Proceedings of the Linux Symposium

June 27th–30th, 2007
Ottawa, Ontario
Canada

Conference Organizers

Andrew J. Hutton, *Steamballoon, Inc.*
C. Craig Ross, *Linux Symposium*

Review Committee

Andrew J. Hutton, *Steamballoon, Inc.*
Dirk Hohndel, *Intel*
Martin Bligh, *Google*
Gerrit Huizenga, *IBM*
Dave Jones, *Red Hat, Inc.*
C. Craig Ross, *Linux Symposium*

Proceedings Formatting Team

John W. Lockhart, *Red Hat, Inc.*
Gurhan Ozen, *Red Hat, Inc.*
John Feeney, *Red Hat, Inc.*
Len DiMaggio, *Red Hat, Inc.*
John Poelstra, *Red Hat, Inc.*

Cleaning up the Linux Desktop Audio Mess

Lennart Poettering
Red Hat, Inc.
lennart@poettering.net

Abstract

Desktop audio on Linux is a mess. There are just too many competing, incompatible sound systems around. Most current audio applications have to support every sound system in parallel and thus ship with sound abstraction layers with a more or less large number of back-end plug-ins. JACK clients are incompatible with ALSA clients, which in turn are incompatible with OSS clients, which in turn are incompatible with ESD clients, and so on. "Incompatible" often means "exclusive;" e.g., if an OSS application gets access to the audio hardware, all ALSA applications cannot access it.

Apple MacOS X has CoreAudio, Microsoft Windows XP has a new user-space audio layer; both manage to provide comprehensive APIs that make almost every user happy, ranging from desktop users to pro audio people. Both systems provide fairly modern, easy-to-use audio APIs, and a vast range of features including desktop audio "bling."

On Linux we should be able to provide the same: a common solution that works on the desktop, in networked thin-client setups and in pro audio environments, scaling from mobile phones to desktop PCs and high-end audio hardware.

1 Fixing the Linux Audio Stack

In my talk, I want to discuss what we can do to clean up the mess that desktop audio on Linux is: why we need a user-space sound system, what it should look like, how we need to deal with the special requirements of networked audio and pro-audio stuff, and how we should expose the sound system to applications for allowing Compiz-style desktop "bling"—but for audio. I then will introduce the PulseAudio sound server as an attempt to fix the Linux audio mess.

PulseAudio already provides compatibility with 90% of all current Linux audio software. It features low-latency audio processing and network transparency in an extensible desktop sound server. PulseAudio is now part of many distributions, and is likely to become the default sound system on Fedora and Ubuntu desktops in the next releases of these distributions.

The talk will mostly focus on the user-space side of Linux audio, specifically on the low-level interface between hardware drivers and user-space applications.

2 Current State of Linux Audio

The current state of audio on Linux and other Free Software desktops is quite positive in some areas, but in many other areas, it is unfortunately very poor. Several competing audio systems and APIs are available. However, none of them is useful in all types of applications, nor does any meet the goals of being easy-to-use, scalable, modern, clean, portable, and complete. Most of these APIs and systems conflict in one way or another.

On the other hand, we have a few components and APIs for specific purposes that are well accepted and cleanly designed (e.g., LADSPA, JACK). Also, Linux-based systems can offer a few features that are not available on competing, proprietary systems. Among them is network transparent audio and relatively low-latency scheduling.

While competing, proprietary systems currently lack a few features the Linux audio stack can offer, they managed to provide a single (specific to the respective OS) well-accepted API that avoids the balkanisation we currently have on Free Software desktops. Most notably, Apple MacOS X has CoreAudio which is useful for the desktop as well as for professional audio applications. Microsoft Windows Vista, on the other hand, now ships a new user-space audio layer, which also fulfills many of the above requirements for modern audio systems and APIs.

In the following sections, I will quickly introduce the systems that are currently available and used on Linux desktop, their specific features, and their drawbacks.

2.1 Advanced Linux Sound Architecture (ALSA)

The ALSA system [2] has become the most widely accepted audio layer for Linux. However, ALSA, both as audio system and as API, has its share of problems:

- The ALSA user-space API is relatively complicated.
- ALSA is not available on anything but Linux.
- The ALSA API makes certain assumptions about sound devices that are only true for hardware devices. Implementing an ALSA plug-in for virtual devices ("software" devices) is not doable without nasty hacks.
- Not "high-level" enough for many situations.
- `dmix` is bug-ridden and incomplete.
- Resampling is very low quality.
- You need different configurations for normal desktop use (`dmix`) and pro audio use (no `dmix`).

On the other hand, it also has some real advantages over other solutions:

- It is available on virtually every modern Linux installation.
- It is very powerful.
- It is (to a certain degree) extendable.

2.2 Open Sound System (OSS)

OSS [7] has been the predecessor of ALSA in the Linux kernel. ALSA provides a certain degree of compatibility with OSS. Besides that the API is available on several other Unixes. OSS is a relatively "old" API, and thus has a number of limitations:

- Doesn't offer all the functionality that modern sound hardware provides which needs to be supported by the software (such as no surround sound, no float samples).
- Not high-level enough for almost all situations, since it doesn't provide sample format or sample rate conversions. Most software silently assumes that S16NE samples at 44100Hz are available on all systems, which is no longer the case today.
- Hardly portable to non-Unix systems.
- `ioctl()`-based interface is not type-safe, not the most user-friendly.
- Incompatible with everything else; every application gets exclusive access to the sound device, thus blocking all other applications from accessing it simultaneously.
- Almost impossible to virtualize correctly and comprehensively. Hacks like `esddsp`, `aoss`, `artsdsp` have proven to not work.
- Applications silently assume the availability of certain driver functionality that is not necessarily available in all setups. Most prominently, the 3D game Quake doesn't run with drivers that don't support `mmap()`-based access to the DMA audio buffer.
- No network transparency, no support for desktop "bling."

The good things:

- Relatively easy to use;
- Very well accepted, even beyond Linux;
- Feels very Unix-ish.

2.3 JACK

The JACK Audio Connection Kit [3] is a sound server for professional audio purposes. As such, it is well accepted in the pro-audio world. Its emphasis, besides playback of audio through a local sound card, is streaming audio data between applications.

Plusses:

- Easy to use;
- Powerful functionality;
- It is a real sound server;
- It is very well accepted in the pro-audio world.

Drawbacks:

- Only floating point samples;
- Fixed sampling rate;
- Somewhat awkward semantics which makes it unusable as a desktop audio server (i.e., server doesn't start playback automatically, needs a manual "start" command);

- Not useful on embedded machines;
- No network transparency.

2.4 aRts

The KDE sound server aRts [5] is no longer actively developed and has been orphaned by its developer. Having a full music synthesiser as desktop sound server might not be such a good idea, anyway.

2.5 EsounD

The Enlightened Sound Daemon (EsounD or ESD) [4] has been the audio daemon of choice of the GNOME desktop environment since GNOME 1.0 times. Besides basic mixing and network transparency capabilities, it doesn't offer much. Latency querying, low-latency behaviour, and surround sound are not available at all. It is thus hardly useful for anything beyond basic music playback or playing event sounds ("bing!").

2.6 PortAudio

The PortAudio API [6] is a cross-platform abstraction layer for audio hardware access. As such it sits on top of other audio systems, like OSS, ALSA, ESD, the Windows audio stack, or MacOS X's CoreAudio. PortAudio has not been designed with networked audio devices in mind, and also doesn't provide the necessary functionality for clean integration into a desktop sound server. PortAudio has never experienced wide adoption.

3 What we Need

As shown above, none of the currently available audio systems and APIs can provide all that is necessary on a modern desktop environment. I will now define four major goals which a new desktop audio system should try to achieve.

3.1 A Widely Accepted, Modern, Portable, Easy-to-Use, Powerful, and Complete Audio API

None of the described Linux audio APIs fulfills all requirements that are expected from a modern audio API. On the other hand, Apple's CoreAudio and Microsoft's new Windows Vista user-space audio layer reach this goal, for the most part. More precisely, a modern sound API for Free Software desktops should fulfill the following requirements:

- Completeness: an audio API should be a general-purpose interface; it should be suitable for simple audio playback as well as professional audio production.
- Scalability: usable on all kinds of different systems, ranging from embedded systems to modern desktop PCs and pro audio workstations.
- Modernness: provide good integration into the Free Software desktop ecosystem.
- Proper support for networked and "software" (virtual) devices, besides traditional hardware devices.
- Portability: the audio API should be portable across different operating systems.
- Easy-to-use for both the user, and for the programmer. This includes a certain degree of automatic fine-tuning, to provide optimal functionality with "zero configuration."

3.2 Routing and Filtering Audio in Software

Classic audio systems such as OSS are designed to provide an abstract API around hardware devices. A modern audio system should provide features beyond that:

- It needs to be possible to play back multiple audio streams simultaneously, so that they are mixed in real time.
- Applications should be able to hook into what is currently being played back.
- Before audio is written to an output device, it might be transferred over the network to another machine.
- Before audio is played back some kind of post-processing might take place.
- Audio streams might need to be re-routed during playback.

3.3 Desktop "Bling"

Free Software desktops currently lack an audio counterpart for the well known window manager Compiz. A modern desktop audio systems should be able to provide:

- Separate per-application and per-window volumes.
- Soft fade-ins and fade-outs of music streams.
- Automatically increasing the volume of the application window in the foreground, decreasing the volume of the application window in the background.
- Forward a stop/start request to any music-playing applications if a VoIP call takes place.
- Remember per-application and per-window volumes and devices.
- Reroute audio to a different audio device on-the-fly without interruption, from within the window manager.
- Do "hot" switching between audio devices whenever a new device becomes available. For example, when a USB headset is plugged in, automatically start using it by switching an in-progress VoIP call over to the new headset.

3.4 A Compatible Sound System

Besides providing the features mentioned above, a new sound system for Linux also needs to retain a large degree of compatibility with all the currently available systems and APIs, as much as possible. Optimally, all currently available Linux audio software should work simultaneously and without manual intervention. A major task is to marry the pro-audio and desktop audio worlds into a single audio system.

4 What PulseAudio already provides

The PulseAudio [1] sound server is our attempt to reach the four aforementioned goals. It is a user-space sound server that provides network transparency, all kinds of desktop "bling," relatively low-latency, and is extensible through modules. It sits atop of OSS and ALSA sound devices and routes and filters audio data, possibly over the network.

PulseAudio is intended to be a replacement for systems like ESD or aRts. The former is entirely superseded; PulseAudio may be installed as drop-in replacement for EsounD on GNOME desktops.

PulseAudio ships with a large set of modules (plug-ins):

- Driver modules (i.e., accessing OSS, ALSA, Win32, Solaris drivers).
- Protocol support (i.e., native TCP-based protocol, EsounD protocol, RTP).
- Integration into LIRC, support for multimedia keyboards.
- Desktop integration (i.e., hooks into the X11 system for authentication and redirecting the X11 bell).
- Integration with JACK, EsounD.
- Zeroconf support, using Avahi.
- Management: Automatically restore volumes, devices of playback streams, move a stream to a different device if its original devices becomes unavailable due to a hot-plug event.
- Auto-configuration: integration with HAL for automatic and dynamic configuration of the sound server based on the available hardware.
- Combination of multiple audio devices into a single audio device while synchronising audio clocks.

PulseAudio is not intended to be a competitor to JACK, GStreamer, Helix, KDE Phonon, or Xine. Quite the opposite: we already provide good integration into JACK, GStreamer, and Xine. We have different goals.

The PulseAudio core is carefully optimised for speed and low latency. Local clients may exchange data with the PulseAudio audio server over shared memory data transfer. The PulseAudio sound server will never copy audio data blocks around in memory unless it is absolutely necessary. Most audio data operations are based on `liboil`'s support for the extended instruction sets of modern CPUs (MMX, SSE, AltiVec).

Currently the emphasis for PulseAudio is on networked audio, where it offers the most comprehensive functionality.

PulseAudio support is already available in a large number of applications. For others, we have prepared patches. Currently we have native plug-ins, drivers, patches, and compatibility for Xine, MPlayer, GStreamer, `libao`, XMMS, Audacious, ALSA, OSS (using `$LD_PRELOAD`), EsounD, Music Player Daemon (MPD), and the Adobe Flash player.

PulseAudio has a small number of graphical utility applications:

- Volume Control.

- Panel Applet (for quickly changing the output device, selecting it from a list of Zeroconf-announced audio devices from the network).
- Volume Meter.
- Preferences panel, for a user-friendly configuration of advanced PulseAudio functionality.
- A management console to introspect a PulseAudio server's internals.

5 PulseAudio Internals

5.1 Buffering Model

PulseAudio offers a powerful buffering model which is an extension of the model Jim Gettys pioneered in the networked audio server AF [9]. In contrast to traditional buffering models it offers flexible buffering control, allowing large buffers—which is useful for networked audio systems—while still providing quick response to external events. It allows absolute and relative addressing of samples in the audio buffer and supports a notion of "zero latency." Samples that have already been pushed into the playback buffer may be rewritten at any time.

5.2 Zero-Copy Memory Management

Audio data in the PulseAudio sound server is stored in reference-counted memory blocks. Audio data queues contain only references to these memory blocks instead of the audio data itself. This provides the advantage of minimising copying of audio data in memory, and also saves memory. In fact the PA core is written in a way that, in most cases, data arriving on a network socket is written directly into the sound card DMA hardware buffer without spending time in bounce buffers or similar. This helps to keep memory usage down and allows very low-latency audio processing.

5.3 Shared Memory Data Transfer

Local clients can exchange audio data with a local PulseAudio daemon through shared-memory IPC. Every process allocates a shared memory segment where it stores the audio data it wants to transfer. Then, when the data is sent to another process, the recipient receives only the information necessary to find the data in that segment. The recipient maps the segment of the originator in read-only mode and accesses the data.

The shared memory data transfer is the natural extension of the aforementioned zero-copy memory management, for communication between processes.

5.4 Synchronisation

Multiple streams can be synchronised together on the server side. If this is done, it is guaranteed that the playback indexes of these streams never deviate. Clients can label stream channels freely (e.g., "left," "right," "rear-left," "rear-right," and so on). Together with the aforementioned buffering model, this allows implementation of flexible server-side multi-track mixing.

5.5 Buffer Underrun Handling

PulseAudio does its best to ensure that buffer underruns have no influence on the time axis. Two modes are available: In the first mode, playback pauses when a buffer underrun happens. This is the mode that is usually available in audio APIs such as OSS. In the second mode playback never stops, and if data is available again, enough data is skipped so that the time function experiences no discontinuities.

6 Where we are going

While the PulseAudio project in the current state fulfills a large part of the aforementioned requirements, it is not complete yet. Compatibility with many sound systems, a wide range of desktop audio "bling," networked audio, and low-latency behaviour are already available in current versions of PulseAudio. However we are still lacking in other areas:

- Better low-latency integration into JACK.
- Some further low-latency fixes can be made.
- Better portability to systems which don't support floating-point numbers.
- The client API PulseAudio currently offers is comparatively complicated and difficult to use.

Besides these items, there are also a lot of minor issues to be solved in the PulseAudio project. In the following subsections, I quickly describe the areas we are currently working on.

6.1 Threaded Core

The current PulseAudio core is mostly single-threaded. In most situations this is not a problem, since we carefully make sure that no operation blocks for longer than necessary. However, if more than one audio device is used by a single PulseAudio instance, or when extremely low latencies must be reached, this may become a problem. Thus the PulseAudio core is currently being moved to a more threaded design: every PulseAudio module that is important in low-latency situations will run its own event loop. Communication between those separate event loops, the main event loop, and other local clients is (mostly) done in a wait-free fashion (mostly not lock-free, however). The design we are currently pursuing allows a step-by-step upgrade to this new functionality.

6.2 `libsydney`

During this year's Foundation of Open Media Software (FOMS) conference in January in Sydney, Australia, the most vocally expressed disappointment in the Linux audio world is the lack of a single well-defined, powerful audio API which fulfills the requirements of a modern audio API as outlined above. Since the current native PulseAudio API is powerful but unfortunately overly complex, we took the opportunity to define a new API at that conference. People from Xiph, Nokia, and I sat down to design a new API.

Of course, it might appear as a paradox to try fix the balkanisation of Linux audio APIs by adding yet another one, but given the circumstances, and after careful consideration, we decided to pursue this path. This new API was given the new name `libsydney` [8], named after the city we designed the first version of the API in. `libsydney` will become the only supported API in future PulseAudio versions. Besides working on top of PulseAudio, it will natively support ALSA, OSS, Win32, and Solaris targets. This means that developing a client for PulseAudio will offer cross-platform support for free. `libsydney` is currently in development; by the time of the OLS conference, an initial public version will be made available.

6.3 Multi-User

Currently the PulseAudio audio server is intended to be run as a session daemon. This becomes a problem if multiple users are logged into a single machine simultaneously. Before PulseAudio can be adopted by modern distributions, some kind of hand-over of the underlying audio devices will need to be implemented to support these multi-user setups properly.

7 Where you can get it

PulseAudio is already available in a large number of distributions, including Fedora, Debian, and Ubuntu. Since it is a drop-in replacement for EsounD, it is trivial to install it and use it as the desktop sound server in GNOME.

Alternatively, you may download a version from our web site [1].

It is planned for PulseAudio to replace EsounD in the default install in the next versions of Fedora and Ubuntu.

8 Who we are

PulseAudio has been and is being developed by Lennart Poettering (Red Hat, Inc.) and Pierre Ossman (Cendio AB).

References

[1] PulseAudio, `http://pulseaudio.org/`

[2] ALSA, `http://alsa-project.org/`

[3] JACK Audio Connection Kit, `http://jackaudio.org/`

[4] EsounD, `http://www.tux.org/~ricdude/overview.html`

[5] aRts, `http://www.arts-project.org/`

[6] PortAudio, `http://www.portaudio.com/`

[7] Open Sound System, `http://www.opensound.com/oss.html`

[8] libsydney, `http://0pointer.de/cgi-bin/viewcvs.cgi/trunk/?root=libsydney`

[9] AF, `http://tns-www.lcs.mit.edu/vs/audiofile.html`