RAPPORT DE PROJET DE SPÉCIALITÉ GIN950

RÉSEAU DE HAUT PARLEURS WIFI





3IT - Université de Sherbrooke

Jean-Philippe Lingrand et Thomas Duhautois

Encadré par François Grondin

Automne 2014

Sommaire

Introduction

Avant de commencer nous souhaitons remercier François Grondin pour son aide précieuse et sa disponibilité non exhaustive. De même il faut remercier François Michaud pour nous avoir donné l’opportunité de réaliser ce projet.

Dans le cadre de notre échange universitaire à l’Université de Sherbrooke en Automne 2014, nous avons réalisé un projet de conception d’un réseau de hauts parleurs WiFi.

Ce rapport présente nos travaux. Il peut être lu comme une documentation au système produit donc un guide pour comprendre l’architecture du système ainsi que son code source. Le code complet est libre d’accès et d’amélioration.

Il est disponible sur

* <https://github.com/krapskill/hpWifiServer> pour la partie server
* <https://github.com/krapskill/hpWifiClient> pour la partie client.

De tels systèmes sont commercialisés par des entreprises comme SONOS ou SAMSUNG pour ne citer qu’elles.

Ces travaux ont étés réalisés dans les locaux du 3IT à Sherbrooke.

En cas de besoin, nous restons entièrement disponibles pour toutes questions concernant notre système et son code source.

**Jean-Philippe LINGRAND :**

jean-philippe.lingrand@usherbrooke.ca

**Thomas DUHAUTOIS :**

[thomas.duhautois@usherbrooke.ca](mailto:thomas.duhautois@usherbrooke.ca)

tel français : +336 67 46 50 93

Problématique

Le projet consiste à mettre un serveur sonore en place avec de multiples clients.

Les clients doivent être mobiles et l’utilisation de Raspberry Pi a été suggérée. Coté serveur aucune contrainte n’est appliquée. Enfin il est demandé de choisir des technologies logiciels multi plateformes.

Au niveau des flux audio, nous travaillerons avec des fichiers de type .raw.

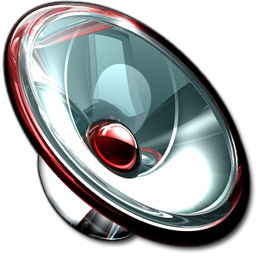
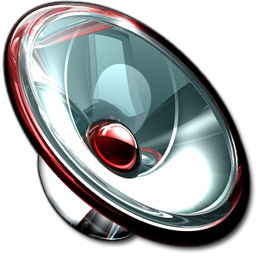
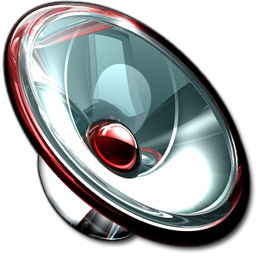
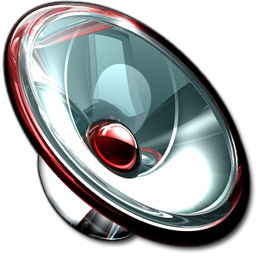
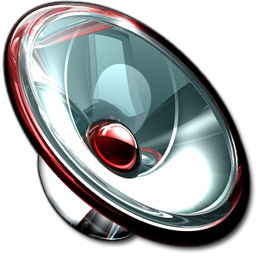
Le RAW est un format de donnée binaire. Autrement dit, ouvrez le fichier RAW avec un éditeur de texte et vous verrez du binaire (10011001010…) probablement au format hexa-décimal.

Un fichier RAW étant un simple fichier binaire, il peut contenir différents types d’informations (musique, photo…).

Ici, on sait que le fichier RAW est un fichier son, et qu’il est encodé de la manière suivante :

* 16 bits signed
* Interleaved mode
* 8 channels

Toute la problématique du projet tient dans la synchronisation des flux entre les haut-parleurs. En effet lorsque l’on diffuse les chaines d’un même son sur des haut-parleurs il faut que ces derniers émettent le son de façon à ce qu’aucun décalage ne soit audible.



Concept

Le concept de synchronisation peut être abordé de différentes façons. Dans l’idée de développer efficacement un système fiable nous avons décidé de baser notre système de synchronisation sur un protocole utilisé et approuvé par la quasi-totalité des systèmes informatiques actuels : Network Time Protocol (NTP).

Le protocole NTP consiste à synchroniser l’heure d’un client avec celle de son serveur. Ainsi la toile est composée d’un arbre de clients serveurs NTP. Plus précisément, un réseau local d’entreprise aura ses machines synchronisées un serveur local NTP, lui-même client d’un serveur NTP de plus haut niveau et ainsi de suite jusqu’à l’horloge atomique. L’algorithme protocolaire implémenté permet à NTP, dans une configuration locale, d’obtenir une flotte de systèmes synchronisés avec une précision de 5ms. En effet le protocole NTP est basé sur une méthode de PULL du client vers son serveur. Dans un schéma macroscopique, des requêtes PULL trop fréquentes intoxiquerait le réseau cependant dans un contexte local, celles-ci peuvent être faites à des fréquences de l’ordre de la seconde.

Une fois admis qu’il est techniquement possible d’avoir la même heure sur un réseau de machines nous avons décidé de finalisé le concept de synchronisation de flux audio en se basant sur le « time stamping » ou horodatage. Plus précisément, le serveur a la capacité de marquer le flux audio d’une référence temporelle. Ainsi le client peut savoir l’heure exacte où il doit diffuser ce flux.

Flux audio brut

Flux audio

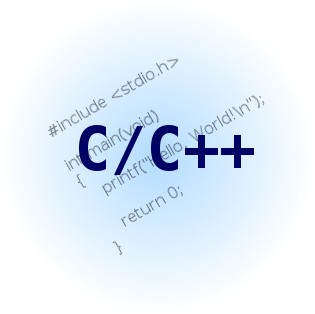
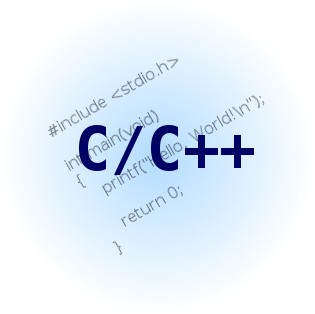
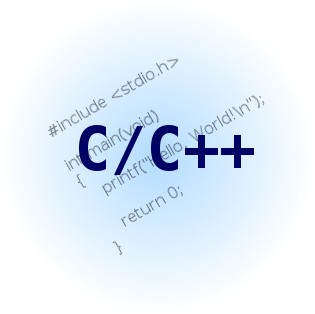
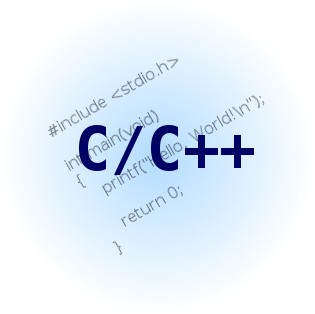
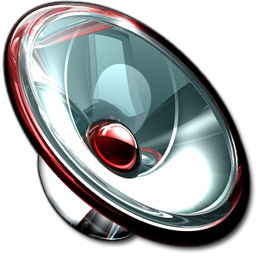
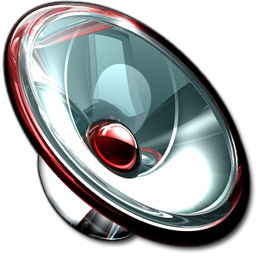
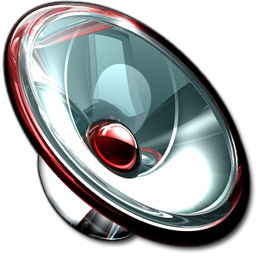
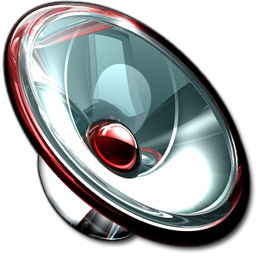
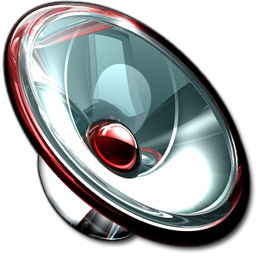
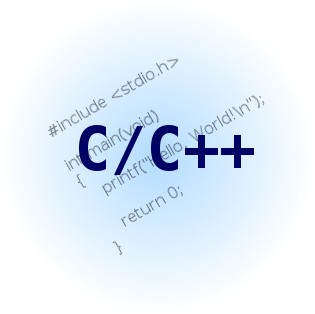
Horodatage

Ensuite il faut prendre en considération les décalages de vitesse de lecture dans chaque haut-parleur. Ainsi si les flux audio commencent en même temps sur toutes les machines il peut se créer un décalage au fur et à mesure de la lecture. En effet chaque système aura une erreur de taux d’échantillonnage qu’il faut rattraper ou compenser. Dans cet optique nous souhaitons que le client sache au cours de la lecture sa position. On doit alors horodater l’ensemble du flux.

Flux audio brut

D’un point de vu des choix de langage de développement le client sera développé en C/C++ et le serveur en node.js. Ces choix ont été fait pour des soucis de portabilité et concernant le node.js, pour sa facilité de mise en place.

Ensuite concernant le moyen de communication nous avons opté pour une connexion web-socket multi-cast entre le serveur et ses clients.



Réalisation

# Serveur

Comme annoncé plus haut le serveur est réalisé avec la technologie node.js. Ce serveur est propulsé par un moteur JavaScript, celui du navigateur Google Chrome.

Node.Js utilise la notion de modules qui peuvent être installés avec l’outil dédié npm ou alors développés par les soins du développeur lui-même.

**Construction d’un module.**

Un module node.js est construit avec l’utilisation de la variable « exports ». Plus précisément, dans le code du module, on peut exporter une fonction ou un attribut de celui-ci en utilisant le mot clé exports.<la fonction ou l’attribut à exporter>. Ensuite on utilise le module avec la fonction « require ». En effet on accède a la fonction ou a ‘attribut précédemment exporté comme ceci : require(<le chemin et le nom du fichier js>).<l’attribut ou la fonction>.

**Architecture**

L’application à lancer est contenue dans le fichier « server.js ». Celui-ci va être responsable d’effectuer les actions demandées par l’utilisateur et de l’informer de la situation et dans un second temps de dialoguer avec les clients. Ces deux communications sont réalisées par web socket. Afin de gérer les connexions web-socket, l’application nécessite l’installation préalable du module « ws.js ». Le serveur envoie aux haut-parleurs des données binaires et utilise le format JSON pour ses échanges avec le navigateur.

La partie de l’application dédiée au traitement et a la diffusion du son utilise un module objet « Player.js » , dédié à maintenir en mémoire plusieurs sons. Et le module « TimeStamper.js » dont les uniques fonctions sont d’extraire les chaines du fichier et de les horodater.

Afin de faciliter les opérations de débogage on utilise le module de log « winston.js ».

## TimeStamper.js

La première fonction de ce module est la bufferisation et l’extraction des fichiers audio.

**Extraction**

Afin de comprendre le processus d’extraction des chaines d’un fichier .raw audio il fau comprendre sa formation. Un fichier .raw audio est composé d’une succession d’échantillons de chaque chaine. Il faut prendre en considération le taux d’échantillonnage afin de réaliser l’opération complète. Le résultat de cette opération est donc un tableau de buffers.

Echantillons 1 à N chaine 1

Echantillons 1 à N chaine 1

Echantillons 1 à N chaine 1

...

Echantillon 1 chaine 1

Echantillon 1 chaine 2

Echantillon 1 chaine M

Echantillon N chaine 1

Echantillon N chaine M

Echantillon N chaine 2

...

...

...

**Horodatage**

L’horodatage est effectué sur chaque chaine indépendamment. Il nécessite une date de début de lecture, une vitesse de lecture et l’information sur le type d’encodage.

## Player.js

# Client

Chaque client représente un haut-parleur. Il se compose donc d’un haut-parleur avec ampli intégré et d’une Raspberry-pi connectée en wifi sur le réseau local du serveur.

Voici une description des items qui sont développés dans le code source client.

Le code source est commenté afin de donner des précisions à ces descriptions. Je vous invite donc à vous plonger dans notre code afin d’avoir de plus amples informations.

1. **Gestion multi-thread**
   * **Réception et traitement des données.**

Le premier thread gère la réception des données. C’est lui qui créé le Websocket et appelle la fonction « HandleBinany\_message » lorsqu’un message de donnée binaire arrive sur le webSocket.

La fonction « HandleBinany\_message » traite la donnée reçue. C’est cette fonction qui à la réception, découpe le message afin de récupérer tous les timestamps (qui sont insérés par le serveur dans la donnée tous les 2048 octets).

* + **Vérification de l’heure.**

Ce second thread a pour objectif de vérifier l’heure du système pour savoir quand il faut jouer la première trame de 2048 octets.

Il est créé dans à la réception d’un message et s’arrête de lui-même une fois la première trame jouée.

* + **Traitement audio.**

Une fois la donnée enregistrée de façon à être lisible par ALSA, le dernier thread réalise un traitement audio sur cette donnée afin qu’elle soit jouée sur le haut-parleur au moment précis indiqué par le timestamp de cette dernière.

Le traitement effectué est une interpolation linéaire. Ce traitement audio permet de recalculer un échantillon de musique en se basant sur l’heure actuelle, l’heure à laquelle la trame doit être jouée, la trame précédente, la trame suivante et un coefficient K calculé pour chaque nouvelle trame.

C’est cette opération qui permet une partie de la synchronisation des flux entre les haut-parleurs.

1. **Websocket**
   * **Permet l’envoi et la réception de données sans requêtes particulières.**

Le client communique donc avec le serveur via un websocket mis en place au lancement du programme en demandant de rentrer l’adresse IP du serveur.

La librairie utilisée est « easywsclient » disponible en libre accès ici :

<https://github.com/dhbaird/easywsclient>

1. **Synchronisation NTP**

Configurer le NTP sur linux se fait via un fichier de configuration « ntp.conf » présent dans le répertoire suivant :

/etc/ntp.conf

Dans ce fichier vous pourrez configurer :

* Le server NTP à utiliser (dans notre cas et sur un réseau local, l’adresse IP du server).
* Les autres appareils connectés au réseau que vous autorisez à se connecter comme client NTP. (inutile ici car tous les clients doivent se connecter au même serveur. Sinon il y aurait un client de client et risque d’avoir des délais supplémentaires dans la communication NTP Client/server.
* La fréquence de mise à jour de l’heure de votre système.

Compilation du code :

Concernant la compilation du code, je vous invite à aller directement sur le github du projet partie client et à lire le README.md qui vous donnera les différentes lignes de commandes à taper dans votre console linux et une description de leur contenu.