**RENPAR'15 / CFSE'3 / SympAAA'2003**
**La Colle sur Loup, France, 15 au 17 octobre 2003**

# Un algorithme d'auto synchronisation distribuée de flux audio dans le concert virtuel réparti

Bouillot Nicolas

CNAM-CEDRIC
292 rue St Martin
74141 Paris cedex 03 France
bouillot@cnam.fr

**Résumé**
La croissance des systèmes de transmission de flux multimédia sur réseau nous laisse envisager la possibilité de fournir à des musiciens physiquement éloignés un médium permettant de jouer de la musique ensemble en temps réel. L'interaction musicale traditionnelle impose que chaque musicien entende les autres de la même façon. Il est alors nécessaire pour des musiciens physiquement éloignés de synchroniser le mixage des flux pour qu'ils s'entendent tous "virtuellement ensemble". Nous proposons dans ce document l'algorithme d'auto synchronisation distribuée de flux audio qui compense les retards introduits par le réseau et qui permettra aux différents participants d'une performance multimédia musicale répartie d'avoir une vision semblable de celle-ci.

**Mots-clés :** réseau, multimédia interactif, synchronisation, musique, multicast

## 1. Réseau, temps réel et performances multimédia musicales réparties

Dans le contexte de musique "Live", l'interaction musicale se fait par les différents signes visuels et les conventions prédéterminées sur le morceau de musique joué (comme par exemple une suite d'accords avec un thème pour un morceau de Jazz). En même temps, le contenu musical ajouté par chaque musicien informe les autres sur l'évolution possible du morceau. C'est le cas par exemple des ensembles de percussions qui fonctionnent en phrases d'appels rythmiques. Toutes ces interactions sont possibles car les musiciens évoluent dans un même environnement (la scène, la pièce...) qui leur permet en particulier de s'entendre tous en même temps de façon synchronisée. Dans notre contexte de musique "Live" distribuée où les musiciens sont physiquement éloignés, les supports technologiques tels réseaux et systèmes d'exploitations sont asynchrones. Etant des sources de flux audio PCM[1] et étant physiquement répartis (donc percevant de la latence à l'audition), les musiciens doivent être synchronisés pour avoir l'impression de jouer ensemble de façon cohérente. Les flux audio étant produit à la volée sur les différents sites, nous ne pouvons pas faire la synchronisation directement dans les flux de départ, comme cela est fait pour les flux multimédia pré-enregistrés comme de la vidéo, où le son et l'image sont déjà synchronisés avant l'envoi grâce à un estampillage commun. Nous avons donc choisi d'ajouter un mécanisme assurant la propriété d'écoute cohérente directement à la réception des flux. Ce mécanisme est un consensus entre auditeurs (les musiciens qui s'écoutent) sur le mixage des flux audio après réception. Sachant que nous désirons avoir la latence la plus faible possible pour conserver la meilleure interaction, le délai ajouté dans les tampons de réception pour assurer la cohérence doit être le plus faible possible. L'algorithme que nous proposons ici est qualifié "d'auto" car il s'exécute automatiquement chez chacun des récepteurs dès que des échantillons sont reçus de chacune des sources de son.

Dans le cas d'Internet, nous avons globalement un réseau asynchrone sans réservation de ressources, il faut donc compter sur des temps de latence significatifs (de quelques millisecondes à plusieurs secondes) et une gigue dépendant de l'état de charge du réseau et des pertes. Certains pensent que ces

[1] Pulse Code Modulation : L'encodage utilisé pour les CD audio

délais variables restent trop importants pour permettre des performances audio distribuées interactives [13, 1]. Malgré ces délais, nous pensons pouvoir permettre une interaction entre les différents musiciens. Rappelons que certains musiciens sont capables de lire une partition à plusieurs mesures de l'endroit où ils sont en train de jouer. L'exemple de l'acoustique réverbérante des cathédrales imposait au moyen âge des contraintes de tempo : le chant grégorien devait être composé de longues notes afin de rendre les textes compréhensibles. Rappelons aussi que dans un orchestre symphonique, le chef d'orchestre est toujours en avance vis-à-vis des musiciens qui ont leur propre vitesse. Ces exemples montrent que les musiciens sont capables de jouer de la musique malgré certains décalages. Considérant cela, nous pensons être capables de fournir un outil d'interaction temps réel à des musiciens distants physiquement en y associant un mode d'interaction dédié. Soit en minimisant les temps de latence de bout en bout, soit en exagérant ces temps pour obtenir un retard cadré sur la métrique du morceau joué. Le choix dépend directement de la façon dont les musiciens peuvent s'adapter aux latences. Dans notre contexte d'expérimentation (réseau de type LAN et prochainement de type MAN), la latence peut être très proche du seuil de perception des décalages par l'oreille humaine (environs 20ms). Nous avons donc choisi de minimiser la latence. Pour les musiciens, l'adaptation à la latence n'en sera que plus confortable. Assurer une perception cohérente de la musique produite est l'objectif de l'algorithme d'auto synchronisation distribuée de flux audio.

Plusieurs expériences de performances multimédia temps réel sur Internet ont déjà été expérimentées. Plusieurs d'entre elles [7, 9, 18] utilisent la norme MIDI[2]. Dans [7], aucun mécanisme de synchronisation des différents flux n'est mentionné. La latence entre la production et la consommation de sons n'étant pas prévue pour être constante, certaines variations sont perçues à l'audition. Dans le système d'enseignement du piano de J.P. Young et I. Fujinaga [18], un unique musicien enseigne pendant que les élèves écoutent, il n'y a pas besoin ici de jouer de façon synchronisée puisque pas ensemble. L'équipe de Masataka Goto [9, 11] a développé VirJa, un outil de session de Jazz virtuelle où un humain peut jouer de la musique avec deux processeurs. Là aussi, aucun mécanisme de synchronisation des flux n'y est spécifié. Une expérience de musique à distance a vu le jour grâce au protocole RMCP utilisé dans VirJa. *Open RemoteGIG* [10] permet de jouer de la musique à distance en MIDI avec des délais constants modulo le tempo et la métrique du morceau. Dans ce système, les musiciens des différents sites n'entendront pas la même chose, en effet le retour de ce qui est joué localement ne tient pas compte de la latence. L'exemple décrit dans [11] nous montre ce décalage : pendant que le joueur joue et s'entend à la deuxième mesure d'un morceau, il entend ce qu'ont fait les autres à la première.
Malgré l'intérêt que peut représenter de telles performances, le protocole MIDI permet de mettre de côté certains aspects de la transmission de flux audio. Le format descriptif de la note en MIDI permet d'économiser de la bande passante. Mais aussi il diminue le domaine des sons transportés. Il est donc difficile de faire une analogie directe avec les contraintes imposées par les flux audio PCM.
Le transport de flux audio PCM temps réel a été expérimenté dans [17, 4]. Les auteurs ont utilisé un studio d'enregistrement pour échantillonner une performance ayant lieu dans un autre pays. Les musiciens étant localisés au même endroit, la synchronisation telle que nous l'entendons n'avait pas lieu d'être. Deux expériences ont été testées avec des musiciens physiquement distants dans le cadre du projet Global Visual Music [15]. La première (*Lemma 1*), s'est faite sur réseau local avec deux musiciens en interactions mais dans la même pièce. Lors de *Lemma 2*, les musiciens étaient distants physiquement mais ne s'entendaient et ne se voyaient pas de façon synchronisée, "le piano et les percussions étaient analysés sur chaque site afin de faire émerger à l'autre bout les informations de différentes façons". Par exemple sous forme d'image abstraite indiquant certains aspects du jeu du musicien distant, comme par exemple la vélocité.

L'objet de cet article est de proposer un algorithme de synchronisation de flux audio assurant aux musiciens distants une écoute cohérente du jeu de groupe. A notre connaissance, dans le domaine de l'informatique musicale en réseau, il n'existe pas de travail équivalent à ce que nous proposons dans ce papier. Le paragraphe 2 présente le projet du concert virtuel réparti, le paragraphe 3 la synchronisation répartie, le paragraphe 4 l'implémentation de l'algorithme dans le prototype du projet. Enfin, le paragraphe 5 conclut.

[2] Musical Instrument Digital Interface

## 2. Le concert Virtuel réparti

La conjonction des travaux des laboratoires de recherche de l'IRCAM et du CNAM-CEDRIC doit permettre la mise en place d'un orchestre virtuellement réparti avec des musiciens physiquement éloignés. Le CNAM-CEDRIC est chargé de la conception d'une architecture système distribuée pour supporter cette métaphore de l'orchestre virtuel. La figure 1 présente l'architecture générale du projet. Les musi-

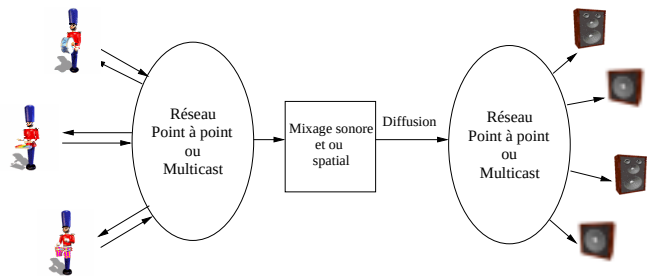



FIG. 1 – *Le concert réparti, une architecture générale*

ciens sont physiquement répartis mais doivent jouer "virtuellement ensemble" en temps réel. Le retour ainsi que l'écoute des autres musiciens se fait grâce à un mécanisme de synchronisation des flux audio développé comme élément de la librairie du logiciel jMax[3]. Cette synchronisation suppose une latence constante au niveau de la restitution du son. Les musiciens joueront en avance par rapport à ce qu'ils entendront mais pourront alors jouer de la musique à plusieurs de façon synchronisée. Afin de permettre à un public d'assister au concert virtuel, un ingénieur du son collectera les flux audio des musiciens pour faire le mixage sonore et spatial de la scène. Le mixage spatial consiste à positionner les sources sonores dans un espace à trois dimensions. Afin de permettre ce type de mixage, le son transporté sera du son multi-canaux[4]. Le mixage pourra se faire soit à la source directement par un contrôle distant des périphériques des musiciens, soit par un traitement des flux après réception. Les travaux en cours sur le contrôle distant des périphériques utilisent l'implémentation OpenTaz[5] [2] de la norme TASE 2.0 [12], un protocole de gestion de données temps réel sur CORBA 2.

## 3. La restitution cohérente des flux audio

### 3.1. Introduction

L'objet de l'algorithme est de synchroniser la consommation d'événements (ici des échantillons audio) entre plusieurs sites, sachant que ces événements sont générés à la volée par plusieurs sites distants les uns des autres. Cette synchronisation va fournir une consommation fortement cohérente des événements issus du système de production. Ce système étant composé de différentes "usines" indépendantes, ce sont les consommateurs qui vont se mettre d'accord sur l'ensemble d'événements à consommer simultanément.
L'autre objectif de cet algorithme est de minimiser le temps entre la production et la consommation (la latence de bout en bout) pour chaque producteur puisque chaque consommateur va se synchroniser par rapport à celui qui a la plus grande latence. Nous aurions pu prendre pour chaque producteur une latence délibérément grande et ainsi compenser les retards du réseau. Notons que pour nous, les producteurs peuvent être aussi des consommateurs. Minimiser les temps de latence permet donc d'obtenir les performances les plus proches possibles du jeu musical traditionnel.

[3] jMax [6, 5] est un environnement de programmation visuelle pour des applications musicales et multimédia temps réel. Il est disponible à l'adresse http : / / www.ircam.fr / jmax
[4] permettant la reproduction d'un effet tridimensionnel (positionnement dans l'espace du son)
[5] http : / / savannah.nongnu.org / projects / opentaz. Mars 2003

Nous considérons ici les liens de communication entre le producteur et le consommateur comme étant sans pertes. Ce qui est acceptable dans le contexte d'expérimentations actuel (LAN) mais devra être reconsidéré dans les prochains essais (MAN). Dans un contexte de réseau avec pertes, il faudra les compenser et assurer que la consommation des échantillons est continue et sans trous.
Pour que l'on puisse utiliser cet algorithme, il faut connaître à l'avance le nombre de sources du système, le nombre de consommateurs, ainsi que les adresses multicast de réception des événements et de réception des statistiques nécessaires à la synchronisation. Il faut donc que le trafic multicast soit possible entre les différents sites. Notons que l'algorithme peut aussi fonctionner dans un contexte unicast. Le principal problème étant la démultiplication des flux audio (un flux devra être émis $n$ fois s'il y a $n$ récepteurs).
La figure 1 est un exemple de système à producteurs et consommateurs multiples. En effet, dans ce système, les musiciens ainsi que les auditeurs doivent tous entendre la même musique, comme s'ils étaient dans un même lieu. Les producteurs d'événements sont les musiciens. Un événement correspond à un échantillon sonore. Les spectateurs distants sont consommateurs. Les musiciens le sont aussi puisqu'ils doivent s'entendre un à un. La communication entre instrumentistes se faisant par le réseau, il faut minimiser les temps de latence production/consommation et garantir une écoute commune fortement cohérente qui rendra le jeu de groupe possible.

Voici une description générale du système :
- Soit $n_p$ le nombre de producteurs du système
- Soit $n_c$ le nombre de consommateurs qui se synchronisent, c'est le nombre de processeurs participant à l'algorithme
- Soit $pc_1 \cdots pc_{n_c}$ les processeurs consommateurs et $pp_1 \cdots pp_{n_p}$ les processeurs producteurs
- Chaque $pp_i$ $(1 < i < n_p)$ produit régulièrement des événements $ev_k^i$ qui seront datés par une horloge logique de production notée $hlp_i(ev_k^i)$. Il y a une notion d'ordre dans les événements, $ev_{k+1}^i$ est l'événement suivant de $ev_k^i$ sur le processeur $pp_i$. L'horloge logique est en fait un compteur, i.e. $hlp_i(ev_{k+1}^i) = hlp_i(ev_k^i) + 1$. Les processeurs $pp_i$ $(1 < i < n_p)$ envoient leurs événements sous forme d'un flux respectant l'ordre local.
- Les $pp_i$ $(1 < i < n_p)$ envoient les événements ainsi que leur date $(ev_k^i, hlp_i(ev_k^i))$ en flux à chaque récepteur
- Chaque récepteur $pc_j$ possède une horloge logique de consommation des événements reçus notée $hlc_j(ev_k^i)$ avec $1 < j < n_c$ et $ev_k^i$ un événement reçu de $pp_i$. Les événements sont consommés chez le récepteur comme ils sont envoyés par le producteur, i. e. si $hlp_i(ev_{k+1}^i) = hlp_i(ev_k^i) + 1$, alors $hlc_j(ev_{k+1}^i) = hlc_j(ev_k^i) + 1$ (respect de l'ordre local de chaque source). De plus si $hlc_j(ev_k^i) = hlc_j(ev_g^h)$ alors $ev_k^i$ et $ev_g^h$ sont consommés simultanément par le récepteur $pp_j$ (addition des sons)

La propriété de cohérence entre les consommateurs s'énonce donc de la façon suivante :

$$\forall m, n \in [1 \ldots n_c]\ hlc_m(ev_{k_1}^1) = hlc_m(ev_{k_2}^2) = \cdots = hlc_m(ev_{k_{n_p}}^{n_p})$$
$$\Rightarrow hlc_n(ev_{k_1}^1) = hlc_n(ev_{k_2}^2) = \cdots = hlc_n(ev_{k_{n_p}}^{n_p})$$

Autrement dit, à la fin de l'algorithme, si le processeur $pc_i$ consomme les événements $(ev_{k1}^1 \cdots ev_{k_{n_p}}^{n_p})$ simultanément, alors les autres processeurs $pc_j$ consommeront aussi ces événements de façon simultanée. De plus la latence ajoutée dans les files d'attente de chaque consommateur (pour assurer la consommation cohérente) est minimale. Dans le cas de transferts de flux audio avec plusieurs sources, les récepteurs auront la même perception du mixage des flux.

### 3.2. L'algorithme

Regardons l'algorithme d'envoi des estampilles (figure 2) exécuté par chaque consommateur :
$estampille[i]$ va contenir les valeurs des estampilles (de l'horloge $hlp_i$) d'événements $(ev_{k_1}^1 \cdots ev_{k_{n_p}}^{n_p})$ joués simultanément. Ce qui s'énonce pour chaque consommateur par :
$\forall i\ estampille_{pc}[i] := hlp_i(ev_{k_i}^i)$ et $hlc_{pc}(ev_{k_i}^i) == t$ avec t un moment quelconque ou $pc$ (le consommateur qui exécute la procédure) a commencé à consommer des événements de chaque producteur. Le tableau $estampille$ contient un "instantané" (celui de la date $t$) des estampilles jouées simultanément pour le processeur $pc$ où s'exécute la procédure.

```
1  int estampille[n_p]
2  Attendre d'avoir reçu des événements des n_p producteurs
3  A la date t : ∀i ∈ [1...n_p] calculer estampille_pc[i]
4  Envoyer estampille_pc aux n_c consommateurs
```

FIG. 2 – Algorithme d'envoi des estampilles

```
5   sync = faux        variable d'arrêt
6   dif_temp[n_c]

7   Recevoir les n_c tableaux d'estampilles dans les tableaux
     estampille_pc_k correspondants
8   Tant que sync != vrai
9      Choisir pp_ref comme max(pp_x) avec 1 < x < n_p et
          pp_x n'ayant jamais été pris comme référence
10     Pour i de 1 à n_p
11        pp_ajust := pp_i
12        Pour j de 1 à n_c
13           dif_temp[pp_j] := estampille_pc_j[pp_ref] − estampille_pc_j[pp_ajust]
14        FinPour
15        dif[pp_i] := max_j(dif_temp[pp_j])
16     FinPour;
17      Si ∀i ∈ [1...n_c] dif[pp_i] > 0 alors
18       sync := vrai
19       Pour i de 1 à n_p
20          ajust[pp_i] := dif[pp_i] − (estampille_pc_local[pp_ref] − estampille_pc_local[pp_i])
21           Ajouter ajust[pp_i] événements dans la file de
               consommation d'événements issu de pp_i
22       Finpour
23  FinTantque
```

FIG. 3 – Algorithme de calcul de la synchronisation

Regardons maintenant l'algorithme de calcul de la synchronisation (figure 3) :
Soit $pp_{ref}$ le producteur choisi comme référence pour les calculs.
$pp_{ajust}$ est le producteur avec lequel nous calculons l'ensemble des écarts avec le producteur référence. Pour chaque flux, seul l'écart maximum sera pris en compte. Chaque processeur producteur sera au moins une fois processeur $pp_{ajust}$ au cours de la vie de l'algorithme.
$estampille_{pc_i}[]$ est le tableau d'estampilles reçu de $pc_i$. La valeur $estampille_{pc_i}[pp_j]$ est donc l'estampille d'un événement joué par $pc_i$ et issu de $pp_j$.
$ajust[pp_i]$ va correspondre au nombre d'événements à ajouter dans le flux reçu de $pp_i$ pour effectuer la synchronisation. Si la consommation d'un événement prend un temps $t$ alors le délai ajouté perçu par le consommateur sera $t * ajust[pp_i]$. Après l'ajout des événements (donc du délai), $hlc_{Local}(ev_l^k) = hlp_k(ev_l^k) + ajust[pp_k]$ pour tous les flux reçus.
$dif[pp_i]$ contient $\max_j(estampille_{pc_j}[pp_{ref}] - estampille_{pc_j}[pp_{ajust}])$. Ce sont les retards à avoir entre $pp_{ref}$ et $pp_i$.
$pc_{local}$ est le consommateur sur lequel l'algorithme se déroule.
L'idée principale de l'algorithme présenté figure 3 est de prendre un des processeurs $pp_i (1 < i < n_p)$ comme référence de calcul (ligne 9). Ensuite, les différences calculées à la ligne 13 permettront d'estimer le processeur $pp_k$ ayant le plus de retard par rapport à $pp_{ref}$. Pour chaque flux, la compensation appliquée sera alors fonction de celui calculé avec $pp_k$. Si le processeur choisi comme référence ($pp_{ref}$) ne convient pas, un autre processeur producteur sera choisi comme référence. Afin que le choix de la

référence soit consensuel, chaque consommateur de flux testera les processeurs $pp_i (1 < i < n_p)$ du plus grand au plus petit. Cela permet à chacun de s'arrêter sur la même référence.
Chaque consommateur des flux audio exécutera les deux procédures, l'envoi et le traitement des estampilles. Les variables $n_p$ et $n_c$ sont particulières car fournies par l'utilisateur. Elles doivent donc être les mêmes pour chacun.
La synchronisation devant être consensuelle, il est nécessaire que l'algorithme se déroule avec les mêmes valeurs chez chaque participant. A la ligne 9, choisir $pp_{ref}$ comme étant le producteur de plus grand identifiant permet de faire un choix commun sans négociation. La boucle Pour de la ligne 10 calcule pour chaque $pc_i$ les écarts entre l'estampille mesurée pour $pp_{ref}$ et les estampilles des autres processeurs producteurs. Seuls les maximas (parmi tous les processeurs $pc_i$ confondus) sont conservés. Pour tout processeur $pc_i$, si $pp_{ref}$ n'est pas toujours en retard par rapport à un autre processeur $pp_j$, alors $dif[pp_i] > 0$ reste vraie. Cela veut dire que tout le monde pourra ajouter du retard sur les flux pour achever le consensus et permettre alors une consommation identique. Si $dif[pp_i] < 0$, alors il faut changer de producteur référence et recommencer les calculs (retour ligne 8).
Une fois les plus grands écarts calculés, chaque consommateur va compenser le retard manquant sur ses flux locaux (ligne 19 à 22). La consommation des événements issus des processeurs $pp_i$ devient alors identique entre tous les processeurs $pc_j$.

Pour que l'algorithme termine, nous devons être certains que nous allons trouver un producteur référence qui satisfasse la formule $dif[pp_i] > 0$ (ligne 17). Autrement dit, il faut que chacun des processeurs $pp_i$ soit perçu au moins une fois en retard par rapport au processeur $pp_{ref}$. Cela peut être perçu chez n'importe quel processeur $pc_j$ puisque l'on ajuste les flux par rapport au consommateur qui met le plus de temps à recevoir les événements. Nous allons maintenant démontrer que cette propriété est vérifiée.

### 3.3. Terminaison de l'algorithme

Nous devons démontrer qu'il existe toujours un processeur référence $pp_{ref}$ qui rende la formule $\forall i dif[pp_i] > 0$ vrai avec $1 < i < n_p$ et $i \neq ref$ (ligne 17 de l'algorithme). Autrement dit, il faut que chacun des processeurs $pp_i$ ($1 < i < n_p$ et $i \neq ref$) soit perçu au moins un fois en retard par rapport au processeur référence $pp_{ref}$. Remarquons que nous excluons la comparaison du processeur référence avec lui-même car cela n'a pas de sens d'être en avance ou en retard par rapport à soi même. En fait, nous ne considérons comme intéressante dans l'algorithme que les différences positives.

**Définition 1** *Soit* $f(i,j) = i \xrightarrow{k} j$ *la relation binaire telle que le processeur* $pp_i$ *est perçu par un autre processeur* $pc_k$ *comme en retard par rapport au processeur* $pp_j$.

De par sa nature, la relation f est anti-réflexive ($f(i,i)$ n'a pas de sens) et est anti-symétrique :

$$\neg(i \xrightarrow{k} j) \Leftrightarrow j \xrightarrow{k} i \tag{1}$$

En effet, pour le processeur $pc_k (1 < k < n_c)$, si le processeur $pp_i$ est perçu comme en avance (respectivement en retard) par rapport au processeur $pp_j$ alors $pp_j$ est perçu par $pc_k$ comme en retard (respectivement en avance) par rapport à $pp_i$. Notons qu'en cas d'estampilles identiques, $pc_k$ peut départager $pp_i$ et $pp_j$ avec leurs identifiants. De plus, sachant que dans notre cas, une vision globale des retards correspond à une vision unique :

$$\forall k(\neg(i \xrightarrow{k} j)) \Rightarrow \forall k(j \xrightarrow{k} i) \tag{2}$$

La relation f est transitive :

$$(i \xrightarrow{k} j) \wedge (j \xrightarrow{k} l) \Rightarrow i \xrightarrow{k} l$$

En effet, si pour $pc_k$, $pp_i$ est en retard par rapport à $pp_j$ qui est lui même en retard par rapport à $pp_l$, alors $pp_i$ est en retard par rapport à $pp_l$.
De plus, de par la propriété de conjonction du quantificateur $\forall$ :

$$\forall k(i \xrightarrow{k} j) \wedge \forall k(j \xrightarrow{k} l) \Rightarrow \forall k((i \xrightarrow{k} j) \wedge (j \xrightarrow{k} l)) \Rightarrow \forall k(i \xrightarrow{k} l) \tag{3}$$

Notons alors que f est une relation d'ordre total. Elle l'est car elle correspond à l'unique vision d'un des consommateurs ($pc_k$). Le théorème ci-dessous ne peut exploiter cette propriété car il considère plusieurs consommateurs avec un ordre total propre à chacun.

**Théorème 1** *Dans la procédure de traitement des estampilles, il existe toujours une source référence* $pp_{ref}$ *telle que :*
*Chacun des processeurs* $pp_i$ $(1 < i < n_p$ *et* $i \neq ref)$ *du système est perçu par au moins un des processeurs* $pc_k$ $(1 < k < n_c)$ *comme ayant du retard par rapport à* $pp_{ref}$.

Le théorème 1 s'énonce donc comme suit (le processeur $pp_j$ vérifiant cette formule est le processeur référence) :

$$\exists j \forall i \exists k (i \xrightarrow{k} j) \tag{4}$$

A partir de maintenant, nous allons restreindre le domaine de i et j à $1 < i < n_p$, $1 < j < n_p$ et $i \neq j$ (conforme à la définition 1).

**Preuve 1** *Nous allons montrer que le théorème 1 est vérifié. Pour cela, nous allons montrer que la formule 4 est vraie. Il suffit alors de montrer que la contraposé de cette formule est fausse. Donc que la formule 5 est fausse :*

$$\neg \textit{(formule 4)} \Rightarrow \neg(\exists j \forall i \exists k (i \xrightarrow{k} j)) \Rightarrow \forall j \exists i \forall k \neg (i \xrightarrow{k} j)$$

*Donc, par la formule 1 :*

$$\Rightarrow \forall j \exists i \forall k (j \xrightarrow{k} i) \tag{5}$$

*Nous allons procéder par l'absurde pour démontrer que cette formule est fausse. Nous allons alors la supposer vraie, donc si* $j = x_1$ *alors il existe* $i = x_2$ *tel que* $\forall k (x_1 \xrightarrow{k} x_2)$. *Sachant que* i *et* j *évoluent dans le même domaine, choisissons maintenant* $j = x_2$. *On sait qu'il existe* $i = x_3$ *tel que* $\forall k (x_2 \xrightarrow{k} x_3)$. *Nous pouvons continuer de même jusqu'à obtenir* $\forall k (x_{n_p} \xrightarrow{k} x_{n_p+1})$ *(*$j = x_{n_p}$ *et* $i = x_{n_p+1}$*). Nous aurons donc :*

$$\forall k (x_1 \xrightarrow{k} x_2) \wedge \forall k (x_2 \xrightarrow{k} x_3) \wedge \ldots \wedge \forall k (x_{n_p} \xrightarrow{k} x_{n_p+1}) \tag{6}$$

*A ce moment, nous aurons trouvé au moins un doublon dans les valeurs que peuvent prendre les membres de gauche et les membres de droite des différents facteurs puisque qu'il y en a* $n_p$ *différents possibles et qu'ils sont au nombre de* $n_p + 1$. *Prenons alors* $x_a$ *et* $x_b$ *ce doublon (*$x_a = x_b$*). Comme* $\forall k (x_n \xrightarrow{k} x_{n+1})$ *pour* $n \in [1 \ldots n_p]$, *alors par définition,* $x_n \neq x_{n+1}$. *Il existe donc* $x_c$ *tel que :*

$$\forall k (x_a \xrightarrow{k} x_{a+1}) \wedge \ldots \wedge \forall k (x_{c-1} \xrightarrow{k} x_c) \wedge \forall k (x_c \xrightarrow{k} x_{c+1}) \wedge \ldots \wedge \forall k (x_{b-1} \xrightarrow{k} x_b)$$

*Dans ce cas, en appliquant la propriété (3) à la formule précédente (6) sur les valeurs intermédiaires à* $x_a$, $x_b$ *et* $x_c$ *:*

$$\forall k (x_a \xrightarrow{k} x_c) \wedge \forall k (x_c \xrightarrow{k} x_b)$$

*Or* $x_a = x_b$, *nous obtenons donc une contradiction (propriété 2) suivante :*

$$\forall k (x_a \xrightarrow{k} x_c) \wedge \forall k (x_c \xrightarrow{k} x_a) \tag{7}$$

*Nous pouvons donc dire par l'absurde que la contraposé du théorème (formule 5) est fausse.*

Le théorème 1 étant vérifié, l'algorithme s'arrêtera et appliquera les retards nécessaires à la synchronisation collective. Donc une décision collective sera prise par tous.

### 3.4. Complexité de l'algorithme et proposition architecturale

En nombre de messages, la complexité est de $\theta(x)$ pour une transmission en multicast (ligne 4), en effet le nombre de messages envoyés sera toujours de $n_c$. Si la transmission n'est pas multicast, la complexité augmentera à $\theta(x^2)$ et le nombre de messages à $n_c * n_c$.

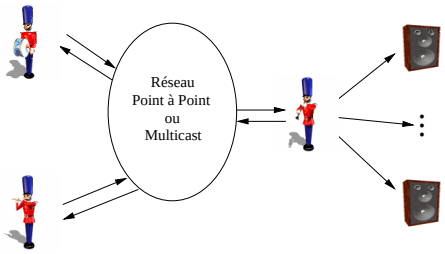



FIG. 4 – *Une architecture pour un système à consommateurs passifs.* $n_p = n_c = 3$

Dans l'algorithme de synchronisation, en temps de calcul, la boucle $(b)$ de la ligne 10 à 16 est en $\theta(n_p * n_c)$. Étant donné que l'algorithme se termine, la boucle $(a)$ de la ligne 8 à 23 s'exécutera au maximum $n_p$ fois. La boucle $(b)$ a donc une complexité de $\theta((n_p)^2 * n_c)$. La boucle $(c)$ de la ligne 19 à 22 ne s'exécutera qu'au dernier tour de la boucle $(a)$, la variable $sync$ étant mise à $vrai$ ligne 18. Sa complexité est de $n_p$. La complexité totale de l'algorithme de traitement des statistiques est donc en $\theta((n_p)^2 * n_c + n_p)$ soit en $\theta((n_p)^2 * n_c)$. Le seul calcul de la procédure d'envoi des statistiques est celui du tableau $tab$ de taille $n_p$. La complexité de cette procédure est donc en $\theta(n_p)$.

L'objectif de l'algorithme d'auto synchronisation distribuée de flux audio est de fournir une vision cohérente aux consommateurs d'un système de production. De plus, il minimise les temps de latence entre la production et la consommation. Cette propriété est particulièrement importante pour les sites hébergeant à la fois un producteur et un consommateur, ceux-ci pouvant être au sein de la même application. Cela exclut donc un algorithme "centralisé" où un récepteur $P_c$ imposerait aux autres un ensemble d'événements $(ev^1_{k1} \cdots ev^{n_p}_{k_{n_p}})$ comme base de consommation simultanée. En effet, chaque consommateur "esclave" devra ajouter la latence nécessaire à la bonne réception d'un flux sur tous les autres flux, ce qu'il est préférable de minimiser dans le cas de sites hébergeant des producteurs et des consommateurs. L'algorithme d'auto synchronisation distribuée de flux audio ajuste chaque flux au mieux pour le système. Cela ouvre une proposition architecturale (figure 4) pour un tel système : seuls les consommateurs hébergés par un site producteur participeront à l'algorithme de synchronisation. La vision du système par les participants à l'algorithme sera ensuite imposée aux sites uniquement consommateurs. En effet, seuls les sites hébergeant à la fois une production et une consommation d'événements sont capables de percevoir la latence, il n'est donc pas nécessaire de prendre en compte les délais des autres. Toutefois, nous n'excluons pas qu'il puisse exister au sein du système des consommateurs non producteurs qui désirent aussi cette latence minimale, surtout s'ils produisent un autre type d'événements différents mais aussi liés au système global. Par exemple dans l'orchestre virtuel réparti, un chef d'orchestre qui dirigerait avec de la vidéo mais sans émettre de son. Nous distinguons donc toujours $n_p$ et $n_c$.
Sur la figure 4, la synchronisation se fera uniquement entre les musiciens et c'est le joueur de hautbois qui diffusera la musique collective aux auditeurs. Si les auditeurs sont considérés comme passifs dans le système (ce qui n'est pas forcément le cas), alors la diffusion peut se faire avec une latence plus forte que celle perçue entre les musiciens.

Cette nouvelle architecture permet aussi de réduire la complexité en calculs et en nombre de messages de l'algorithme de traitement des statistiques. En effet, dans une situation de diffusion importante des flux, l'algorithme de synchronisation ne passerait pas à l'échelle d'un nombre de consommateurs ($n_c$) trop grand.

### 3.5. Un exemple d'exécution de l'algorithme

Pour le test, $n_p = 3$ et $n_c = 3$. Trois machines participent au test : chacune d'elles envoie le son sur une adresse multicast et reçoit les trois flux du système. La figure 5 montre un des patchs jMax[6] utilisés durant le test. Les patchs utilisés sur chaque machine sont identiques. Ici, le patch "gamme mineur Harmonique" prend en entrée une impulsion (objet "metro"), un entier tiré au hasard et la hauteur de la gamme. L'ascenseur permet de régler le volume du signal audio.

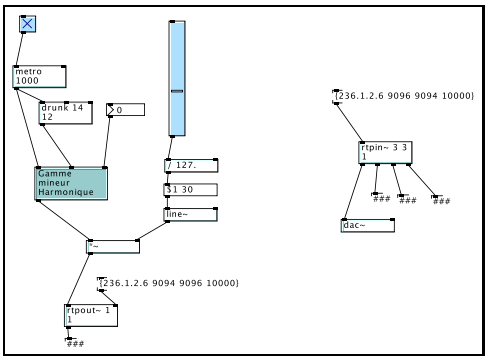



FIG. 5 – *Version multicast : le patch de la première machine*

Remarquons que le patch présenté va recevoir (objet rtpin) le flux qu'il produit (objet rtpout). En effet, il est nécessaire que l'audition de ce qui est joué localement participe à la synchronisation générale. Dans le cas contraire, chaque site aura une audition différente du jeu de groupe.
Dans cet exemple, les musiciens ne vont pas essayer de s'adapter au retard introduit par le réseau puisque les humains sont simulés par un signal audio généré et donc incapables d'écouter les autres. L'objectif ici est uniquement de dérouler l'algorithme d'auto synchronisation distribuée de flux audio.

Sachant que les machines **ppc1** (noté $pc_1$ et $pp_1$), **mais** (noté $pc_2$ et $pp_2$), **oceanonix** (noté $pc_3$ et $pp_3$) sont les machines réceptrices et sources des flux, voici le tableau dont dispose chacun des processeurs $pc_i (1 < i < n_c)$ après réception de toutes les statistiques :
Précisons que Ox2d0b61d0, 0x38e53b22, 0x7bc14b05 sont les identifiants de source du protocole rtp respectivement de **ppc1**, **mais** et **oceanonix**.

| | $pc_1$ | $pc_2$ | $pc_3$ |
|---|---|---|---|
| $pp_1$ | e9a00 | eb2c0 | e9d80 |
| $pp_2$ | 118000 | 1199c0 | 1183c0 |
| $pp_3$ | 6a880 | 6c280 | 6ac00 |

Chaque colonne contient l'instantané pris par le processeur consommateur indiqué. Notons que les estampilles (dates) mesurées sont en base hexadécimale. Regardons maintenant comment se déroule l'algorithme de traitement des estampilles.
$pp_3$ a le plus grand indice, c'est donc ce processeur qui sera choisi comme référence pour les calculs.
En regardant les différences avec le processeur $pp_2$ pour chaque récepteur $pc_i$, $pp_3$ est toujours en retard par rapport à $pp_2$. Notre référence n'est donc pas bonne.
Précisons que l'on ne dispose pas forcément d'assez de temps dans les tampons audio pour enlever des échantillons, nous préférons chercher une référence qui aura de l'avance par rapport aux autres.
Nous allons donc changer de référence, la plus grande n'ayant pas encore été choisie (ligne 9 de l'algorithme) : $pp_2 = pp_{ref}$
Les différences sont calculées comme suit :

[6] jMax est un langage de programmation graphique, un patch jMax est une fonction graphique

Dans la case $pp_i$ x $pc_j$ la différence est $hlc_{pc_j}(ev^{ref}) - hlc_{pc_j}(ev^i)$. La colonne "Différence max" contient le tableau $dif[pp_i]$.

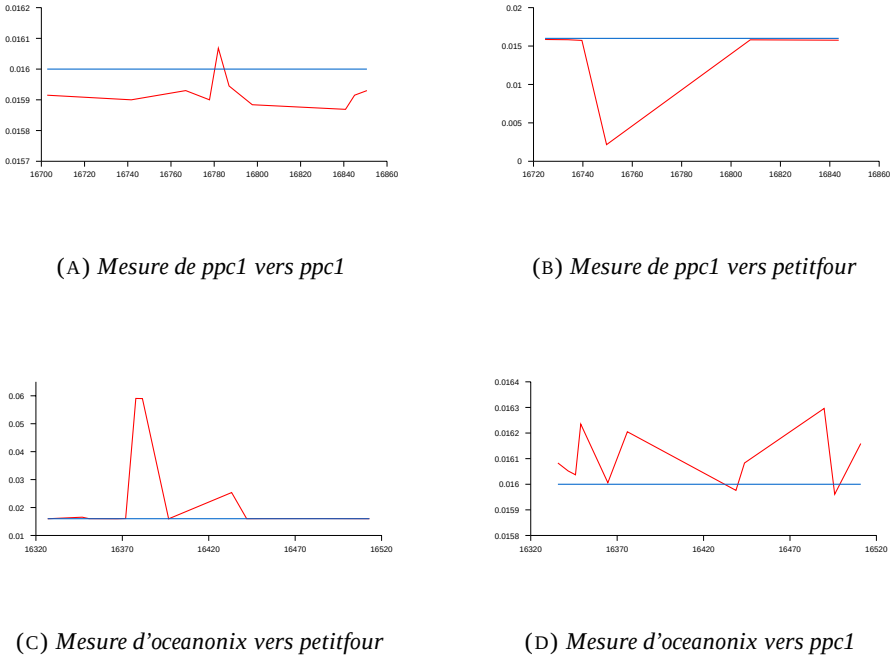
| | $pc_1$ | $pc_2$ | $pc_3$ | Différence max |
|---|---|---|---|---|
| $pp_1$ | 2e600 | 2e640 | 2e700 | 2e700 |
| $pp_2$ | 0 | 0 | 0 | 0 |
| $pp_3$ | ad780 | ad7c0 | ad740 | ad7c0 |

Ces calculs ont été fais sur chaque processeur pc, voyons maintenant comment chacun d'eux va adapter sa propre vision des flux afin d'avoir tous la même synchronisation à l'estampille.
Pour chaque flux, on compare la différence locale et celle maximale, ensuite on obtient le nombre d'échantillons à ajouter pour obtenir exactement l'écart maximal par rapport au flux référence. Rappelons que l'estampillage se fait à l'échantillon.
Sur $pc_1$ :
$dif[pp_1] - (estampille_{pc_1}[pp_{ref}] - estampille_{pc_1}[pp_1]) = 2e700 - 2e600 = 100$
$dif[pp_2] - (estampille_{pc_1}[pp_{ref}] - estampille_{pc_1}[pp_2]) = 0 - 0 = 0$
$dif[pp_3] - (estampille_{pc_1}[pp_{ref}] - estampille_{pc_1}[pp_3]) = ad7cO - ad780 = 40$
Soit 256 échantillons (en base décimale) à ajouter au flux venant de $pp_1$, aucun pour $pp_2$ et 64 pour $pp_3$. En faisant de même, $pc_3$ va retarder le flux de $pp_3$ de 128 échantillons et $pc_2$ le flux issu de $pp_1$ de 192 échantillons. L'algorithme se termine là et chaque récepteur a la même vision des retards entre les flux audio.

## 4. Implémentation, tests et mesures

L'algorithme d'auto synchronisation distribuée de flux audio a été déployé dans le prototype de l'orchestre virtuel réparti. Le nombre d'émetteurs ($n_p$) de musique ainsi que le nombre de récepteurs ($n_c$) à synchroniser peuvent être aussi grands que peuvent le supporter les machines hôtes et le ou les réseau(x). Chaque $pc_1 \cdots pc_{n_c}$ est un objet rtpin, développé comme un objet du logiciel jMax [6, 5] (pour exemple, voir la figure 5). De même, Chaque $pp_1 \cdots pp_{n_p}$ est implémenté dans un objet rtpout. Les objets rtpout produisent des échantillons sonores à une cadence de 44100Hz, qui seront consommés par les objets rtpin à cette même cadence. La communication entre producteurs et consommateurs se fait via le réseau à l'aide du protocole RTP [16]. Nous avons utilisé la librairie RTP appelée UCL Common Code Library version 1.2.8 développée par le département informatique de l'université "university college London". Les objets sont écrits en langage C et représentent 2354 lignes pour 5766 mots. Chaque objet rtpout numérote les échantillons qu'il produit dans le champ *timestamp* de RTP. Ce champ est incrémenté à chaque échantillon et correspond donc à l'horloge logique de production hlp. Chaque échantillon correspond à un événement. L'utilisation du protocole RTP permet l'identification des processeurs $pc_1 \cdots pc_{n_c}$ et $pp_1 \cdots pp_{n_p}$ grâce au champ *ssrc* de RTP. Chaque objet rtpin consomme simultanément un échantillon issu de chaque objet rtpout à la même cadence que celle de production, i.e. 44100Hz. Pour le système de l'orchestre virtuel, les musiciens et auditeurs devront avoir une perception identique du mixage de sources sonores (la consommation des événements). Le test se faisant sur réseau local 100Mb/s pour trois flux audio à 44100Hz 16bits mono, nous avons considéré les liens sans pertes. Les valeurs de synchronisation mesurées sont celles de l'exemple d'exécution de l'algorithme du paragraphe 3.5.

Afin de mettre en valeur les conditions dans lesquelles s'exécutent l'algorithme de synchronisation des flux, nous avons choisi de montrer les mesures faites sur les latences estimées grâce au protocole RTCP (protocole de contrôle associé à RTP). Nous avons donc testé la configuration suivante : deux musiciens à distance sur deux postes avec une troisième machine envoyant du son calculé, tous étaient placés sur le même réseau local 100Mb/s. Seuls deux objets rtpin permettaient l'audition de la scène, un pour chaque musicien. Les deux musiciens recevaient du son calculé par la machine oceanonix. Celle-ci ne fonctionnait qu'en émission. Le son calculé était une mélodie avec des notes prises au hasard dans la gamme de Do mineur Harmonique. Les deux musiciens devaient donc jouer avec la machine de façon synchronisée. Sur la figure 6, nous pouvons voir différentes mesures (en secondes) des délais aller-retour mesurés durant l'expérience. La ligne continue nous indique la valeur moyenne des délais mesurés sur




(A) *Mesure de ppc1 vers ppc1* (B) *Mesure de ppc1 vers petitfour*

(C) *Mesure d'oceanonix vers petitfour* (D) *Mesure d'oceanonix vers ppc1*

FIG. 6 – *Mesure des délais aller retour*

les machines (16ms). Quelques pics et creux apparaissent malgré les conditions favorables. Les quatre flux 44100Hz en 16bits correspondent à une charge de 2,7Mbs sans les entêtes et environ 2,73Mbs avec les entêtes (celles-ci représentant environ 1,3% du flux) représentent une faible charge face au réseau local 100Mb/s. Ces grandes variations peuvent être provoquées soit par la gigue réelle du réseau soit par l'ordonnancement non temps réel des systèmes d'exploitations (SuSE 7.1 et 8.0).
Ces mesures nous montrent qu'il est possible de jouer de la musique sur un réseau local comme si les musiciens étaient côte à côte. En effet, l'oreille humaine ne perçoit pas les décalages inférieurs à 20ms, donc supérieurs à la moitié des délais aller-retour mesurés (environ 16ms en moyenne). Les tests entre deux musiciens sur ce même réseau ont pu confirmer cette hypothèse. Cela laisse envisager l'utilisation d'un LAN dans les studios d'enregistrement, afin par exemple de faciliter les câblages.
Cependant, cela nous montre aussi que l'on ne peut pas vraiment vérifier de façon empirique les effets de l'algorithme de synchronisation sur un tel réseau. Les latences réseau étant inférieures à la contrainte des 20ms, appliquer les compensations de retard ne change rien au niveau de l'audition. C'est pourquoi les tests du prototype sur un réseau de type MAN nous donneront plus d'informations sur l'impact des délais ajoutés sur le jeu des musiciens. Dans l'état actuel des choses, nous avons pu vérifier en faisant jouer deux musiciens ensemble avec le prototype que les délais ne sont pas perceptibles.

## 5. Conclusion et travaux futurs

Le travail en cours vise le passage à l'échelle aussi bien en terme d'étendue du réseau que du nombre de musiciens (nous visons 10 musiciens pour les prochains tests). Le travail d'expérimentation est prévu pour être complété par une étude de dimensionnement car la garantie de latence constante et minimale est de nature fondamentalement statistique. Il faudra à l'avenir choisir une stratégie de gestion des pertes parmi les mécanismes présentés dans [3] et identifier une solution pour le problème de dérive des horloges des cartes sons [14, 8]. Une étude sur le portage de jMax sur un système d'exploitation à ordonnancement temps réel est actuellement en cours (jMax ordonnance lui-même l'exécution de ses objets).

Nous avons présenté ici un algorithme permettant à des musiciens distants physiquement d'entendre la musique qu'ils jouent respectivement de façon synchronisée avec une latence constante et minimale. De façon générale, cet algorithme s'étend à tout système réparti temps réel avec plusieurs sources produisant des flux à la volée et avec des consommateurs multiples.
Les résultats obtenus nous permettent d'envisager l'installation de notre mécanisme dans des infrastructures de type studio d'enregistrement (pour simplifier le câblage par exemple) ou encore dans les écoles de musique afin d'automatiser l'apprentissage de l'interaction musicale entre instrumentistes ou d'automatiser l'apprentissage de gestes musicaux élémentaires.
Cependant, nous privilégions les tests sur un réseau de type MAN. Cela nous permettrait d'une part de dimensionner les mécanismes mis en jeux et d'autre part de déterminer plus précisément le type d'interaction que nous allons fournir aux musiciens distants.

**Bibliographie**

1. Robin Bargar, Steve Church, Akira Fukuda, James Grunke, Douglas Keislar, Bob Moses, Ben Novak, Bruce Pennycook, Zack Settel, John Strawn, Phil Wiser, and Wieslaw Woszczyk. AES white paper : Networking audio and music using internet2 and next-generation internet capabilities. Technical report, AES : Audio Engineering Society, 1998.
2. Erwan Becquet, Mazen Abdallah, Eric Gressier-Soudan, François Horn, and Laurent Bacon. Object oriented timed messaging service for industrial ethernet : a fieldbus like architecture for power plant control and factory automation. *In proceeding Fieldbus Technologie (FeT'2001)*, November 2001. IFAC. Nancy, France.
3. Nicolas Bouillot. Transport du son produit en temps réel sur les réseaux best-effort. Rapport bibliographique de DEA, 2002. DEA SIR, P6, CNAM, ENST.
4. Jeremy R. Cooperstock and Stephen P. Spackman. The recording studio that spanned a continent. In *IEEE International Conference on Web Delivering of Music,WEDELMUSIC*, Florence, Italie, 2001.
5. F. Déchelle. jmax : un environnement pour la réalisation d'applications musicales temps réel sous linux. *Actes des journées d'Informatique Musicale*, 2000.
6. F. Déchelle, R. Borghesi, M. De Cecco, E. Maggi, B. Rovan, and N. Schnell. jmax : an environment for real-time musical applications. *ComputerMusic Journal*, 23(3) :50–58, 1999.
7. Anton Eliëns, Martijn van Welie, Jacco van Ossenbruggen, and Bastiaan Schönhage. Jamming (on) the web. In *Proceedings of WWW6*, 1997.
8. D. Fober, Y. Orlarey, and S. Letz. Clock skew compensation over a high latency network. In ICMA, editor, *Proceedings of the International Computer Music Conference*, pages 548–552, 2002.
9. Masataka Goto, Isao Hidaka, Hideaki Matsumoto, and Yosuke Kuroda Yoichi Muraoka. A jazz session system for interplay among all players. In *ICMC Proceedings*, 1996.
10. Masataka Goto and Ryo Neyama. Open RemoteGIG :an open-to-the-public distributed session system overcoming network latency. *IPSJ JOURNAL*, 43(2) :299–309, February 2002. (en japonais).
11. Masataka Goto, Ryo Neyama, and Yoichi Muraoka. RMCP : Remote music control protocol, design and applications. In *ICMC Proceedings*, 1997.
12. Utility Communications Specification Working Group. TASE.2 services and protocol. version 1996-08. iccp inter-control centre communication protocol version 6.1. Technical Report IEC 870-6-503, IEC, 1996.
13. Fabio Kon and Fernando Iazzetta. Internet music : Dream or (virtual) reality. In *Proceedings of the 5th Brazilian Symposium on Computer Music*, Belo Horizonte, Brazil, 1998.
14. Orion, Hodson, and Colin. Skew detection and compensation for internet audio applications, 2000.
15. Miller Puckette, Mark Danks, Rand Steiger, and Vibeke Sorensen. http : / / visualmusic.org / gvm.htm, 1999.
16. Schulzrinne, Casner, Frederick, and Jacobson. RTP : A transport protocol for real-time applications. *RFC 1889*, 1998.
17. Aoxiang Xu and Jeremy Cooperstock. Real-time streaming of multichannel audio data over internet. In *AES 108th convension*, Paris, 2000.
18. J.P. Young and I. Fujinaga. Piano master classes via the internet. In *Proceedings of the International Computer Music Conference*, 1999.