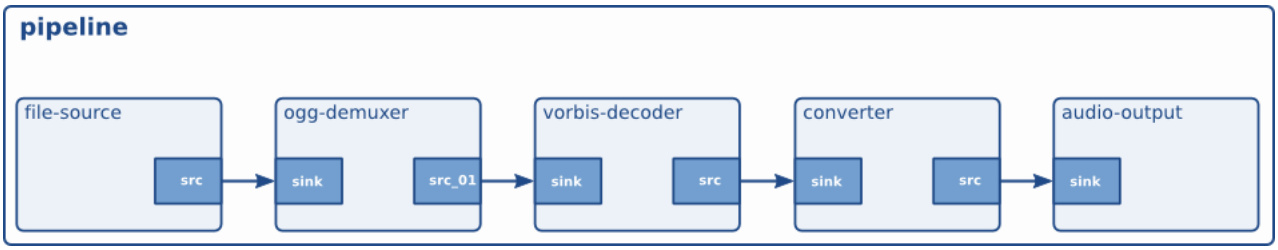

FIGURE 4.2 – Une pipeline gstreamer pour lire un fichier .ogg puis decode en vorbis

pacités) ou «caps». Il s'agit d'une liste de formats et propriétés d'un flux qu'un élément peut accepter sur un pad. Par exemple, un élément acceptant uniquement un format vidéo/x-raw au framerate 24/1 (24 images par secondes) ne pourra pas recevoir d'autre flux (format ou framerate différent). Ainsi les caps peuvent imposer un format, une résolution, un framerate, etc. Ce sont eux qui sont utilisés par GStreamer pour négocier la connexion entre deux éléments et choisir le type de flux qui sera utilisé (si les caps sont compatibles pour au moins un type de flux, sinon la négociation échoue).

```
gst-launch-1.0  videotestsrc ! video/x-raw, framerate=25/1, width=640, height=360 ! ximagesink
```

FIGURE 4.3 – pipeline avec négociation de caps

4.1.2 Présentation des composants Gstreamer

Le framework Gstreamer offre plusieurs types d'éléments, ou "plugins", afin d'effectuer des taches différentes. Gstreamer fourni plus de 200 plugins différent. Ainsi ils permettent :

- la manipulation d'objet multimédia
- gestions des entrées audio/vidéo (caméra/flux réseau)
- gestion des protocoles réseaux
- gestion des filtres
- gestion des sorties

Lors de ce stage, j'ai particulierement travailler avec deux types d'élément, les multipléxeurs (ou "muxers") et les démultipléxeurs (ou " demuxers"). Les muxers sont de élement "N-to-1" qui prennent N flux en entrée puis les "entrelace" en un seul. Particulierement utiles lorsque l'on voudra muxer le flux vidéo avec la méta-donnée. Les démuxers font l'exact inverse, ainsi, ils prennent un flux "entrelacé" puis en fourni N.

3.2.1.2 Débogage

GStreamer possède des outils de débogage très utiles qui m'ont servis tout au long du stage. Le premier est GST_DEBUG. Il s'agit d'une variable d'environnement qui peut être initialisé avec une valeur de 1 à 9 permettant d'afficher plus ou moins d'informations sur le déroulement des actions initiées par GStreamer. L'autre outils est un générateur de fichier *.dot permettant de générer une représentation graphique de la pipeline. Cela permet de vérifier que la pipeline obtenue par le programme est bien celle souhaitée lors de la conception et permet repérer simplement et rapidement certaines erreurs

4.2 OpenCV

Le projet contient des briques algorithmiques fonctionnant grâce à la bibliothèque OpenCV. OpenCV (pour Open Computer Vision) est une bibliothèque graphique libre, initialement développée par Intel, spécialisée dans le traitement

d'images en temps réel. La société de robotique Willow Garage et la société ItSeez se sont succédé au support de cette bibliothèque, Depuis 2016 et le rachat de ItSeez par Intel, le support est de nouveau assuré par Intel.

4.3 Protobuf

Protobuf ou "Protocol Buffers" sont un mécanisme flexible, efficace et automatisé pour sérialiser des données structurées - pensez XML, mais plus petit, plus rapide et plus simple. Vous définissez comment vous souhaitez que vos données soient structurées une fois, vous pouvez utiliser un code source généré spécialement pour écrire et lire facilement vos données structurées vers et à partir d'une variété de flux de données et en utilisant diverses langues. Vous pouvez même mettre à jour votre structure de données sans rompre les programmes déployés qui sont compilés par rapport au format.

Listing 4.1 – modèle pour la structure d'un message

```
1
2  message Person {
3    required string name = 1;
4    required int32 id = 2;
5    optional string email = 3;
6
7    enum PhoneType {
8      MOBILE = 0;
9      HOME = 1;
10     WORK = 2;
11   }
12
13   message PhoneNumber {
14     required string number = 1;
15     optional PhoneType type = 2 [default = HOME];
16   }
17
18   repeated PhoneNumber phone = 4;
19 }
```

C'est à l'aide de cet technologie que nous formattons les métadonnées générer par les algorithmes de traitement d'images afin de les "muxer" avec le flux vidéo.

4.4 Outils de provisionnement

4.4.1 Vagrant

Vagrant est un outil de création et de gestion d'environnements de machines virtuelles. Il est un outil qui permet de démarrer des machines virtuelles avec une configuration donnée. Un très bon moyen de mettre au point et de diffuser des environnements de travail, et ce, de manière reproductible. Vagrant permet donc d'éviter le syndrome « works on my machine ». Plusieurs cas d'usages sont possibles. Le plus courant est celui de faire démarrer un développeur "from scratch" (à partir de zéros). Ensuite, à partir d'un VagrantFile [1] , il suffit de saisir vagrant up et on dispose d'un environnement de développement.

4.4.2 Ansible

Écrit en Python, Ansible est un outil Open Source qui permet l'automatisation de tâches. Grâce à lui, nous gérons vos configurations de VM plus facilement, et de façon automatique grâce à l'exécution de tâches sur des groupes d'hôtes. Il n'utilise pas de client, d'agent, sur les hôtes clients mais nécessite uniquement une partie serveur Ansible, forcément sous Linux. Par ailleurs, Ansible peut éventuellement être utilisé à des fins de contrôle, par exemple pour s'assurer que certains services soient exécutés sur un groupe de machine. Sachant qu'il est possible de développer ses propres tâches grâce aux langages YAML, vous pourrez dompter vos serveurs comme bon vous semble avec Ansible !

1. fichier de configuration de vagrant.

Chapitre 5

Travail effectués

Dans ce chapitre je décrie les tâches accomplies lors de ce stage, ainsi que les difficultés rencontrées.

5.1 Modification du plugin Gstreamer MPEG-TS

Comme expliqué précédemment, mon stage s'inscrit dans une continuité, et donc a pour but d'améliorer le système déjà présent. Avant ma participation au projet, le système prenait des flux vidéo de source différentes (cameras de sécurité, fichier vidéo, etc...), pour ensuite faire passer ce flux vidéo par des briques algorithmiques de traitement d'images (détection des visages, détection de mouvement...). De ce traitement d'image on extrait des métadonnées, dans le cas de la détection de visages nous obtenons des rectangles représentant la surface recouvrants les visages. Ces métadonnées était formatés au format ONVIF [1]. Le flux vidéo et le flux de métadonnées sont "timestamper",afin qu'ils soit synchronisés plus tard, puis il sont ajoutés au conteneur MKV [2] pour la transmission. Ainsi, le flux de métadonnées était passé en sous-titres dans le dans MKV. Or, le format MKV recevait les métadonnées formatés au format ONVIF (sous forme XML), il reformattait de nouveau ce flux, ce qui corrompait les métadonnées. C'est de cet problématique que découle mon stage, on avait besoin de transmettre le flux vidéo/métadon-

1. ONVIF ou "Open Network Video Interface Forum" est une convention international, qui a pour but de faciliter le développement de standard pour l'interfaçage des produits de sécurité avec adresse IP, exemple caméra de sécurité

2. Le format MKV (Matroska Video) est un format vidéo entièrement libre. Plus exactement il s'agit d'un conteneur (d'où le nom Matroska, en référence aux poupées russes) permettant de contenir de la vidéo (DivX, Xvid, RV9, etc.), du son (MP3, Ogg, MP2, AC3, DTS, AAC, PCM), ainsi que des sous-titres (SRT, ASS, SSA, USF, etc.) dans un même fichier.

nées au travers d'un protocole standard sans avoir avoir modifier le format des métadonnées. Ainsi nous nous sommes intéresser au protocole MPEG-TS.

5.1.1 Architecture de MPEG-TS

MPEG-TS ou MPEG Transport Stream permet de contenir du flux digital (audio, vidéo, ou bien privé tel que des sous-titres). Il intègre aussi la synchronisation de flux et la corrections de d'erreur lors d'une perte de paquet. L'autre intérêt de MPEG-TS est qu'il peut contenir plusieurs "programme" dans un flux. Chaque programme peut contenir N flux vidéos ou audio ou privés. Chacun de ces flux à un identifiant ou "PID" [3] et chaque programme à un PID [4]. Les programmes sont enregistré dans une PMT [5] une table des programmes regroupant les différents programmes. Les PMT sont ensuite enregistré dans une PAT [6].

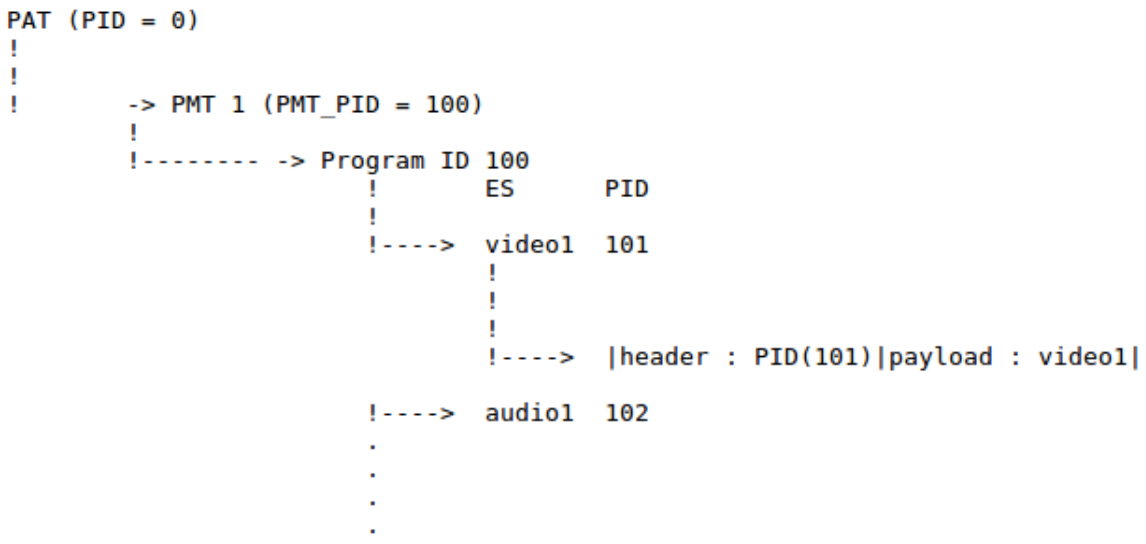

FIGURE 5.1 – schéma de l'encapsulation de média dans MPEG-TS

5.1.2 Explication des plugins mpegtsmux et tsdemux

Suite à l'étude de MPEG-TS au niveau architecturale, je me suis m'y à lecture du code de son implémentation sous Gstreamer. Le l'implémentation se compose

3. Packet ID
4. dans ce cas si PID veut dire Program ID
5. Program Map Table
6. Program Association Table

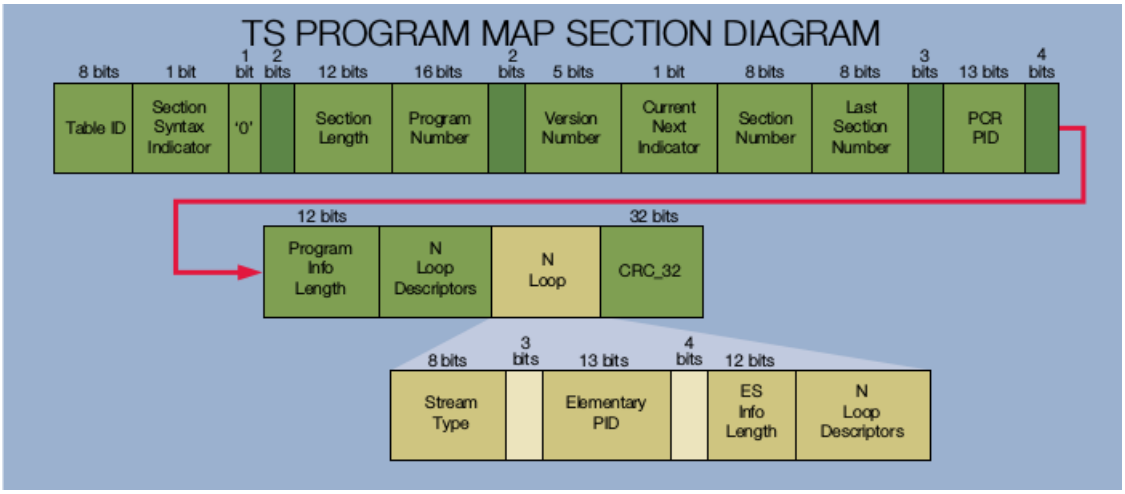

FIGURE 5.2 – schéma d'une trame d'une section Transport Stream

en deux partie, la partie "multiplexage",et la partie "démultiplexage". Comme presque tout les plugins, elle sont codés en C avec la librairie GObject [7].

Le plugin mpegtsmux Le multiplexage se fait au travers du plugin **mpegtsmux**. Le flux de donnée entre depuis le pad d'entré [8] avec une négociation des caps qui donne le type multimédia du flux (exemple : video/x-raw, qui est un flux vidéo pure sans encodage) puis le flux est découpé en paquet "timestamper", et identifier. Ainsi sont traités tout les flux. Ensuite tout les paquets sont "entrelacé" dans un seul flux puis mis à disposition sur le pad de sortie.

Le plugin mpegtsdemux Le demultipléxage fonctionne à l'inverse. Les paquets reçue sont analyser par **tsparse**, plugin qui parse les paquets TS et qui les donne à **mpegtsdemux** . Ensuite le mpegtsdemux refabrique les différents flux et créer un pad à la demande (ou "request pad" dans la littérature anglophone) par flux. Il suffit donc de ce connecter à ces pad de sortie pour avoir le flux que l'on souhaite

7. GObject est une interface de programmation et une bibliothèque logicielle multiplate-forme publiée sous licence libre (LGPL), qui permet de manipuler des objets, ainsi que d'avoir accès à une palette d'objets élémentaires. Deplus, GObject s'interface très bien avec d'autres langages de programmation comme Python ou bien Vala.

8. cf. 4.1.1

5.1.3 Modifications apportés au plugins

Ajout d'un nouveau types élémentaire : Les plugins permettent de muxer et demuxer du flux multimédia connue, dont le type multimédia à un code [9]. Or pour des raisons d'interopérabilité avec d'autres plateformes et logiciels nous avons décidé de créer un nouveau type élémentaire dans le code. Ce type est ajouté au deux plugins [10].

Modifications du muxer pour accepter le nouveau type : Après avoir ajouté un nouveau type élémentaire, il faut que lorsque le muxage soit fait, on puisse identifier le flux avec son type et lui donner son ID . Cela est fait en modifiant la méthode **mpegtsmux_create_stream**. Lors de la négociation des caps on récupère le type du flux, ici "application/x-private". Ensuite on test le type et on attribut le l'"element steam type" pour le flux, dans notre cas **TSMUX_ST_PS_KLV**. Ensuite on attribut le PID du flux, grâce à la méthode de **tsmux_create_stream**, soit en passant un entier de 32 bit, soit la méthode boucle jusqu'a trouvé un PID vacant.

modifications du demuxer : Les modification du démuxer sont plus simple puisqu'il s'agit recupérer un flux de type "application/x-private" puis d'ouvrir un pad de sortie et de fournir le flux demandé . Pour ce faire j'ai ajouté un test dans la méthode "create_pad_for_stream" puis j'ai créer les caps qui convenait.

Listing 5.1 – création de caps pour le flux sortant

```
...
case DRF_ID_XPRIVATE:
            sparse = TRUE;
            is_private = TRUE;
            caps = gst_caps_new_simple ("application/x-private",
                "parsed", G_TYPE_BOOLEAN, TRUE, NULL);
            break;
...
```

Après test j'ai pu transmettre du flux vidéo accompagné de donnée générer aléatoirement et donc pue passer à la deuxième partie de mon stage .

9. cf. `https://en.wikipedia.org/wiki/Program-specific_information#Elementary_stream_types`
10. cf. **??**

5.2 Projet SisellBox

Pour la seconde phase de mon stage, il m'a été attribué de modifier et d'automatisé le système de provisionnement du système tournant sur VM. et ensuite de tester les plugins MPEG-TS dans le contexte de projet.

5.2.1 Étude de l'existant

J'ai dans un premier temps étudier le projet dans son ensemble, Voici une description non exhaustive du projet

ansible : contient tout ce qui concerne le provisionnement de la machine VM ainsi que la gestion des VMs avec Vagrant.

dataset : contient des vidéos de test .

httpproxy : contient les configurations du serveur web nginx.

sboxweb : contient l'interface web du projet, à ce la ajouté le backend codé en Python Flaskr.

simple_stream : contient l'application de test que j'ai développé.

sisell_elements : contient les briques algorithmiques de traitement d'images et interfacé avec Gstreamer sous forme de plugins

5.2.2 mise à jour du système de provisionnement

La nouvelle architecture qui m'a été demandé d'implémenter à pour but de simplifier la compréhension du système de provisionnement mais aussi de simplifier les modifications futures. La découpe est la suivante :

1_devenv une brique réservé au provisionnement des outils qui mettes en place un environnement de développement

2_streamer partie installe les outils de dévelopement necessaire au streamer, principalement les paquets Gstreamer

3_streamer_dev roles qui cherche les source Gstreamer, les configures, les compile, et enfin fabrique des paquets Debian pour faciliter la gestion de ces paquets

4_control script qui compile et installe toutes sources et librairies qui sont reponsable du contrôle au sens le plus abstrait de l'application. On peut y trouver les librairies gRpc et Protobuf, ainsi que leurs binders Python.

5_algo brique consacré à l'installation des librairies OpenCV.

Listing 5.2 – exemple de rôles Ansible

```
    - name: Install Mysql package
  yum: name={{ item }} state=present
  with_items:
   - mysql-server
   - MySQL-python

- name: Configure SELinux to start mysql on any port
  seboolean: name=mysql_connect_any state=true persistent=yes
  when: sestatus.rc != 0

- name: Create Mysql configuration file
  template: src=my.cnf.j2 dest=/etc/my.cnf
  notify:
  - restart mysql

- name: Start Mysql Service
  service: name=mysqld state=started enabled=yes

- name: Create Application Database
  mysql_db: name={{ dbname }} state=present

- name: Create Application DB User
  mysql_user: name={{ dbuser }} password={{ upassword }}
```

Ensuite, pour créer une machine virtuelle à l'aide Vagrant et la provisionner, il suffit de taper la commande :

```
vagrant up --provision
```

Une VM est alors lancer on peut donc y accéder depuis un lien ssh sécurisé que Vagrant fournit :

```
vagrant ssh
```

5.2.3 Application de Test

La finalité de mon stage fut de pouvoir streamer un flux MPEG-TS contenant un flux vidéo, et des métadonnées obtenues depuis des algorithmiques de traitement d'images, au travers du protocole RTP. Ainsi mon but a été de transmettre de puis une VM jusqu'à ma machine. Voici les pipelines réalisé :

Coté streamer :

```
GST_DEBUG=3 GST_DEBUG_DUMP_DOT_DIR=./stream \
gst-launch-1.0 filesrc location=/dataset/videoMP4.mp4 do-timestamp=true !\
  decodebin ! video/x-raw,format=I420 ! tee name=src  \
mpegtsmux name=m ! video/mpegts ! rtpmp2tpay ! udpsink host=10.1.75.173 port=5555
src. ! queue name=post_face_queue max-size-time=0 max-size-bytes=0\
  max-size-buffers=0  ! videoconvert ! video/x-raw,format=BGR ! \
    owfacedetect name=face !\
      capssetter caps="application/x-private,parsed=true"\
replace=true join=false ! m. \
src. ! queue name=video_q  ! x264enc ! m.
```

Le pipeline ci-dessus est celui qui se charge de streamer. On peut Lire les variables GST_DEBUG et GST_DEBUG_DUMP_DOT_DIR qui sont utiles pour avoir des logs plus verbeux et indiqué ou l'on veut que les graphes .dot soit générer. Le plugin **filesrc** cherche le fichier MP4, le lit, et le fournit sur sont pad de sortie. La propriété "do-timestamp" est mise à true afin que le plugin écrive des paquets timestamper. le flux vidéo et ensuite donné à un élément appelé **decodebin**. Il va se charger de décoder le flux et retourné sur son pad se sortie le type demandé en caps, ici "video/x-raw,format=I420". la vidéo décoder va être récupérer par l'élément **tee** que l'on nomme "src". Il se charge de dupliqué sur le flux pour autant de branche que demandé. La prémière branche fourni est celle relié au plugin mpegtsmux patché,tandis que la deuxième branche convertie l'encodage des couleurs du flux vidéo de I420 vers du BGR[11] pour le plugin de détection owfacedetect. Les métadonnées sont ensuite donné,post-négociation de caps, au multiplexeur. Il en va de même pour la vidéo qui est encodé en H.264 par **x264enc**. le muxer produits des paquet TS qu'il fourni au plugin **rtpmp2tpay** qui les empaquette dans des paquets RTP pour ensuite les envoyer par UDP grâce au plugin **udpsink** .

Coté recoder :

```
 GST_DEBUG=4 GST_DEBUG_DUMP_DOT_DIR=./record \
gst-launch-1.0 udpsrc port=5555 ! application/x-rtp,media=video,clock-rate=90000,\
  encoding-name=MP2T ! rtpmp2tdepay ! tsparse ! tsdemux name=d \
d.private_0041 ! queue name=meta_queue ! filesink location=lol\
d.video_0042 !  queue name=video_queue ! video/x-h264 ! h264parse ! \
  avimux ! filesink location=lol.avi  \
```

11. BGR : Blue, Green, Red

ce pipeline fait l'inverse du pipeline streamer . On récupère le flux RTP/TS au travers d'UDP, puis **rtpmp2tdepay** dés-empaquette les paquets TS. Ces derniers sont analyser par **tsparse** qui fourni au plugin **tsdemux** un flux TS à demuxer. Ensuite il suffit de lire les bons pad pour avoir ce que l'on veut. Comme par exemple ci dessus le flux privée est sur le pad "private_0041".

Problème de trou dans le flux de métadonnées Le principale problème rencontré est la désynchronisation des flux due à un manque de métadonnées, cas qui arrive souvent dans le cas où l'on ne détecte pas de visages. La solution a été de modifier le plugin **owfacedetect** qui hérite d'un classe mère OWCollectPadData. Cet dernières est un classe générique qui fabrique est gère la vie d'un plugin ainsi que la gestion de pads. Nous avons donc choisie d'hérité d'un classe plus riche, GstBaseTransform qui offre des fonctionnalité innée comme le comblement de vide dans un flux par des Segment.La classe Segment sert à signaler que le paquet transmit et vide et combien de temps.

Chapitre 6

Conclusion

Chapitre 7

Bibliographie

Sources des plugins mpegtsmux et mpegtsdemux https://github.com/GStreamer/gst-plugins-bad/tree/master/gst/mpegtsmux et https://github.com/GStreamer/gst-plugins-bad/tree/master/gst/mpegtsdemux

How To Install and Use PostgreSQL on Ubuntu 16.04 DigitalOcean https://www.digitalocean.com/community/tutorials/how-to-install-and-use-postgresql- .

GStreamer-1.8.1 rtsp server and client on ubuntu Telecom R & D https://telecom.altanai.com/2016/05/20/gstreamer-1-8-1-rtsp-server-and-client-on-ub .

gstreamer - Plugins : "ugly" and "bad" - Ask Ubuntu https://askubuntu.com/questions/468875/plugins-ugly-and-bad .

grpc http://www.grpc.io/docs/quickstart/cpp.html .

grpc http://www.grpc.io/docs/quickstart/python.html .

Projects · Dashboard · GitLab https://gitlab.openwide.fr/ .

gst-plugins-base/gst-libs at master · GStreamer/gst-plugins-base · GitHub https://github.com/GStreamer/gst-plugins-base/tree/master/gst-libs .

Installing OpenCV 2.4.9 in Ubuntu 14.04 LTS – Sebastian Montabone http://www.samontab.com/web/2014/06/installing-opencv-2-4-9-in-ubuntu-14-04-l .

Installing OpenCV on Debian Linux | Indranil's world https://indranilsinharoy.com/2012/11/01/installing-opencv-on-linux/ .

grpc/INSTALL.md at master · grpc/grpc · GitHub https://github.com/grpc/grpc/blob/master/INSTALL.md .

Gstreamer basic real time streaming tutorial | Einar Sundgren http://www.einarsundgren.se/gstreamer-basic-real-time-streaming-tutorial/ .

GStreamer Debugging - RidgeRun Developer Connection https://developer.ridgerun.com/wiki/index.php/GStreamer_Debugging#Use_standard_GStreamer_debug_output_with_filter .

manual.pdf https://gstreamer.freedesktop.org/data/doc/gstreamer/head/manual/manual.pdf .

open-wide / sisellbox-ng · GitLab https://gitlab.openwide.fr/open-wide/sisellbox-ng .

Accueil - Smile Intranet https://intranet.smile.fr/portal/fr/web/guest/home .

Webmail - BlueMind https://bluemind.smile.fr/webmail/?_task=mail&_mbox=INBOX&_refresh=1 .

Vim as a Python IDE | Unlogic http://unlogic.co.uk/2013/02/08/vim-as-a-python-ide .

Real Time Streaming Protocol - Wikipedia https://en.wikipedia.org/wiki/Real_Time_Streaming_Protocol .

toto.html file:///home/niaba/Documents/sisellbox-ng/doc/architecture/toto.html .

Troubleshooting GStreamer https://gstreamer.freedesktop.org/documentation/frequently-asked-questions/troubleshooting.html .

Gstreamer cheat sheet - MyLabWiki http://wiki.oz9aec.net/index.php/Gstreamer_Cheat_Sheet .

pygst-tutorial.pdf https://brettviren.github.io/pygst-tutorial-org/pygst-tutorial.pdf .

Instantiable classed types : objects : GObject Reference Manual https://developer.gnome.org/gobject/stable/gtype-instantiable-classed.html .

GObject Reference Manual : GObject Reference Manual https://developer.gnome.org/gobject/stable/ .

Mpeg TS helper library : GStreamer Bad Plugins 1.0 Library Reference Manual https://gstreamer.freedesktop.org/data/doc/gstreamer/head/gst-plugins-bad-libs/html/mpegts.html .

gst-plugins-bad/gst-libs/gst/mpegts at master · GStreamer/gst-plugins-bad https://github.com/GStreamer/gst-plugins-bad/tree/master/gst-libs/gst/mpegts .

Diapo : ces bricolages de génie qui donnent envie d'acheter un Raspberry Pi http://www.tomshardware.fr/articles/raspberry-pi-bestof-bricolage-hack,1-63056.html .

MPEG transport stream - Wikipedia https://en.wikipedia.org/wiki/MPEG_transport_stream#PCR .

Novacut/GStreamer1.0 - Ubuntu Wiki https://wiki.ubuntu.com/Novacut/GStreamer1.0 .

rubenrua/GstreamerCodeSnippets : Gstreamer Code Snippets in C, Python and Shel https://github.com/rubenrua/GstreamerCodeSnippets .

Adding Properties https://gstreamer.freedesktop.org/documentation/plugin-developme basics/args.html .

Example GStreamer Pipelines - IGEP Community Wiki http://labs.isee.biz/index.php/Example_GStreamer_Pipelines .

Base MPEG-TS descriptors : GStreamer Bad Plugins 1.0 Library Reference Manual https://gstreamer.freedesktop.org/data/doc/gstreamer/head/gst-plugins-bad-libs html/gst-plugins-bad-libs-Base-MPEG-TS-descriptors.html .

Creating special element types https://gstreamer.freedesktop.org/documentation/plugin-development/element-types/index.html .

GStreamer and in-band metadata - RidgeRun Developer Connection https://developer.ridgerun.com/wiki/index.php/GStreamer_and_in-band_metadata .

Novacut/GStreamer1.0 - Ubuntu Wiki https://wiki.ubuntu.com/Novacut/GStreamer1.0#audio.2Fx-raw.2C_video.2Fx-raw .

cerbero-docs/start.md at master · centricular/cerbero-docs https://github.com/centricular/cerbero-docs/blob/master/start.md .

centricular/cerbero : GStreamer's Cerbero forked to add Meson and MSVC support https://github.com/centricular/cerbero .

GstPlugin : GStreamer 1.0 Core Reference Manual https://gstreamer.freedesktop.org/data/doc/gstreamer/head/gstreamer/html/GstPlugin.html .

YBlog - Learn Vim Progressively http://yannesposito.com/Scratch/en/blog/Learn-Vim-Progressively/ .

The Basics of Writing a Plugin https://gstreamer.freedesktop.org/documentation/plugin-development/basics/index.html .

Things to check when writing an element https://gstreamer.freedesktop.org/documentation/plugin-development/appendix/checklist-element.html .

Shared Libraries http://tldp.org/HOWTO/Program-Library-HOWTO/shared-libraries.html .

Install Qt 5 on Ubuntu - Qt Wiki https://wiki.qt.io/Install_Qt_5_on_Ubuntu .

Vim Awesome http://vimawesome.com/ .

Base MPEG-TS descriptors : GStreamer Bad Plugins 1.0 Library Reference Manual https://gstreamer.freedesktop.org/data/doc/gstreamer/head/gst-plugins-bad-libs
html/gst-plugins-bad-libs-Base-MPEG-TS-descriptors.html#GstMpegtsDescriptor-s
.

Base MPEG-TS sections : GStreamer Bad Plugins 1.0 Library Reference Manual https://gstreamer.freedesktop.org/data/doc/gstreamer/head/gst-plugins-bad-libs
html/gst-plugins-bad-libs-Base-MPEG-TS-sections.html .

MPEG-2 TS Overview http://cmm.khu.ac.kr/korean/files/02.mpeg2ts1_es_pes_ps_ts_psi.pdf .

Packetized elementary stream - Wikipedia https://en.wikipedia.org/wiki/Packetized_elementary_stream .

GStreamer-devel - Gstreamer-1.0 blacklisted entries http://gstreamer-devel.966125.n4.nabble.com/Gstreamer-1-0-blacklisted-entries-td4671087.html .

DevDocs — Offline https://devdocs.io/offline .

Example GStreamer Pipelines - Texas Instruments Wiki http://processors.wiki.ti.com/index.php/Example_GStreamer_Pipelines .

GStreamer Debugging - RidgeRun Developer Connection https://developer.ridgerun.com/wiki/index.php/GStreamer_Debugging .

gst-launch-1.0 https://gstreamer.freedesktop.org/documentation/tools/gst-launch.html#environment-variables .

Basic tutorial 11 : Debugging tools https://gstreamer.freedesktop.org/documentation/tutorials/basic/debugging-tools.html#the-debug-log
.

Sisselbox | Trello https://trello.com/b/2O788DTs/sisselbox .

GStreamer Typefinding and Dynamic Pipelines - Part 2 · pyVision https://pi19404.github.io/pyVision//2014/12/17/21/ .

OpenVPN < Systeme/ConfigPostes < Foswiki https://wiki.smile.fr/view/Systeme/ConfigPostes/OpenVPN#Installation_client_Linux_40Debian_45_Ubuntu_41 .

Good patching practice http://en.tldp.org/HOWTO/Software-Release-Practice-HOWTO/patching.html .

7 Patch Command Examples to Apply Diff Patch Files in Linux http://www.thegeekstuff.com/2014/12/patch-command-examples .

Developer Guide Protocol Buffers https://developers.google.com/protocol-buffers/docs/overview#what-are-protocol-buffers .

Annexe A

Shell d'installation de Gstreamer

https://gist.github.com/nizarAbak-kali/e0e2166b8d363391e79424fa2bebd4a3

Annexe B

Header tsmux.h : énumeration des differents type d'élement stream

Listing B.1 – extrait du header tsmux.h enumerant les differents types connues

```
/* Stream type assignments
 *
 *   0x00     ITU-T | ISO/IEC Reserved
 *   0x01     ISO/IEC 11172 Video
 *   0x02     ITU-T Rec. H.262 | ISO/IEC 13818-2 Video or
 *            ISO/IEC 11172-2 constrained parameter video
 *            stream
 *   0x03     ISO/IEC 11172 Audio
 *   0x04     ISO/IEC 13818-3 Audio
 *   0x05     ITU-T Rec. H.222.0 | ISO/IEC 13818-1
 *            private_sections
 *   0x06     ITU-T Rec. H.222.0 | ISO/IEC 13818-1 PES
 *            packets containing private data
 *   0x07     ISO/IEC 13522 MHEG
 *   0x08     ITU-T Rec. H.222.0 | ISO/IEC 13818-1 Annex A
 *            DSM CC
 *   0x09     ITU-T Rec. H.222.1
 *   0x0A     ISO/IEC 13818-6 type A
 *   0x0B     ISO/IEC 13818-6 type B
 *   0x0C     ISO/IEC 13818-6 type C
 *   0x0D     ISO/IEC 13818-6 type D
 *   0x0E     ISO/IEC 13818-1 auxiliary
 * 0x0F-0x7F ITU-T Rec. H.222.0 | ISO/IEC 13818-1 Reserved
```

```
  * 0x80-0xFF User Private
  */
enum TsMuxStreamType {
    TSMUX_ST_RESERVED                   = 0x00,
    TSMUX_ST_VIDEO_MPEG1                = 0x01,
    TSMUX_ST_VIDEO_MPEG2                = 0x02,
    TSMUX_ST_AUDIO_MPEG1                = 0x03,
    TSMUX_ST_AUDIO_MPEG2                = 0x04,
    TSMUX_ST_PRIVATE_SECTIONS           = 0x05,
    TSMUX_ST_PRIVATE_DATA               = 0x06,
    TSMUX_ST_MHEG                       = 0x07,
    TSMUX_ST_DSMCC                      = 0x08,
    TSMUX_ST_H222_1                     = 0x09,

    /* later extensions */
    TSMUX_ST_AUDIO_AAC                  = 0x0f,
    TSMUX_ST_VIDEO_MPEG4                = 0x10,
    TSMUX_ST_VIDEO_H264                 = 0x1b,
    TSMUX_ST_VIDEO_HEVC                 = 0x24,
    TSMUX_ST_VIDEO_JP2K = 0x21,

    /* private stream types */
    TSMUX_ST_PS_AUDIO_AC3               = 0x81,
    TSMUX_ST_PS_AUDIO_DTS               = 0x8a,
    TSMUX_ST_PS_AUDIO_LPCM              = 0x8b,
    TSMUX_ST_PS_DVB_SUBPICTURE          = 0x8c,
    TSMUX_ST_PS_TELETEXT                = 0x8d,
    TSMUX_ST_PS_KLV                     = 0x8e,     /* only used internally
    TSMUX_ST_PS_OPUS                    = 0x8f,     /* only used internally
    TSMUX_ST_PS_DVD_SUBPICTURE          = 0xff,

    /* Non-standard definitions */
    TSMUX_ST_VIDEO_DIRAC                = 0xD1
};
```