Institut National des Sciences
Appliquées de Lyon
&
KompleXKapharnaüM

Stage de 4ème année du département Génie Électrique
Projet Do Not Clean

Réalisation d'une carte multimédia programmable
et contrôlable via wifi

Protocole de synchronisation décentralisé

Auteur :
Olivier Radisson

Tuteur de stage :
Gilles Gallet

Chef de projet :
Pierre Hoezelle

- 8 janvier 2015 -

Dernière édition le 9 janvier 2015

Résumé

Ce document présente le protocole de synchronisation qui est la base du réseau décentraliser des cartes.
Ce protocole s'occupe de synchroniser entre les cartes :
— La liste des cartes présente, leur nom et l'adresse IP associé
— Le temps sous forme d'un *timetag*
— La version globale du scénario

Le protocole est exécuté par chaque carte via une Machine à nombre Fini d'États[1] qui sera l'objet principal de ce document avec la liste des messages OSC associés.

Ce document présente aussi rapidement le protocole ACK créé pour l'occasion qui est une surcharge de l'OSC permettant de s'assurer qu'un message est bien parvenu à destination.

Table des matières

1 Introduction

Ce document à une visée majoritairement technique. Les points abordés peuvent intéresser les utilisateurs les plus curieux, mais sa vocation première est de présenter le fonctionnement du protocole de synchronisation pour documenter le développement

Les cartes électronique crées dans le cadre du stage pour le projet Do Not Clean sont des cartes reliées à un réseau informatique via un signal WIFI[2]. Il a été explicitement demandé que ces cartes puissent communiquer entre elles et former un réseau, mais sans nécessité la présence d'un élément central ou d'une régie.

De plus ces cartes doivent partager entre elles un certains nombre d'information dont les principales sont :

— Le temps
— Le scénario à jouer
— La liste des cartes présentes

C'est pour répondre à ce besoin que le protocole décrit ci-dessous à été inventé.

Pour assurer un fonctionnement le plus robuste possible malgré l'utilisation d'une connexion WIFI et du protocole UDP[3] pour le transport des données, il a été également inventé un petit protocole d'*acknwolegment* dénommé ACK qui s'assure qu'un message a bien été reçu.

2 Présentation du protocole ACK

Le protocole ACK est une sur-couche du protocole OSC [1][4] qui s'assure de la bonne réception d'un message.

L'émetteur du message OSC ajoute en début de celui-ci deux nouvelles informations permettant d'identifier ne manière unique le message. Ensuite il envoi celui-ci au destinataire sur un port différent que celui prévu pour les communication OSC classique.
Tant que l'émetteur n'a pas reçu de confirmation de réception il envoie le même message autant de fois qu'il lui est défini de faire.

Lorsque le destinataire reçoit un message sur ce port il vérifie si celui-

1. *Open Sound Control*

ci n'a pas déjà été reçus et ce grâce aux deux identifiants uniques. Si le message est reçu pour la première fois celui-ci est transmit à la couche supérieur comme s'il s'agissait d'un message OSC classique. Si le message avait déjà été reçus il est simplement ignoré.
Enfin, dans tous les cas le destinataire renvoie quoi qu'il en soit un message, avec les mêmes identifiants, à l'émetteur pour lui signifier qu'il a bien reçu le message.

Une fois que l'émetteur a reçu confirmation de la réception de son message il arrête d'envoyer en série le message et peut être sur que celui-ci est bien arrivé.

2.1 Empaquetage d'une trame OSC via le protocole ACK

Trame OSC via le protocole ACK

IP	Port	Protocole	Adresse
X.X.X.X	1782	UDP	···

Arguments

Champs	Type	Valeur
ID_time	int	(0xFFFF0000 & *time.tv_sec*) | ((0xFFFF0000 & *time.tv_nsec*) ≫ 16)
ID_rand	int	`rand()`
···	···	···

Table 1 – Trame OSC générique pour un message transporté par le protocole ACK

Un message OSC quelconque se verra empaqueté de cette manière avec les deux identifiants unique `ID_time` et `ID_rand`.
L'adresse ainsi que le reste des arguments sont remplies par le message d'origine.

2.2 Message de réponse

À chaque fois qu'une carte reçoit un message sur le port définit pour le protocole ACK il répond via le message présenté par la table 2.2 pour

signifier que le message lui est bien parvenu [2].

Trame OSC

IP	Port	Protocole	Adresse
X.X.X.X	1782	UDP	/ack

Arguments

Champs	Type	Valeur
ID_time	int	*ID_time*
ID_rand	int	*ID_rand*

Table 2 – Trame OSC pour signifier la réception d'un message ACK

3 Protocole de synchronisation décentralisé

Le protocole qui va permettre de synchroniser chaque carte avec le reste du réseau de manière décentralisé se base sur l'émission d'un court message de présentation sur l'adresse de broadcast [3] et à intervalle régulier.

Ce message de présentation inclus le nom unique de la carte qui l'émet ainsi que son *timetag* [4] et la version du scénario dont il dispose. Chaque carte qui reçoit un message de ce type ajoute, si ce n'est pas déjà le cas, la carte qui l'a émit dans sa liste des cartes présente.
Ensuite elle vérifie si son *timetag* n'est pas plus vieux que le sien, si c'est le cas il y aura demande de synchronisation. De même pour la version du scénario.

Ce fonctionnement totalement décentralisé permet une grande souplesse dans la structure de réseau et surtout la création de sous réseau si la connexion est perdue puis le regroupement en un seul grand réseau lorsque la connexion est rétablie et ce, de manière toute à fait transparente.

2. Chaque message excepté bien-sûr un message sur l'adresse */ack*
3. L'adresse de broadcast est une adresse de destination tel que tous les éléments d'un même réseau prennent le message comme leur étant adressé. Ceci permet de communiquer avec des éléments du réseau sans connaître leur adresse IP.
4. Le *timetag* représente une référence de temps absolue que possède la carte. Cette notion sera développé plus en détails plus tard.

3.1 Présentation sur le réseau

À intervalle régulier et définie en avance. Chaque carte du réseau va envoyer un message de présentation sur l'adresse de broadcast.

3.1.1 Trame OSC de présentation

La trame pour se présenter est la suivant :

Trame OSC

IP	Port	Protocole	Adresse
255.255.255.255	1781	UDP	/iamhere

Arguments

Champs	Type	Valeur
uName	string	*Nom unique de la carte*
timetag	int	*Temps de référence*
scenario	int	*Version du scénario*

Table 3 – Trame OSC pour signifier sa présence sur le réseau

Chaque élément de réseau recevant ce message va vérifier s'il ne connait pas déjà la carte en question. Si celle-ci est nouvelle il l'ajoute dans sa table des cartes présentes sur le réseau.
Ensuite si le timetag de l'émetteur est plus récent il va demander une synchronisation du temps, si il est inférieur il va lui envoyer un message de présentation pour forcer celui-ci à lui demander en retour une synchronisation du temps [5]. Il en va de même pour la version du scénario.

3.1.2 FSM de la partie présentation du protocole

Voici la FSM partielle correspondant à la partie présentation sur le réseau :

5. Il n'y a comme cela toujours que l'élément en retard qui provoque une synchronisation du temps, cela rends plus simple le protocole.

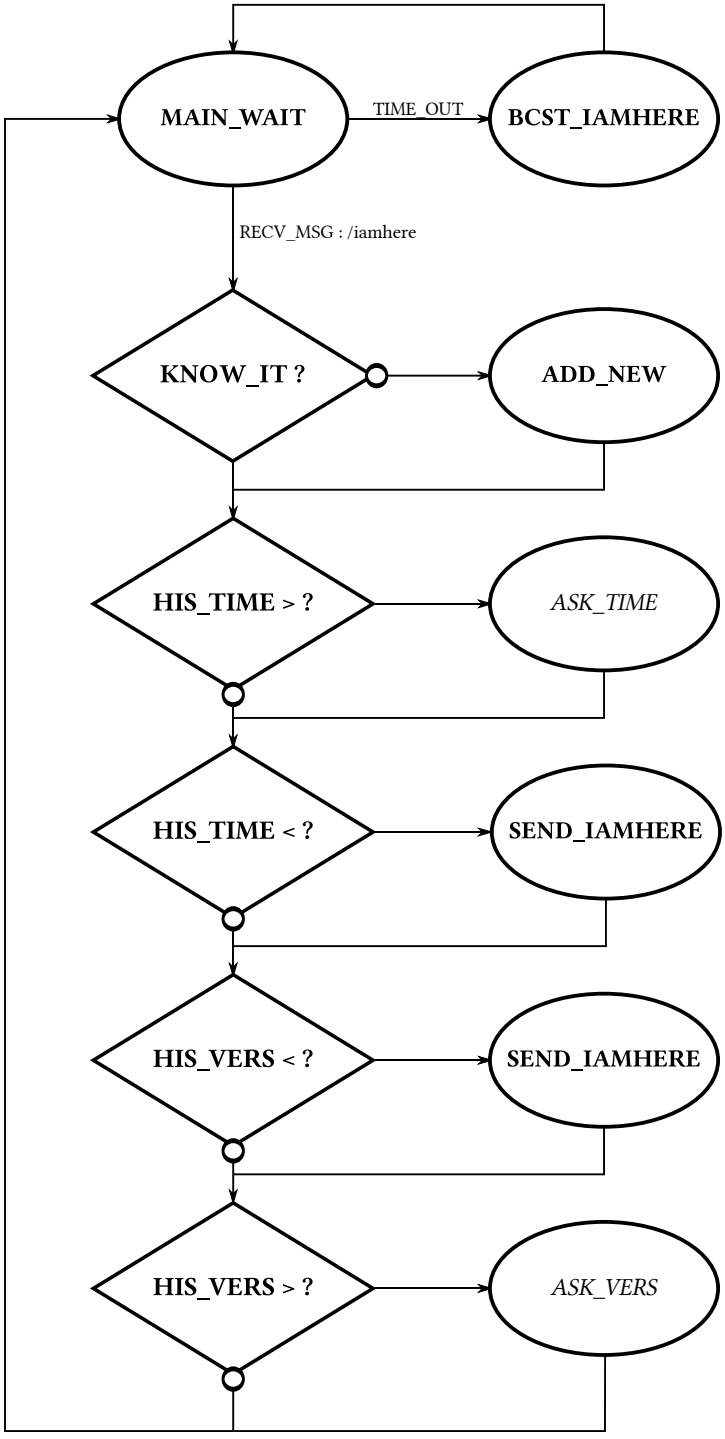

Figure 1 – FSM partielle de la présentation sur le réseau

3.1.3 États correspondants à la présentation sur le réseau

Les différents états présentés sur la FSM figure 1 sont listés et détaillés ci-dessous :

— **MAIN_WAIT** : État principale d'attente.
— **BCST_IAMHERE** : Broadcast le message **iamhere** [6].
— **ADD_NEW** : Ajoute la carte dont vient d'être reçu le message.
— **SEND_IAMHERE** : Envoie le message **iamhere** à l'émetteur.

Les états *ASK_TIME* et *ASK_VERS* sont en réalité un ensemble d'états et seront détaillés plus tard.

3.2 Synchronisation du temps

La partie de synchronisation du temps est regroupé dans le projet sous l'acronyme RTP pour *Relative Time Protocole*. Ce protocole créé pour le projet est inspiré de protocole PTP [7][5][6] proche du protocole NTP [8][7].La différence principale entre RTP et PTP est le fait qu'il soit totalement décentralisé. Il n'y a pas différent niveau ni de réelle référence absolue.

Le but du protocole n'est pas de partager le temps supposé absolue dans le référentiel Terrestre et basé sur un référence commune aux autres éléments en dehors du réseau [9]. La vocation de RTP est que toutes les cartes partage une même référence de temps avec une grande précision.

Au démarrage du protocole est fixé un *timetag* basé sur le *timestamp*[8] de la carte. La valeur absolue de celui-ci n'est pas important, sont rôle est de représenté une référence de synchronisation. La carte qui a le *timetag* le plus élevé va imposer, via RTP, son temps et son *timetag* aux autres de sorte qu'il n'y ait sur le réseau qu'un seul *timetag* de présent.

3.2.1 Principe du protocole

Pour établir un temps commun avec précision malgré les latences du réseau le protocole va tenter d'établir un temps moyen de parcours entre les cartes à synchroniser en cherchant à garder une répartition proche de

6. Message présenté par la table 3.1.1
7. Precision Time Protocole
8. Network Time Protocol
9. Qu'une carte indique le 1er janvier 1970 n'est pas un problème ici pour les objectif du protocole.

ces temps de parcours.

Pour cela la référence va envoyer un premier message de **ping** en notant la date d'émission, puis lorsque la carte à synchroniser le recevra elle répondra immédiatement via un message **pong**. Une fois ce message reçu par la référence il va pouvoir en déduire un premier temps de parcours.

$$dt = \frac{t_{pong} - t_{ping}}{2}$$

Ce processus est répété au moins 5 fois. Lorsqu'un nouveau temps de parcours est trouvé il vient d'ajouter à la pile en effaçant le premier de fait qu'il n'y ai toujours que les 5 plus jeunes éléments dans la pile.

À chaque fois qu'un aller-retour est fait et que la pile est pleine l'écart maximum entre le parcours le plus long et celui le plus court est calculé :

$$accuracy = \max(stack_{dt}) - \min(stack_{dt})$$

Ce temps *accuracy* doit être inférieur à une limite fixée, si c'est le cas la synchronisation est possible et un message est envoyé par la référence avec les information nécessaires. Sinon le seuil est légèrement augmenté et ne devra de toute manière pas dépasser un seuil maximum, puis l'opération continue jusqu'à trouve un temps de parcours correcte.

3.2.2 Trames OSC pour le protocole RTP

Sont listées ci-dessous les différentes trames OSC utilisées pour la synchronisation du temps.

Trame OSC via le protocole ACK

IP	Port	Protocole	Adresse
X.X.X.X[10]	1782	UDP	/rtp/asktime

Table 4 – Trame OSC envoyée pour demander une synchronisation du temps

10. Ici X.X.X.X représente l'adresse IP de la référence de temps

Trame OSC via le protocole ACK

IP	Port	Protocole	Adresse
Y.Y.Y.Y[11]	1782	UDP	/rtp/ping

Arguments [12]

Champs	Type	Valeur
rand1	int	`rand()`
rand2	int	`rand()`
rand3	int	`rand()`
rand4	int	`rand()`
rand5	int	`rand()`

Table 5 – Trame OSC de ping

Trame OSC via le protocole ACK

IP	Port	Protocole	Adresse
X.X.X.X	1782	UDP	/rtp/pong

Arguments

Champs	Type	Valeur
rand1	int	`rand()`
rand2	int	`rand()`
rand3	int	`rand()`
rand4	int	`rand()`
rand5	int	`rand()`

Table 6 – Trame OSC de pong

Tous les messages du protocole RTP utilisent le protocole ACK. Ceci s'explique pour deux raisons. Premièrement car une fois une synchronisation trouvée, ce qui est un processus relativement long, il est souhaitable que le message contenant les informations de synchronisation ne soit pas perdu.

12. Ici Y.Y.Y.Y représente l'adresse IP de la carte qui demande un synchronisation du temps

12. Les arguments des messages **ping** et **pong** sont la uniquement pour que le message **sync** ait la même taille pour que le temps de parcours calculer dans leurs cas ait le plus de chance possible d'être proche de celui du message **sync**

Trame OSC via le protocole ACK

IP	Port	Protocole	Adresse
Y.Y.Y.Y	1782	UDP	/rtp/sync

Arguments

Champs	Type	Valeur
time_s	int	*Temps de référence en secondes*
time_ns	int	*Temps de référence en nanosecondes*
dt	int	*Temps de parcours moyen*
accuracy	int	*Précision moyenne de la synchronisation*
timetag	int	*Timetag de la référence de temps*

Table 7 – Trame OSC de syncronisation

Enfin, vu que le processus de synchronisation du temps est basé sur des FSM qui évoluent l'une par rapport à l'autre, chaque perte de message pourrait compromettre l'ensemble de la synchronisation.
De plus, pour garder les temps de parcours les plus identiques possible, si le message **sync** utilise le protocole ACK, légèrement plus lent que de l'OSC classique, il est logique que les messages **ping** et **pong** l'utilisent également.

3.2.3 FSM de la synchronisation du temps côté référence

Cette FSM a déjà été décrite dans le document de présentation des Machines à nombre Fini d'États[1].

Les états de la FSM s'occupant de la synchronisation du temps côté référence sont les suivants :

— **MAIN_WAIT** : État d'attente principale du protocole.
— **START_SYNC** : État initialisant les variables nécessaires à la synchronisation et un timer pour éviter de rester bloquer lors de la synchronisation.
— **SEND_PING** : État envoyant un **ping** et notant la date d'envoi

dans une variable.
— **WAIT_PONG** : État d'attente après l'envoie d'un **ping**
— **RECV_PONG** : État de réception d'un **pong** et de calcul du temps de parcours puis de l'éventuelle synchronisation
— **SEND_SYNC** : État envoyant un message de synchronisation

La FSM représentant le fonctionnement est la suivante :

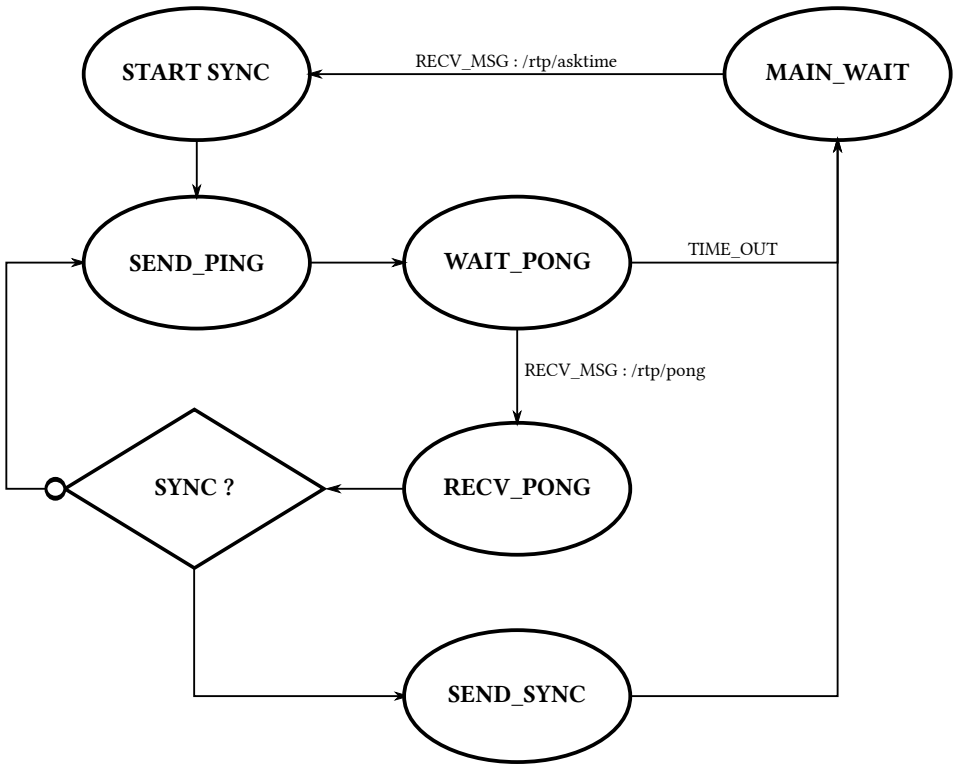

FIGURE 2 – FSM de la synchronisation du temps côté référence

3.2.4 FSM d'une demande de synchronisation du temps

Les états de la FSM demandant une synchronisation du temps sont le suivants :
— **MAIN_WAIT** : État d'attente principale du protocole.
— **ASK_SYNC** : Envoie d'un message **asktime** à une référence.
— **WAIT_PING** : État d'attente après l'envoie d'un **pong** ou après la demande de synchronisation.
— **SEND_PONG** : Envoie d'un message **pong** en réponse à un **ping**

— **RECV_PONG** : État de réception d'un **pong** et de calcul du temps de parcours puis de l'éventuelle synchronisation
— **SYNC_TIME** : État faisant suite à la réception d'un message **sync** synchronisant le temps en fonction des valeurs reçues.

La FSM représentant le fonctionnement est la suivante :

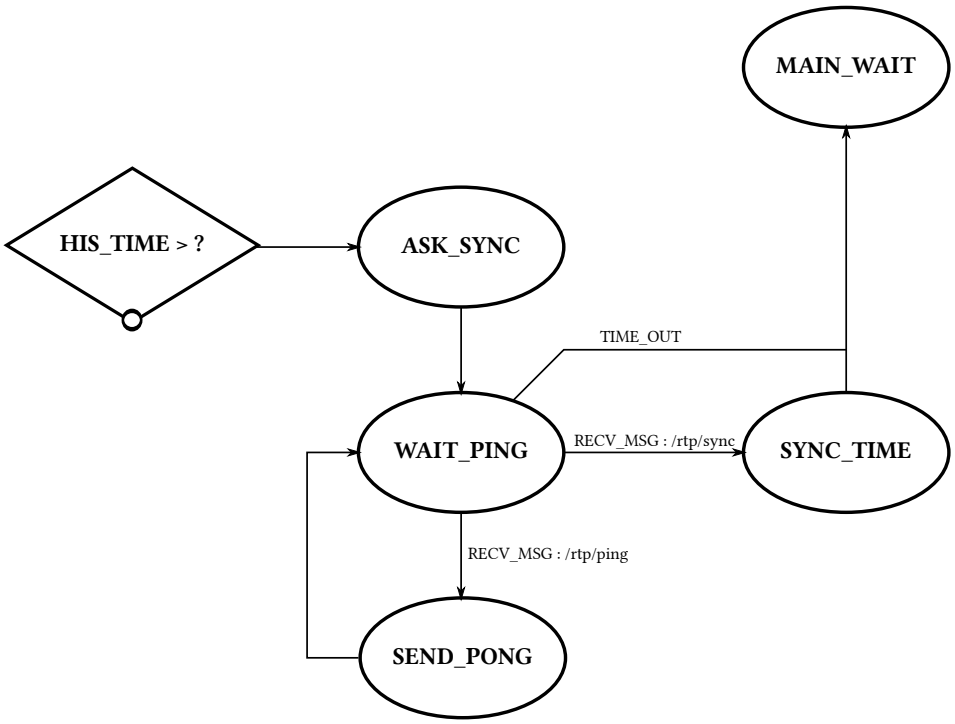

FIGURE 3 – FSM d'une demande de synchronisation du temps

3.3 Synchronisation du scénario

Références

[1] Olivier RADISSON. *Implémentation des Machines à nombre Fini d'Etats*. Jan. 2015.

[2] (en). *Wi-Fi*. Jan. 2015. URL : https://en.wikipedia.org/wiki/Wi-Fi (version du 08/01/2015).

[3] (en) J. POSTEL. *User Datagram Protocol*. URL : https://tools.ietf.org/html/rfc768 (version du 08/01/2015).

[4] (en) Matt WRIGHT. *The Open Sound Control 1.0 Specification*. URL : http://opensoundcontrol.org/spec-1_0 (version du 08/01/2015).

[5] (en) NIST US DEPARTMENT OF COMMERCE. *Introduction to IEEE 1588*. URL : http://www.nist.gov/el/isd/ieee/intro1588.cfm (version du 09/01/2015).

[6] (en) EndRun TECHNOLOGIES. *Precision Time Protocol*. URL : http://www.endruntechnologies.com/pdf/PTP-1588.pdf (version du 09/01/2015).

[7] (en) Network Time FOUNDATION. *Network Time Protocol*. URL : http://www.ntp.org/ (version du 09/01/2015).

[8] (en). *Unix time*. Jan. 2015. URL : https://en.wikipedia.org/wiki/Unix_time (version du 09/01/2015).

[9] *Automate fini*. Déc. 2014. URL : http://fr.wikipedia.org/wiki/Automate_fini (version du 08/01/2015).

[10] (en) Jan DACIUK. *Finite State Automata*. URL : http://galaxy.eti.pg.gda.pl/katedry/kiw/pracownicy/Jan.Daciuk/personal/thesis/node12.html (version du 08/01/2015).