What's new in JACK2

Stephane Letz, Nedko Arnaudov, Romain Moret

Linux Audio Conference : april 16, 2009

GRAME CENTRE NATIONAL DE CREATION MUSICALE

Post LAC 2008 state

Desirable goals from LAC 2008 ”The Future of JACK meeting”

- Startup & configuration : a bit of work
- NetJack : yes in progress
- Desktop integration : yes in progress
- Internal design : yes in progress
- Client programming and API : yes in progress

Not anticiped

- External contacts and developments
- Solaris version with new ”profiling” tools

Post LAC 2008 state

Desirable goals from LAC 2008 ”The Future of JACK meeting”

- Startup & configuration : a bit of work
- NetJack : yes in progress
- Desktop integration : yes in progress
- Internal design : yes in progress
- Client programming and API : yes in progress

Not anticiped

- External contacts and developments
- Solaris version with new ”profiling” tools

What has been done?

2008 Developments

- Internal design : code restructuration (Grame)
- Server control API : (Nedko Arnaudov, Grame)
- D-Bus based server control (Nedko Arnaudov, Juuso Alasuutari, Grame)
- NetJack2 (Romain Moret, Grame)
- ”Profiling” tools (Grame)
- Solaris version (Grame)

Other

- External contacts : RTL, CopperLan, Native Instruments
- OSX and Windows specific tools
- Future developments (MIDI, ”pipelining”, control...)?

What has been done?

2008 Developments

- Internal design : code restructuration (Grame)
- Server control API : (Nedko Arnaudov, Grame)
- D-Bus based server control (Nedko Arnaudov, Juuso Alasuutari, Grame)
- NetJack2 (Romain Moret, Grame)
- ”Profiling” tools (Grame)
- Solaris version (Grame)

Other

- External contacts : RTL, CopperLan, Native Instruments
- OSX and Windows specific tools
- Future developments (MIDI, ”pipelining”, control...)?

Internal design

Goals

- Cleanly separate server and client side services
- Define a server control API
- Improve server internal clients (example of use in NetJack2)

Server and client side libraries

Server side

- libjackserver : JACK API (opens client as ”internal” in the server...) + control API
- used by backends and internal clients
- used by ”jackd” and new ”jackdbus” control applications
- allows an application to embed the server in it’s process

Client side

- libjack : JACK API (opens client in separated processes using IPC...)
- used by client applications

Server and client side libraries

Server side

- libjackserver : JACK API (opens client as ”internal” in the server...) + control API
- used by backends and internal clients
- used by ”jackd” and new ”jackdbus” control applications
- allows an application to embed the server in it’s process

Client side

- libjack : JACK API (opens client in separated processes using IPC...)
- used by client applications

Control API

Objectives

- dynamically retrieve server control parameters, backend and internal clients parameters, get/set their values
- create/destroy the server
- start/stop the server with a given backend
- load/unload internal clients
- now used by ”jackd”

Internal clients

How are they controled and loaded?

- using the ”old” way (with jack_load -i ”parameters” and jack_unload)
- using the new control API way : jackctl_server_load_internal and jackctl_server_unload_internal

D-Bus description

What is D-Bus?

- a simple inter-process communication (IPC) system
- programs register for offering services to others
- clients look up which services are available

D-Bus server control access

D-Bus server control access

- jackdbus executable to start D-Bus service
- behaves as an interface between D-Bus system and the JACK server, to use JACK control API with D-Bus
- server autostart done by libjack using control D-Bus interface

D-Bus server control access

Exported interfaces

- jackdbus controller object export several interfaces
- configure the server (parameters access)
- control the server (start/stop)
- ”patchbay” : improved graph state access (connections, notification of changes...)
- jack_control python control tool
- better presented in Juuso presentation on LASH

NetJack2

Design

- simplify the usability model
- redesign for easier multi-platform support
- currently developed for LAN only

Components

- keep the master/slaves model
- netmanager (master) and jack_net bakend (slave)
- ”adapters” : audioadapter and netadapter

NetJack2

Design

- simplify the usability model
- redesign for easier multi-platform support
- currently developed for LAN only

Components

- keep the master/slaves model
- netmanager (master) and jack_net bakend (slave)
- ”adapters” : audioadapter and netadapter

Netmanager

Life cycle

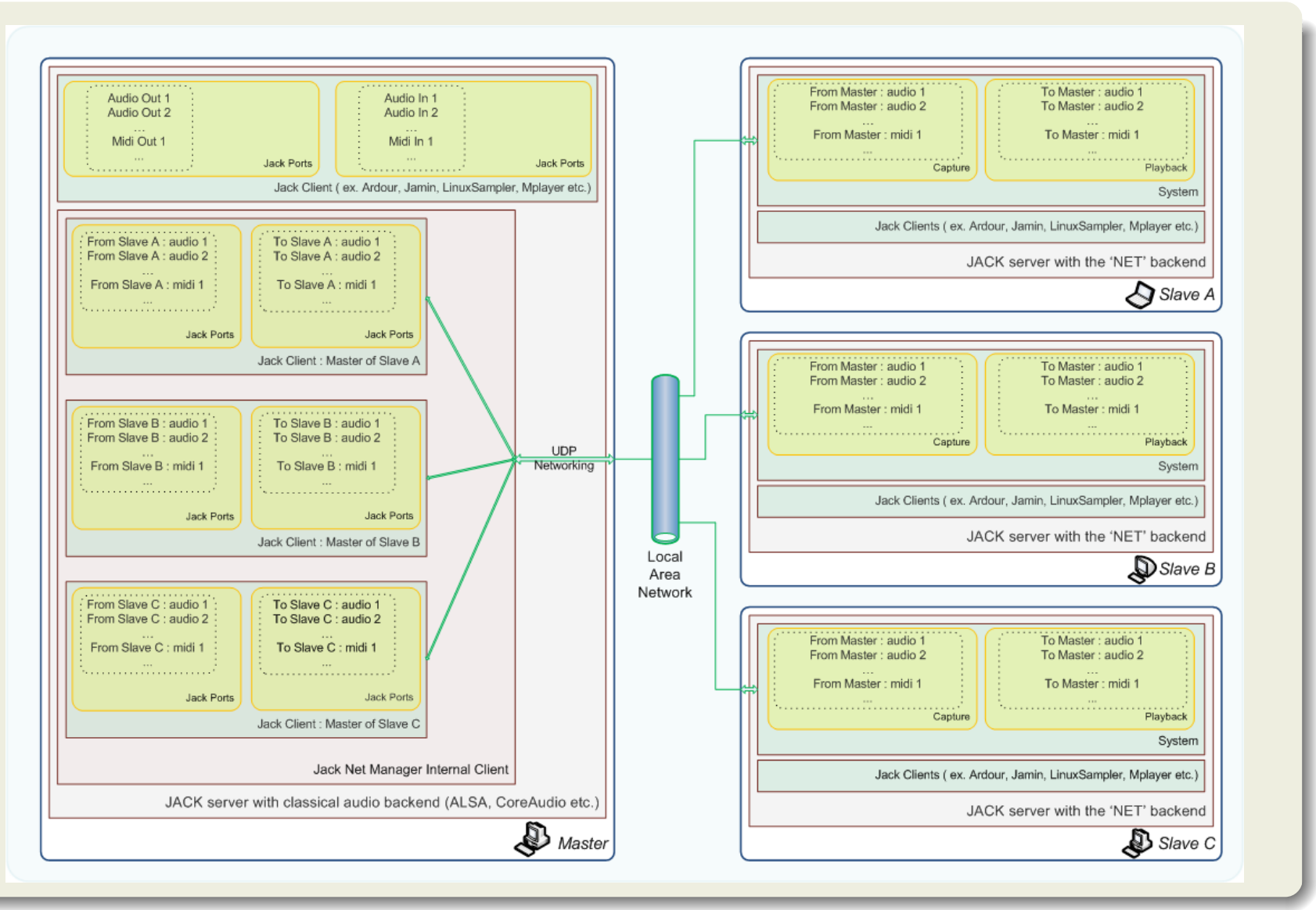
- loaded as an internal client
- waiting for slaves on a multi-cast address
- create ”proxy” internal clients for each slave, with requested number of in/out audio and MIDI ports
- ”proxy” ”automatically” appear and disappear when slaves come and go

Net backend

Life cycle

- appears as ”available”, server is running in ”dummy” mode (clients can connect...)
- connecting to master, describes it’s parameters : in/out ports...
- retrieve sample rate and buffer size from the master
- starts processing
- if master disappear, go back in ”dummy” mode

Example of setup



Data streams

Data stream

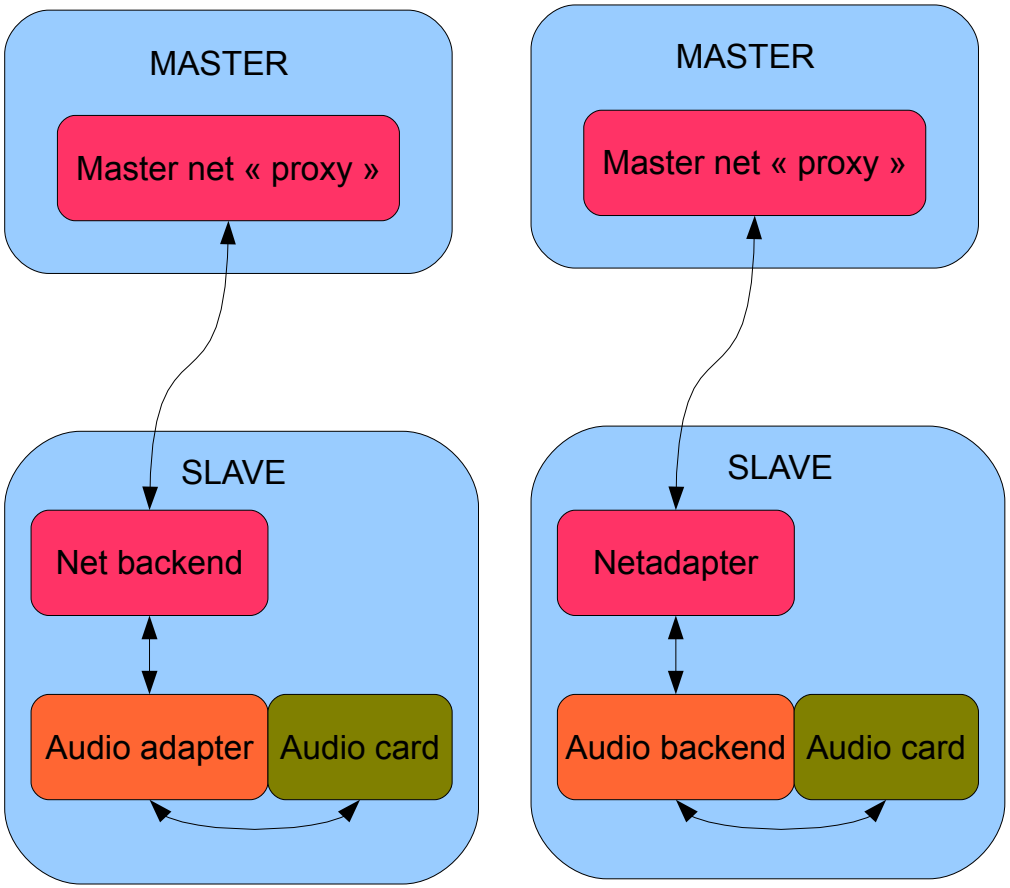
- audio and MIDI
- latency control (using ”fast”, ”normal”, ”slow” transmission modes...)
- number of in/out audio/MIDI ports, address, UDP port, MTU parameters

Adapters

Adapting network to audio interface

- aims at ”adapting” (sample rate, buffer size, clock drift...) a network stream on an audio interface
- audioadapter : adapts a slave synched on Net backend on it’s audio card (generalisation of alsa_in/out)
- a version on each plafform (Linux/ALSA, OSX/CoreAudio, Windows/PortAudio, Solaris/OSS)
- netadapter : adapts the network stream on a slave running an audio backend

Adapters



Audioadapter

Netadapter

Adapters

How does they work?

- ”produces/consumer” model using an intermediate ring buffer
- resampling if needed (using libsamplerate)
- resampling ratio dynamically adjusted using Torben Hohn PI controler (JACK1)

”Profiling” tools

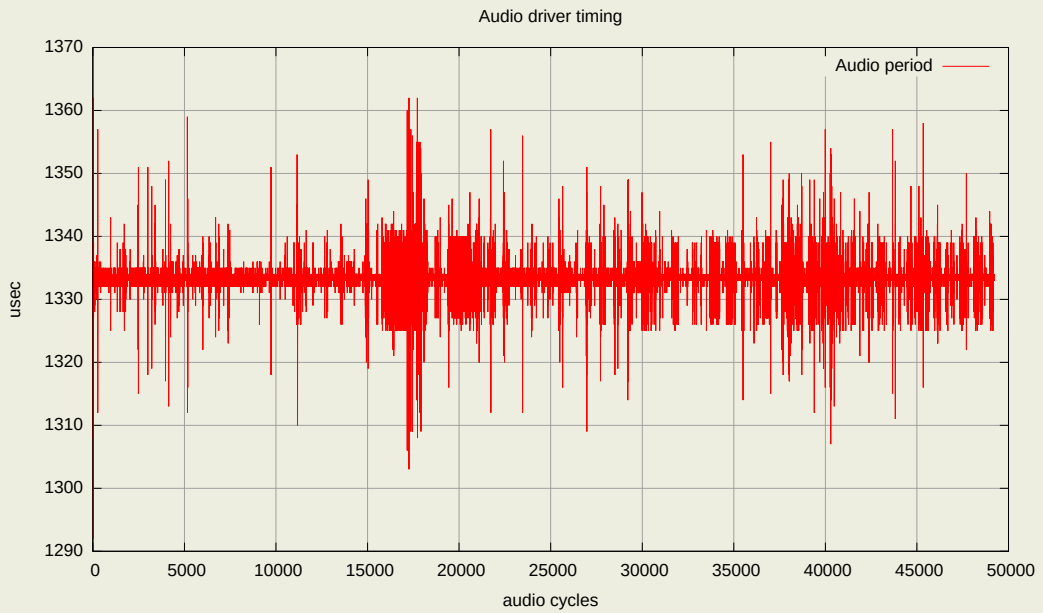
Objectives

- record timing informations when graph is running
- timing of backend, signal, wake-up, end date of each client
- compute scheduling latency and client duration
- generate log files and visualization scripts for GnuPlot

Audio driver interrupt

Duration between successives audio interrupts

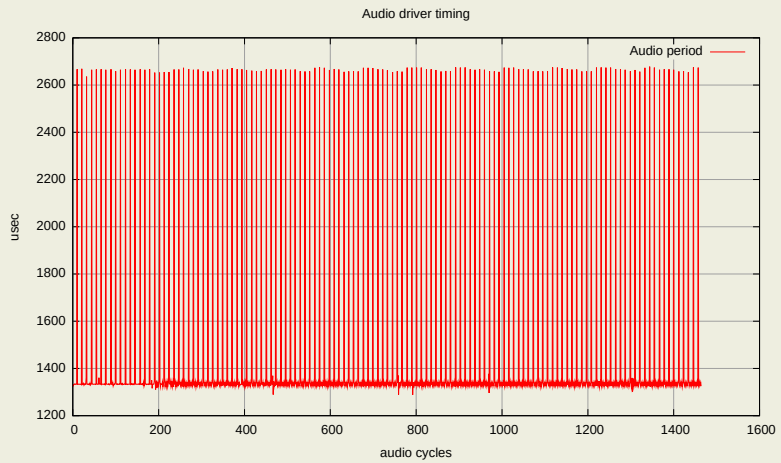
- 64 frames, 48 kHz, regular interrupt



Audio driver interrupt

Duration between successives audio interrupts

- 64 frames, 44.1 kHz, non regular interrupt



Driver end date

Duration between start and end of cycle at driver level

- 64 frames, 48 kHz, asynchronous mode

Driver end date

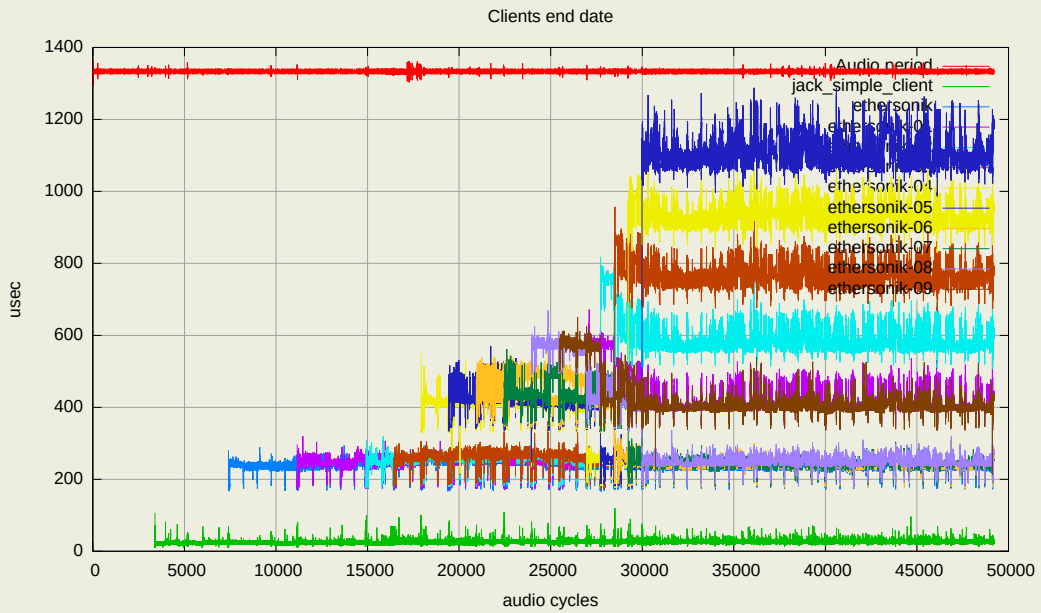
Duration between start and end of cycle at driver level

- 64 frames, 48 kHz, synchronous mode

Clients end date

End date of all clients

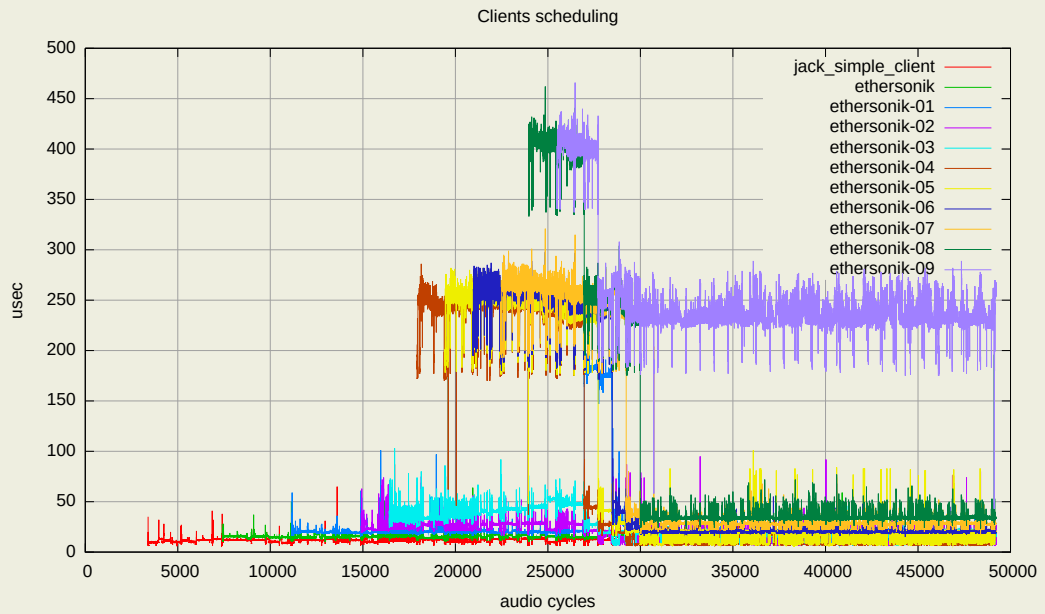
- here at 64 frames, 48 kHz



Clients scheduling latency

Duration between signal date and actual wake-up date

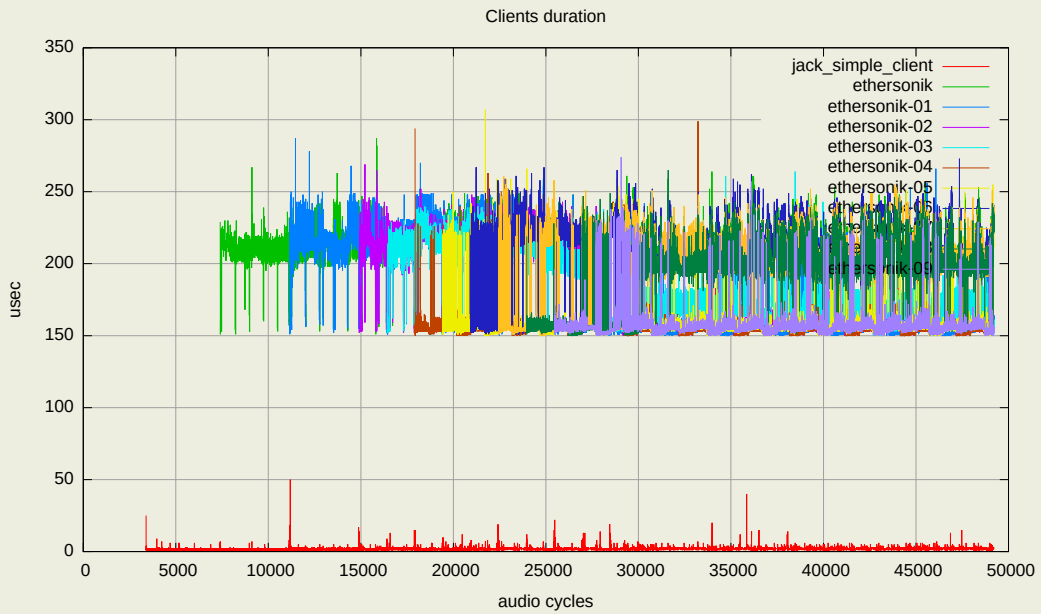
- here at 64 frames, 48 kHz



Clients duration

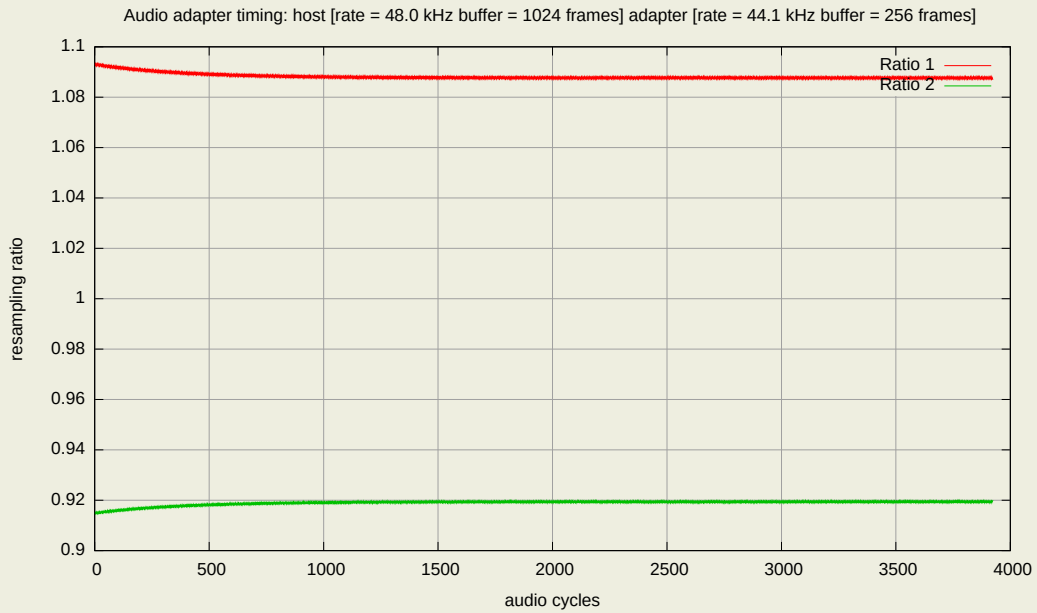
Duration between wake-up date and end date

- here at 64 frames, 48 kHz



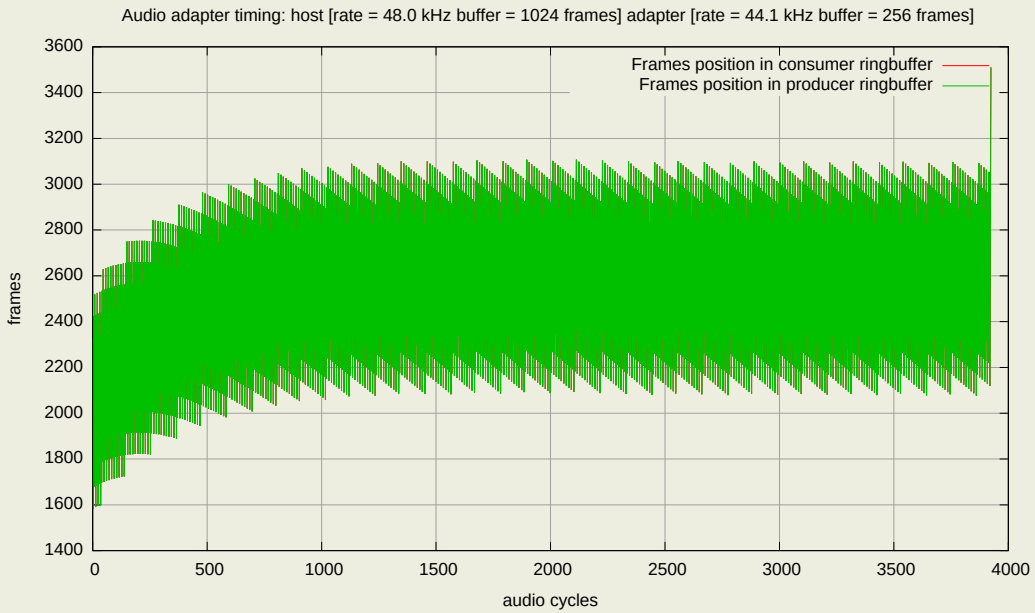
Adapters profiling : resample ratio

Server running with the ”dummy” driver and audio adapter



Adapters profiling : position in ringbuffer

Server running with the ”dummy” driver and audio adapter



Solaris version

Reviewer comment : why on earth would anybody seriously consider Solaris for pro-audio work?

- well, this version was funded by french radio RTL...
- OSS RME MADI 64 in/out driver developed by Hannu Savolainen
- actually not so bad (could be caracterized using the profiling tools) : below 80 usec max scheduling latency using CPU sets on a highly loaded 4 cores 2 Ghz Dell machine
- so good enough for RTL needs

External contacts

RTL french radio

- Developing their entirely ”digital” radio using JACK2 on Solaris with a RME MADI 64 in/out card
- Profiling tools to help caracterize real-time behaviour of the system

CopperLan

- A ”complete solution for networking all equipment in the domains of pro-audio and music” recently presented at Frankfurt MusikMesse
- Klavis Technologies implemented a JACK / CopperLan bridge prototype on OSX

Native Instruments

- Future line of NI product using JACK API natively on OSX and Windows

External contacts

RTL french radio

- Developing their entirely ”digital” radio using JACK2 on Solaris with a RME MADI 64 in/out card
- Profiling tools to help caracterize real-time behaviour of the system

CopperLan

- A ”complete solution for networking all equipment in the domains of pro-audio and music” recently presented at Frankfurt MusikMesse
- Klavis Technologies implemented a JACK / CopperLan bridge prototype on OSX

Native Instruments

- Future line of NI product using JACK API natively on OSX and Windows

External contacts

RTL french radio

- Developing their entirely ”digital” radio using JACK2 on Solaris with a RME MADI 64 in/out card
- Profiling tools to help caracterize real-time behaviour of the system

CopperLan

- A ”complete solution for networking all equipment in the domains of pro-audio and music” recently presented at Frankfurt MusikMesse
- Klavis Technologies implemented a JACK / CopperLan bridge prototype on OSX

Native Instruments

- Future line of NI product using JACK API natively on OSX and Windows

Specific developments on OSX and Windows

JackOSX package

- integrates JACK with CoreAudio (JackRouter CoreAudio device, JackAU audio unit, JackVST VST plugin...)
- based on Jackdmp/JACK2 starting early 2008 (version 0.75)
- 5 versions released in 2008
- 200 download/day, 1100 users on Yahoo Group

Windows

- integrates JACK with Windows ASIO (JackRouter ASIO device)
- some improvements done in QjackCtl
- clean installer (Romain Moret in 2008)

Specific developments on OSX and Windows

JackOSX package

- integrates JACK with CoreAudio (JackRouter CoreAudio device, JackAU audio unit, JackVST VST plugin...)
- based on Jackdmp/JACK2 starting early 2008 (version 0.75)
- 5 versions released in 2008
- 200 download/day, 1100 users on Yahoo Group

Windows

- integrates JACK with Windows ASIO (JackRouter ASIO device)
- some improvements done in QjackCtl
- clean installer (Romain Moret in 2008)

Contributions (1)

A lot in 2008

- Recent code from JACK1 : Paul Davis, Florian Faber, Torben Hohn...
- Nedko Arnaudov, Juuso Alasuutari for D-Bus and waf scripts
- Romain Moret for NetJack2
- Tim Blechmann : code cleanup/optimization, SSE code...
- Marc-Olivier Barre for D-Bus and ”now dead” scons scripts...

Contributions (2)

Michael Voigt : JACK2 on L4/DROPS research project

- L4 : micro-kernel design and DROPS: Dresden Real-Time Operating System Project
- code source restructuration for easier later port
- timing benchmark : less than 10 usec scheduling latency (+/- 1 usec) (RT Linux is 40 usec +/- 20 usec on same machine)
- future : has to be used with L4/Linux to run JACK applications for Linux on DROPS

The future : what is still needed for 2.0 version?

MIDI bridge on all supported platforms (in progress)

- JACK MIDI bridge with native API on each plaform (CoreMIDI on OSX, WinMME on Windows, ALSA MIDI (seq/raw))
- proposal to use the already existing Master / slave model in backend
- allows to activate the MIDI backend independently from the audio backend

NetJack

- both versions should be merged

Missing port latency API?

- Paul and Fons proposal?

The future : what is still needed for 2.0 version?

MIDI bridge on all supported platforms (in progress)

- JACK MIDI bridge with native API on each plaform (CoreMIDI on OSX, WinMME on Windows, ALSA MIDI (seq/raw))
- proposal to use the already existing Master / slave model in backend
- allows to activate the MIDI backend independently from the audio backend

NetJack

- both versions should be merged

Missing port latency API?

- Paul and Fons proposal?

The future : what is still needed for 2.0 version?

MIDI bridge on all supported platforms (in progress)

- JACK MIDI bridge with native API on each plaform (CoreMIDI on OSX, WinMME on Windows, ALSA MIDI (seq/raw))
- proposal to use the already existing Master / slave model in backend
- allows to activate the MIDI backend independently from the audio backend

NetJack

- both versions should be merged

Missing port latency API?

- Paul and Fons proposal?

The future : what is still needed for 2.0 version? (2)

Audio device reservation scheme

- smoother collaboration with PulseAudio
- use Lennart Poettering proposed D-Bus based reservation API
- allows JACK ALSA backend to take precedence over PulseAudio (unconditionnal reservation)
- available and to be tested since JACK 1.9.2 version

The future : various ideas

Multi-backend model

- Master / slave model for audio backend...

More control frontend

- OSC (network control: server start/stop, connection state...)

”libjacknet” idea

- publish the NetJack API (master, slave, adapter) in a separated ”libjacknet” library
- allows audio components to access NetJack without the need of a JACK server
- successfully tested on iPhone simulator (master or slave)

The future : various ideas

Multi-backend model

- Master / slave model for audio backend...

More control frontend

- OSC (network control: server start/stop, connection state...)

”libjacknet” idea

- publish the NetJack API (master, slave, adapter) in a separated ”libjacknet” library
- allows audio components to access NetJack without the need of a JACK server
- successfully tested on iPhone simulator (master or slave)

The future : various ideas

Multi-backend model

- Master / slave model for audio backend...

More control frontend

- OSC (network control: server start/stop, connection state...)

”libjacknet” idea

- publish the NetJack API (master, slave, adapter) in a separated ”libjacknet” library
- allows audio components to access NetJack without the need of a JACK server
- successfully tested on iPhone simulator (master or slave)

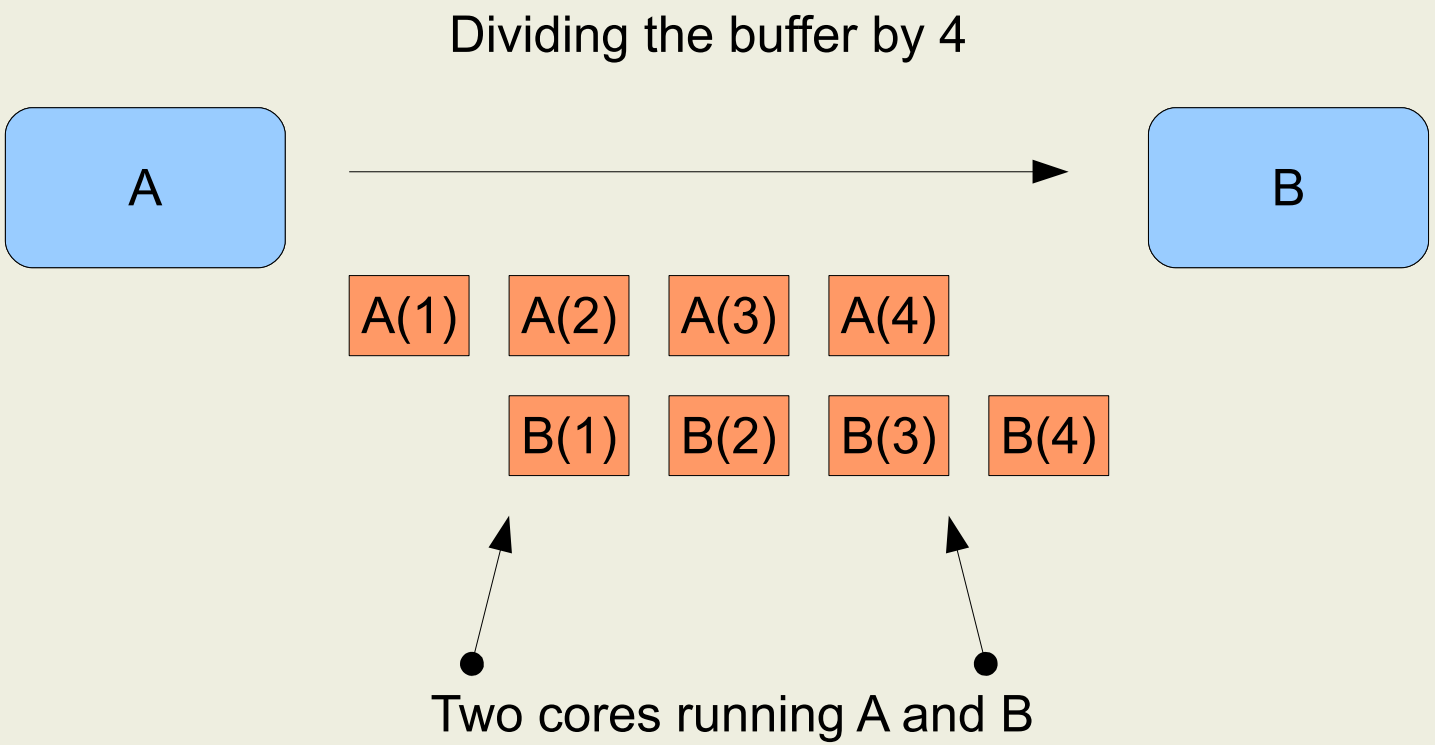
The future : pipelining version (1)

What is it for?

- allows to better use multi-cores machines with ”sequential” graphs
- general principle : executes the graph with a buffer size of D/N (D : driver buffer size, N : divisor)

The future : pipelining version (2)

One example



The future : pipelining version (3)

- divisor can be dynamically changed, and causes a ”buffer-size change” notification
- client can chose *not* to be pipelined : so hybrid graphs can be run
- available on a separated ”pipelining” branch on SVN
- testing in real word situations welcome! (Ardour 2.8 is ready for that...)