PulseAudio on Mac OS X

Daniel Mack
zonque@gmail.com

Abstract

`PulseAudio` is becoming the standard audio environment on many Linux dektops nowadays. As it offers network transparency as well as other interesting features OS X won't offer its users natively, it's time to have a closer look on possibilities on how to port this piece of software over to yet another platform.

In the recent months, I put some effort into attempts to port PulseAudio to Mac OS X, aiming for cross-platform interoperability between heterogen audio host networks and enabling other features PulseAudio offers such as uPnP/AV streaming.

This paper will first give a short overview about how CoreAudio, the native audio system on Mac OS X, is structured, the various ways that can be used to plug into this system, and then focus on the steps it takes to port PulseAudio to Mac OS X.

Keywords

PulseAudio, Mac OS X, network audio, virtual audio driver, portability

1 CoreAudio essentials

1.1 IOAudio Kernel extensions

- The Darwin kernel is in charge of handling hardware drivers, abstracted via the IOKit API framework.
- The kernel's representation of an audio device is an object derived from the IOAudioDevice base class, which holds a reference of an IOAudioEngine (or a derived type thereof).
- The kernel's way of adding audio streams to a device is attaching objects of type IOAudioStream to an IOAudioEngine.
- The kernel's API is only one way to provide an audio device to the system; the other is a plugin for the HAL (see above).
- Sample material is organized in ring buffers which are usually shared with the hardware.
- IOAudioEngines are required to report their sample rate by delivering exact timestamps whenever their internal ring buffer rolls over. The more precise, the better, as its userspace counterpart (the `HAL`, see below) can do better estimation of the device's speed and approximate closer to the actual hardware sample pointer positions, resulting in smaller latencies.

1.2 HAL

The `HAL` is part of the `CoreAudio` framework and is automatically instanciated within the process image of each `CoreAudio` client application. During its startup, it scans for plugins in `/Library/Audio/Plugins/HAL` and this way offers the possibility of loading userspace implementations of audio drivers. The `HAL` is also in charge of interfacing to the `IOAudio` based kernel drivers and hence acts as their bridge to userspace clients.

1.3 AudioHardwarePlugins for HAL

Automatically loaded by the HAL code upon creation of an audio client, AudioHardwarePlugins are instanciated via the standard `CFBundle` load mechanisms. An interface must be implemented to provide the hooks needed by the `HAL`, and a full-fledged infrastructure of APIs for adding audio devices, streams and controls are available. Unlike kernel drivers, virtual drivers implemented as `HAL` plugin are working on a per-client base, so their implementations must care for mixing and inter-client operability themselves.

1.4 System Sound Server

This daemon is in charge for handling system-internal sound requests such as interface and alert sounds.

1.5 coreaudiod

`coreaudiod` is a system-wide daemon that gives home to the `System Sound Server` and

provides the `AudioHardwareServices` API for querying parameters of available audio drivers. The daemon also handles the default sound interface configuration on a per-user level[1].

1.6 AudioUnits

`AudioUnits` are Mac OS X typical `CFBundles` which can be installed user-wide or system-wide to fixed locations in the file system and which can be accessed by arbitrary applications with an standarized API for audio processing. They can also offer a graphical representation for parameter control and visualization. The two supported types of `AudioUnit` plugins are effect processors and virtual instruments.

2 Possible audio hooks

The purpose of this project is to be able to hook into the transport channels of all audio applications - including system sounds, if desired - and re-route audio through an either local or remote PulseAudio server connection.

Mac OS X officially offers a number of ways to access the audio material:

- A virtual sound card interface implemented as kernel driver which can either be configured as standard sound interface for all appliactions and/or system sounds. Applications may let the user decide which sound card to use for input and output sound rendering, but for those which don't (like `iTunes`, `QuicktimePlayer`, `iChat`, ...), setting the system-wide default is the only option.
- A virtual sound card interface implemented as AudioHardwarePlugin for the HAL. The same rules as for the kernel versions apply: if an application doesn't allow its user to choose the device for audio output, the system falls back to the configured default.
- An `AudioUnit` which is loaded by more advanced applications such as `Logic`. For application which don't use this plugin interface, this is no option.

Another possible way of interaction is unofficial, somewhat hackish and based on the idea of library pre-loading for selected applications. Binaries are relaunched with their CoreAudio libraries temporarily replaced by versions which re-route audio differentely. An example of this approach is the closed-source shareware utility `AudioHijack`[2]. More research is needed in order to find out whether this approach is also feasable for PulseAudio sound re-routing. At the time of writing, this option is not being investigated on.

3 PulseAudio on OS X

In order to bring PulseAudio to Mac OS X, some tweaks are needed to the core system, and some parts have to be re-developed from scratch.

3.1 pulseaudiod

Porting the daemon is of course the main part of the work as it is the heart of the whole system other pieces connect to. Since a couple of versions, `pulseaudiod`, along with a selection of its essential modules, builds fine on OS X. Some adoptions were neccessary to make this happen.

- `poll()` is broken since Mac OS X 10.3, disrespecting the timeout argument and returning immediately if no file descriptor has any pending event. This was circumvented by using the select() syscall, just like PulseAudio does for Windows.
- `recv()` with `MSG_PEEK` does in fact eat up data from the given file descriptor. The workaround was to use a different `ioctl()` for this purpose.
- OS X lacks a proper implementation of `POSIX` locks but implements its own thing as defined in `Multiprocessing.h`. A version which uses them internally for the PulseAudio daemon was needed.
- clock functions work differently than on Linux, so a specialized version for the clock wrapper functions in PulseAudio was also neccessary.
- Mac OS X offers a powerful API to give userland tasks high priority. This is essential for real-time applications just like PulseAudio, so an implementation using this API was added to the daemon.
- Some library PulseAudio uses are not suitable for OS X. Work on the build system was done to build some parts of the suite conditionally.

[1] http://lists.apple.com/archives/coreaudio-api/2007/Nov/msg00068.html

[2] `http://rogueamoeba.com/audiohijackpro/`

3.2 CoreAudio device detection module

In order to make use of audio input and output devices CoreAudio knows about, a new `pulseaudiod` module was written which uses the CoreAudio specific callback mechanisms to detect hotplugged devices. For each detected device, a new module instance of `module-coreaudio-device` is loaded, and unloaded on device removal, accordingly.

This module is part of the offcial PulseAudio sources since some months and is called `module-coreaudio-detect`.

3.3 CoreAudio source/sink module

Loaded and unloaded by the house-keeping module `module-coreaudio-detect`, this module accesses the actual CoreAudio device, queries its properties and acts as translation layer between CoreAudio and PulseAudio. An important implementation detail is that code in this module has to cope with the fact that audio is exchanged between different threads.

This module is part of the offcial PulseAudio sources since many months and is called `module-coreaudio-device`.

3.4 Bonjour/ZeroConf service discovery module

Porting the dependency chain for Avahi (dbus, ...) wasn't an easy and straight-forward task to do, and given the fact that Mac OS X features a convenient API for the same task, a new module for mDNS service notification was written. The code for this module purely uses Apple's own API for announcing services to members of a local network.

This module is also part of the official PulseAudio source tree since a while and is called `module-bonjour-publish`.

3.5 Framework

On Mac OS X, libraries, headers and associated resources are bundled in framework bundles. As PulseAudio libraries and the libraries they are linked against are shared amongst several components for this project, they are all put in one single location (`/Library/Frameworks/pulse.framework`). This path was passed to the `configure` script as `--prefix=` directive when PulseAudio was built. A script (`fixupFramework.sh`) is in charge to resolve libraries dependencies which are not part of a standard Mac OS X installation. All libraries that are found to be dependencies for others are copied to the framework bundle and the tool `install_name_tool` which ships with `XCode` is called to remap the path locations recursively.

3.6 PulseConsole

PulseConsole is a Cocoa based GUI application written in Objective-C that aims to be a comfortable configuration tool for PulseAudio servers, both local and remote instances. It offers a way to inspect and possibly modify details and parameters and a nice GUI for per-stream mixer controls and routing settings.

The plan is to make this tool as convenient as possible, also with GUIs for mixer controls, detailed server inspection and all the like. This will need some time to finish, but is actively developed already.

3.7 AudioHardwarePlugin for HAL

CoreAudio allows to add software plugins to register virtual sound interfaces. Such a plugin was developed for PulseAudio, with the following key features.

- Allows audio routing to both the local and any remote server instances.
- Multiple plugin instances communicate with each other over a distributed notification center. This is essential for sharing stream volume information.
- Each plugin instance announces itself to a system-wide message bus and can receive setup controls. This way, an existing connection to a sound server can be changed to some other server instance.
- The plugin is capable of creating multiple virtual sound interfaces. This can be helpful to cope with more than the standard stereo channel mapping. The configuration of which interfaces are created is controlled by the Preference Pane implementation (see below).

3.8 PulseAudio AudioUnits

For a more fine-grained way of routing specific audio pathes through the PulseAudio daemon, AudioUnit plugins were developed. They connect to the local audio daemon and act as sound source and sound sink, respectively. All audio hosts that are capable of dealing with this type of plugin interface (ie, `Apple Logic`) can use this way of connecting specific sound pathes to PulseAudio.

3.9 Virtual audio driver (kext)

Another way of adding an audio device driver to a system is hooking up a kernel driver for a virtual device and communicating with this driver from user space to access the audio material. This is what the virtual audio driver does.

This part of the project mostly exists for historical reasons, before the `AudioHardwarePlugin` approach was followed, which turned out to be much more interesting and feasible for the purpose. The code is still left in the source tree for reference and as proof-of-concept which might act as reference in the future.

Some of its key features include:

- support for any number of interfaces, featuring a configurable number of input and output channels each.
- userspace interface to control creation and deletion of interfaces.
- usage of shared memory between userspace and kernel space, organized as ring buffer.
- infrastructure to register a stream to userspace for each client that is connected to the interface. The framework for this code exists, but all attempts to actually make it work failed so far.

The concept of the driver model is to have one abstract IOService object (instance of PADriver) which is the root node for all other objects. Upon creation (at load time of the driver), the PADriver will be announced to the userspace.

A IOUserClient class named PADriverUserClient can be instanciated by user space, and commands can be issued to create new and delete instances of PADevices. A PADevice is derived from IOAudioDevice and acts as a virtual audio device. To export audio functions, it has to have an PAEngine (derived from IOAudioEngine).

Depending on the type of audio engine (one for the mixed audio stream or one for each individual user client), the PAEngine can have one or many references to PAVirtualDevices, respectively.

Once a PAVirtualDevice is created, it is announced to the userspace, just like a PADriver. A userclient will create an object of type PAVirtualDeviceUserClient which can be used to issue commands specific to a PAVirtualDevice.

More information can be found in the repository at github.com.

3.10 virtual audio driver adapter module

Acting as counterpart of the virtual audio driver kernel module, a special purpose module for `pulseaudiod` takes notice of added and removed virtual sound card instances, maps the shared memory offered by the kernel and creates stream instances inside the PulseAudio daemon. The name for these streams are taken from the kernel space interface. As the kernel extension is not currently used anymore, this part of the source tree is also considered legacy.

3.11 Preference pane

The PulseAudio preference pane hooks itself into the standard Mac OS X system preferences and offers the following features:

- control the startup behaviour of the PulseAudio daemon
- configure authentication settings for network connections
- GUI for adding and deleting virtual sound interfaces

3.12 Component locations

Mac OS X organizes its file system contents in a quite different way than Linux installations. As described above, a framework is built in order to share the PulseAudio libraries amongst the various components. Components linking to the PulseAudio libraries have their linker settings configured to this path. Hence, the daemon and command line utilitily binaries as well as the loadable modules are found at the framework location as well, and if you want to access the PulseAudio command line tools (`pacmd`, `paplay`, ...) in the shell, the `$PATH` environment variable needs tweaking.

Apart from that, the other components are expected to be installed into specific locations so they can be found by the system. There will be documentation in the source tree to describe the exacte pathes.

3.13 Installer and packaging

A PackageMaker receipt has been created to generate installer packages that can be processed by the standard Mac OS X package installer, giving the user the general look and feel

and procedure as most OS X add-ons. Depending on Apple's policy for such tool suites, attempts might be made to publish the package via Apple's application store.

3.14 License and source code

All parts of this suite are licensed under the GNU General Public License in version 2 (GPLv2).

The source code is accessible in the public `git` repository found at `https://github.com/zonque/PulseAudioOSX`

4 Possible scenarios

Once the whole suite is developed as described and stable to a acceptable level, interesting audio routing scenarios are imaginable.

- Sound played back by `iTunes` can be routed through the virtual PulseAudio sound interface and from there be sent to an uPnP/AV audio sink.
- Sound played back by `iDVD` can be routed through the virtual PulseAudio sound interface and then be sent to an Airport Express using PulseAudio's ROAP module. Mac OS X can not natively do that.
- A `LADSPA` proxy plugin could be developed to communicate with PulseAudio directly on Linux hosts. The stream for this plugin could be re-routed to a network host running PulseAudio on Mac OS X, and there be used as virtual input stream in `Logic`, hence allowing virtual instruments and effect plugins on Mac OS X to be used in `LADSPA` environments.
- Without any network interaction, simply routing all audio through the virtual PulseAudio sound interface allows users to control volumes of all connected audio clients individually (eg, silence annoying flash player in your browser, leveling audio applications that don't offer a way to do this natively, etc).
- Soundcards that are not supported by ALSA driver can be accessed from Linux over the network, using a Mac OS X audio host.

5 Challenges and TODOs

This project is considered work in progress and is not yet finished. There are many details that need to be refined in order to make this toolchain fully usable. In particular, the following topics need to be addressed.

- Get the latency down. There are currently problems with untight scheduling in the PulseAudio client implementation, and too big buffer sizes.
- Considerations for multi-architecture libraries and binaries. XCode is not the problem in this regard, but the autoconf/automake build system is.
- The clocking model is subject to reconsideration. While things are comparitively easy in scenarios dealing with real hardware soundcards, it becomes more obfuscated in this virtual case as the PulseAudio daemon is the driving part for all clocks. That means that if audio is actually routed into a null-sink on the PulseAudio side, the virtual sound card will play at high speed, which might cause problems with audio applications that assume real-time playing.
- Cosmetic work on the GUI tools to give them the look of a nice tool users want to accept as part of their system. Currently, they look like debug tools for developers.
- Testing. Of course. The whole project is rather fresh, so it hasn't seen a lot of testers yet.

6 Trademarks

7 Acknowledgements

My thanks go to the world-wide Linux audio community for providing ALSA and PulseAudio as sophisticated audio layers on Linux, making this project possible at all.