Polytech Marseille – IRM5
Promotion 2014

RAPPORT DE PROJET DE FIN D'ÉTUDES

VIDÉOSURVEILLANCE SÉCURISÉE À DISTANCE

Fabien BASCOUL et Jean-Baptiste MARTIN

Table des matières

Introduction

L'objectif du projet est de mettre à disposition pour des gardiens un service de vidéosurveillance à distance de façon sécurisée.

L'architecture peut se décomposer en deux vues :

1. Matérielle : deux Raspberry Pi (nano-ordinateur avec Raspbian pour OS) et une caméra. Un RPi jouera le rôle de serveur et l'autre le rôle de contrôle de la caméra. On nommera la Raspberry Pi avec une caméra RPi caméra et celle qui fait office de serveur RPi serveur.

2. Logicielle : l'ensemble du code reposant sur le langage Python. Les principaux modules utilisés sont : « Socket » pour la création de socket pour la communication entre les deux RPi, « Picamera » pour une acquisition en Python des données de la caméra et une gestion simplifiées de celles-ci, et « Pycrypto », permettant d'apporter les outils cryptographiques pour la sécurité.

Différents services sont proposés aux gardiens :

1. Accès au streaming live de la caméra

2. Activation de la gestion autonome de la caméra avec archivage lors de détections de mouvement

3. Récupération des vidéos sur lesquelles du mouvement a été détecté

Scénario : un gardien lance un programme qui le connecte au serveur en s'authentifiant, identifiant et mot de passe. Le gardien choisit ensuite l'un de ces trois services. Les échanges pour le service 1 et *3* sont sécurisés et se font en totale transparence pour le gardien.

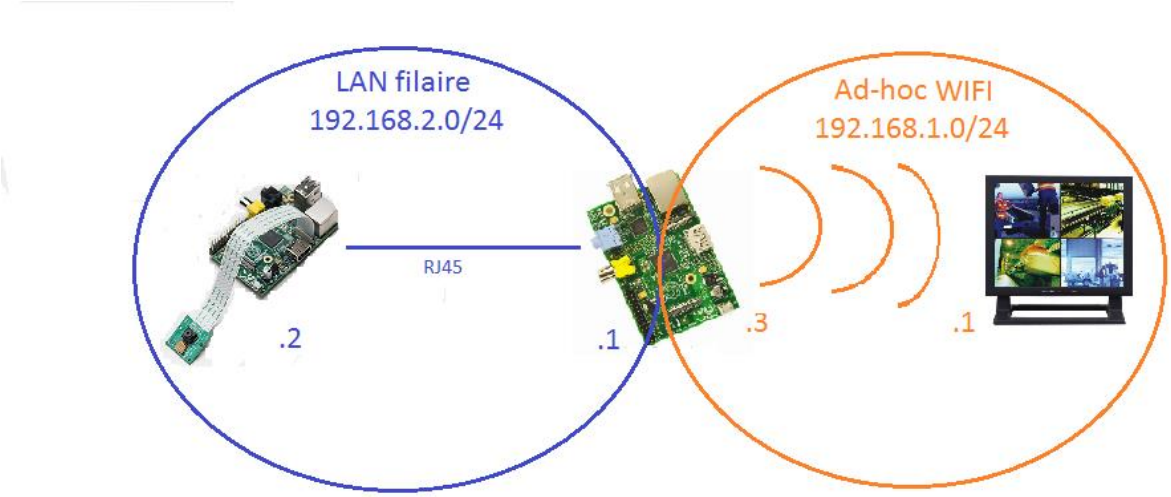
Dans la première partie de ce rapport, nous expliquons l'établissement d'un réseau stable entre les différents équipements et la recherche de bibliothèques et modules pertinents pour nos idées.

Nous verrons ensuite les services proposés et leur fonctionnement.

Enfin, dans une dernière partie, nous évoqueront le système mis en place pour assurer les communications entre les équipements pour accéder aux services ainsi que la couche sécurité que nous avons mis en place sur celui-ci.

I. ARCHITECTURE RÉSEAU

Dans un premier temps, il a fallu mettre en réseau l'ensemble des équipements. Nous avons commencé à faire l'ensemble des tests sur un RPi et un ordinateur connectés en Wi-Fi, puis y avons rajouté l'autre RPi pour arriver à l'architecture suivante.



Pour assurer cette mise en réseau à chaque redémarrage nous avons écrit un script pour chaque équipement : « config_RPI_1.sh », « config_RPI_2.sh » et « config_guardian.sh ».

Une fois ces éléments interconnectés, nous sommes passés à la recherche d'outils, de bibliothèques permettant de répondre aux besoins.

L'acquisition d'image par les commandes simples « raspistill » et « raspivid » ne permettait pas de répondre à nos exigences de manipuler facilement les flux vidéos, nous avons donc essayé différentes bibliothèques.

Motion étant un outil lourd donc peu adapté pour un système embarqué, nous nous sommes orientés vers un module python nommée « Picamera ».

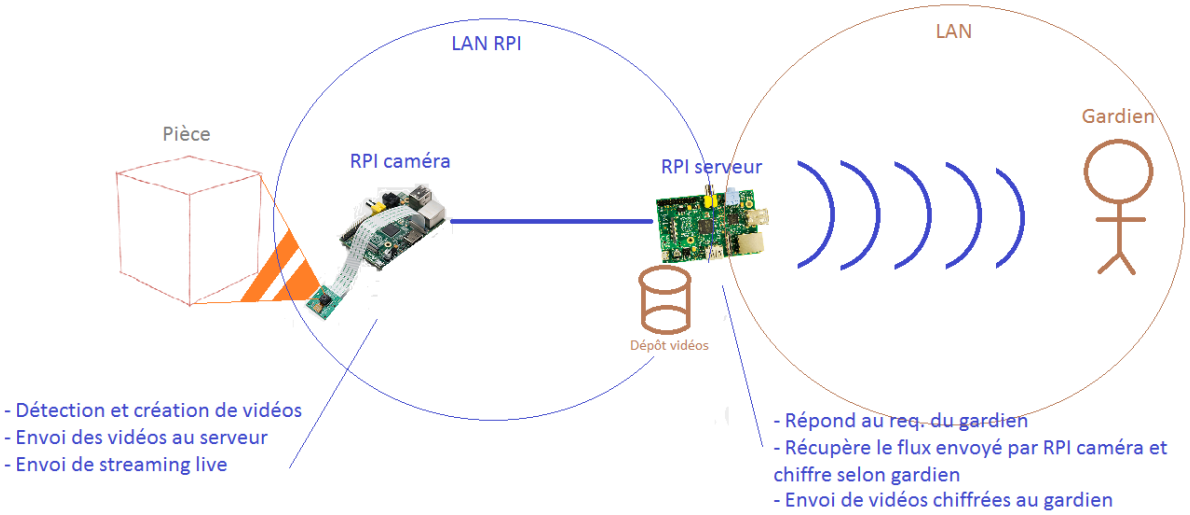
« Picamera » avait plusieurs points positifs :

- ✓ Gestion simplifiée pour l'acquisition des données en provenance de la caméra
- ✓ Redirection simple du flux vidéo vers des sockets ou des fichiers, changement de la destination d'écriture simple
- ✓ Buffers adaptés aux services que nous voulons proposer : buffer permettant de créer un historique de *n* secondes

Dans la partie qui suit nous présentons plus en détails les services liés à l'utilisation du module Picamera.

II. SERVICES

Le schéma suivant représente les services proposés à l'utilisateur.



1. Streaming (RPi caméra → RPi serveur → Gardien)

Le mode streaming permet à un gardien d'avoir le flux vidéo live de la caméra. Le flux est redirigé de la caméra vers le RPi serveur à travers un Socket sécurisé, puis il est fait de même entre le RPi serveur et le gardien. Le transfert des données et la sécurité mise en œuvre sont expliqués dans la partie III. Le flux vidéo est échangé au format h264 et est visionné par le gardien dans le logiciel VLC.

2. Détection de mouvement (RPi caméra seul, puis RPi caméra → RPi serveur)

L'activation du mode détection nécessite d'être initié par un gardien. Cet ordre est transmis au RPi caméra pour que celui-ci travaille en autonomie. Le RPi réalise de la détection de mouvements en exécutant l'algorithme définit dans la fonction de détection.

La fonction de détection en Python consiste à comparer deux images : l'image d'origine (à l'instant t=0) et la dernière image capturée. Un score est déduit de chaque comparaison et s'il est supérieur à un seuil préalablement fixé, il y a détection ce qui implique un enregistrement (écriture de vidéos). Cette comparaison peut détériorer la synchronisation du flux vidéo c'est pourquoi il faut la réaliser à une fréquence peu élevée : environ toutes les 2-3 secondes.

Lors de la détection de mouvement, deux vidéos sont créées :

- `before.h264` : ce qui s'est déroulé jusqu'à 5 secondes avant la détection. Cet historique est réalisé à l'aide d'un buffer circulaire possédant 5 secondes de mémoire sur le flux vidéo de la caméra.

- `after.h264` : ce qu'il se passe du moment où il y a détection jusqu'au moment où aucun mouvement n'est détecté. Le flux de la caméra après une détection va donc être directement redirigé vers une écriture de fichier dans `after.h264`

Lorsque l'événement détection est terminé, c'est à dire que l'on est retourné à l'image d'origine, l'enregistrement dans `after.h264` se termine et l'on retourne dans le mode normal de recherche de détection de mouvements.

L'écriture des fichiers `before.264` et `after.h264` terminé, ceux-ci sont envoyés au RPi serveur. Une fois ces vidéos complètement transmises à la RPi serveur, elles vont être supprimées de la RPi caméra pour éviter de surcharger celle-ci.

Les vidéos sont datées, 'before'+date+'.h264', le format de la date étant : « jour+mois+année+heure+minute+seconde » avec le '+' représentant une concaténation. Datage similaire pour `after.h264`.

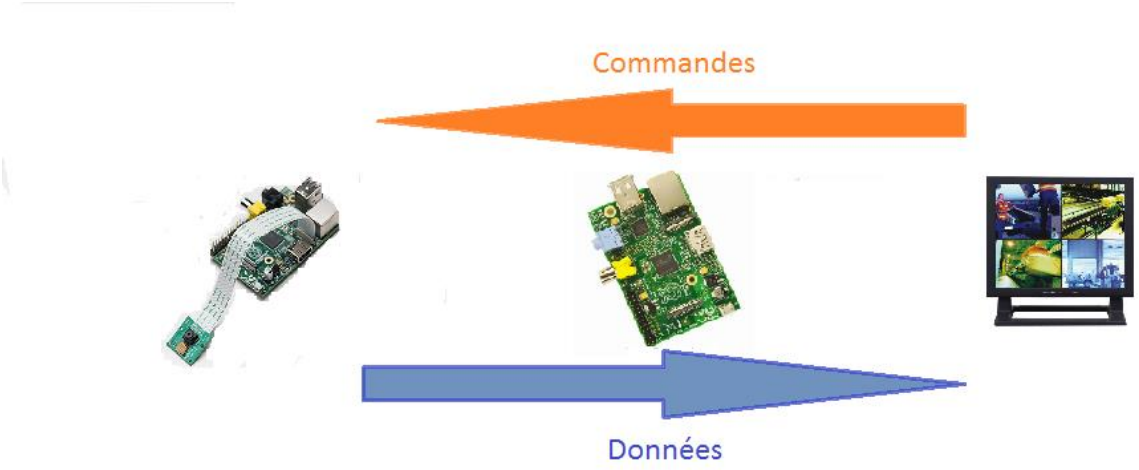
Pour ce service nous avons dû rechercher un compromis entre chiffrement et qualité d'image en utilisant des clés de petites tailles et en se contraignant à 1Mbps à la place de 25Mbps pour la qualité d'image. Pour cela nous avions réalisé des tests pour trouver ce qui rendait une qualité optimale.

Des tests ont également dû être réalisés pour optimiser le seuil du score de détection car avec avec une lumière du jour le seuil de détection doit être revu à la hausse avec la présence de nuages et un rayonnement non constant.

3. Récupération de vidéos enregistrées (RPi serveur -> Gardien)

Le gardien va télécharger les vidéos enregistrées dans le dépôt vidéo de la RPi serveur. Le transfert des données et la sécurité mise en œuvre sont expliqués dans la partie III.

III. COMMUNICATIONS



Ce schéma représente des types d'échanges dans cette architecture.

1. Sockets

Les communications "client / serveur" sont de type "synchrones / bloquantes", c'est à dire dans un ordre prédéfini. Le serveur doit être lancé en premier, puis le RPi caméra et ensuite le gardien.

Lorsque le serveur est lancé, celui-ci attend la demande de connexion du RPi caméra puis la connexion du gardien. L'authentification du gardien a lieu puis le serveur attend la requête demandée par le gardien (message d'un octet).

Ensuite, en fonction du choix du gardien, les communications sont différentes mais le schéma global est:

- Accusé de réception envoyé du serveur au gardien
- Traitement (récupération flux, vidéo, envoie requête au RPi caméra)
- Envoie requête de terminaison au gardien

2. Sécurité

La première étape concernant la sécurité consiste simplement en une authentification “utilisateur / mot de passe” :

- Vérification de la présence d’une clé publique pour le nom d’utilisateur entré (côté client)
- Vérification de la concordance “utilisateur - mot de passe” (côté serveur)

Les clés publiques sont stockées côté client et les mots de passe côté serveur.

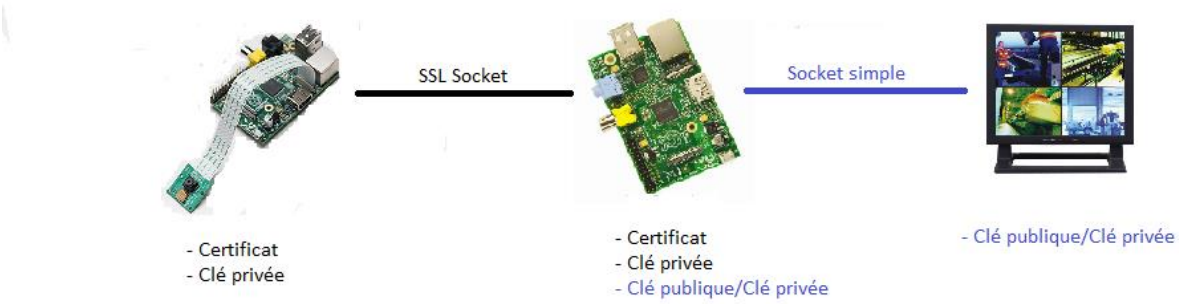


Schéma représentant la sécurité mise en place et la présence de fichiers sur les équipements.

La sécurité par le chiffrement est réalisée à deux endroits, en utilisant l’API Pycrypto, OpenSSL et la bibliothèque SSL de Python.

La connexion entre le RPi caméra et le RPi serveur est sécurisée grâce à l’utilisation de SSL à travers un socket. Pour établir cette connexion, les deux entités doivent contenir des clés RSA et certificats :

- Côté serveur : dans notre cas, nous avons défini le serveur comme autorité de certification. Il faut donc générer une clé et un certificat auto-signé. La clé RSA de 1024 bits est générée grâce à la commande `openssl genrsa 1024 > key_server`
  À partir de cette clé, nous générons un certificat x509 auto-signé d’un an
  `openssl req -new -x509 -days 365 -key key_serveur > cert_serveur`

- Côté caméra : un fichier contenant des autorités de certification.

De cette manière, les informations peuvent transiter entre le RPi caméra et le RPi serveur de façon sécurisée. Le serveur stocke les vidéos en clair.

La connexion entre le RPi serveur et le gardien est sécurisée grâce à une clé secrète. Cette clé secrète est partagée entre les deux entités grâce à un algorithme de chiffrement asymétrique : El Gamal. Chaque gardien dispose de sa propre clé publique (générée par 'genrsa'). Celle-ci sera utilisée pour l'échange de la clé secrète. Pour permettre l'échange de clé secrète, les deux entités doivent se communiquer les paramètres de l'algorithme El Gamal (modulo, générateur, clé publique). Le serveur peut ensuite envoyer la clé secrète.

La clé secrète est générée par le serveur. Les communications sont alors chiffrées en utilisant un algorithme symétrique : AES.

Nous avons choisi un chiffrement AES sur des blocs de 16bits, en mode CFB.

IV. LIMITES ET AMÉLIORATIONS

Les objectifs envisageables par la suite sont les suivants :

- Serveur de sockets multithread : pour une gestion de plusieurs gardiens simultanément
- Améliorer la modularité, c'est-à-dire, ajout de Raspberry et caméra plus facilement permettant une visualisation de différents angles d'une pièce.
- Gestion de l'archivage plus intelligente : suppression ou compression en fonction de la taille et ajout de mémoire physique sur RPi serveur pour archiver plus de vidéos.
- Clé de chiffrement en fonction du temps, c'est à dire que le flux vidéo live ne serait potentiellement lisible que par un gardien. Cependant nous ne l'avions pas appliqué aux vidéos qui auraient dû être chiffrées par le RPi serveur en réalisant une comparaison avec la date de capture.
- Hiérarchisation entre les clés de chiffrement : un administrateur ayant plus de droits que les gardiens lambda, lui permettant de visionner le flux en continu ou n'importe quelle vidéo dédiés à un gardien lambda.

Nous pensons que l'ensemble de ces idées pourraient être intégrées à ce projet et permettrait de donner à celui-ci une dimension plus professionnelle.

Conclusion

Ce fut un projet intéressant qui nous a permis de réfléchir aux contraintes associées à la vidéo surveillance (chiffrement/qualité de vidéo) en utilisant notamment des systèmes embarqués limités en ressources (calcul et mémoire) cependant nous avons considéré un réseau haut débit et avec peu de latences.

Ce projet nous aura également permis d'avoir une première approche des Raspberry Pi et du développement en python et enfin de faire preuve de réflexion quant à la recherche et au choix des technologies à utiliser.

Bibliographie

- Module Picamera : http://picamera.readthedocs.org/en/release-1.2
- Module Pycrypto : https://www.dlitz.net/software/pycrypto/
- Module PyOpenSSL : https://pypi.python.org/pypi/pyOpenSSL
- Code pour la comparaison d'image : http://stackoverflow.com/questions/5524179/how-to-detect-motion-between-two-pil-images-wxpython-webcam-integration-exampl
- Socket et SSLSocket : http://docs.python.org/2/