The PipeWire multimedia framework and its potential in AGL

George Kiagiadakis,
Senior Software Engineer
What is PipeWire

- Initial idea: PulseAudio for video
- Now: generic multimedia daemon
  - Video capture server
    - Camera and other video sources (ex. gnome-shell screencast)
  - PulseAudio and Jack (pro-audio) replacement
    - Borrowing ideas also from CoreAudio, AudioFlinger, and others...
Architecture

- Multi-process, graph based processing
- External session/policy management (unlike PulseAudio)
  - Nothing happens automatically inside the daemon
  - Per-desktop/distro implementations can exist
- External applications can be device providers
- Real-time ultra low-latency (pro-audio) and standard high latency (typical desktop)
PipeWire daemon

Different volume per link

Splitter ('tee') on each port

Different apps on different zones

Each node has a role tag

Similar features also for video

Devices can be made available by external processes

Can use a hardware DSP when the node presents the appropriate interface

Different devices presented as different nodes

Software DSP plugged automatically for other ALSA output nodes

Numerous useful plugins available

Ports can have different capabilities

Audio app role: navigation

Audio app role: emergency

Audio app role: multimedia

ALSAs in hw:1,0

ALSA in hw:0,0

FR

RL

RR

FL

Echo-cancel (plugin)

Video call app role: voip

V4L2 in /dev/video0

PHONES over bluetooth (remote audio manager)

ALSAs out with HW DSP hw:1,0

ALSAs out hw:0,1
Processing Efficiency

- Zero-copy with modern linux kernel APIs (memfd, dmabuf)
  - Buffers passed around with file descriptors
  - **memfd**: shared CPU buffer
    - Kernel ensures only the destination process can modify the buffer
  - **dmabuf**: shared hardware (GPU/VPU) buffer
    - Processing happens in dedicated hardware, the CPU does not need to copy data on the CPU-accessible RAM
Processing Efficiency (continued...)

- Plugins based on SPA (Simple Plugin API)
  - Header-only C library with zero dependencies
  - Extremely lightweight data structures
  - “Like GStreamer, but not so heavy!” - Wim Taymans

- Much lower CPU usage than PulseAudio
  - PulseAudio CPU skyrockets on low latency, even on a powerful i7
  - PipeWire CPU will happily stay low at any latency and any load
CPU Usage Statistics
Playback of a 24bit 96kHz 5.1 channel file, downmixed to 3.1 and resampled to 48kHz

- Measurements:
  - 21.33 ms (1024 samples / buffer)
  - 1.33 ms (64 samples / buffer)
  - 1.33 ms with 2 clients
  - 1.33 ms with CPU pinned @ 800 MHz

- Measurements on Intel(R) Core(TM) i7-4770 CPU @ 3.40GHz

- Comparatively, on 1.33 ms, PulseAudio uses 100% CPU and fails (underruns)
Power Efficiency

- **eventfd** to wake up the processes and schedule the graph
- Devices can be put to *sleep* when unused
  - Like PulseAudio does
  - Unlike ALSA or Jack
Security

• Fine-grained access controls per client
  - Clients are not able to list other nodes or connect to them until the session manager approves
  - Each client can be made to “see” an entirely different graph

• Sandbox / Container support (flatpak, …)
  - Sandbox portal requests a fd from PipeWire and asks the session manager for specific permissions
Security

Standard Desktop:

With sandbox:

flatpak sandbox
Compatibility

- Provides replacement compatibility libraries for:
  - PulseAudio (libpulse.so.0)
  - Jack (libjack.so.0)
  - Pulse & Jack applications can natively work with PW without recompilation
- ALSA plugin for ALSA-only clients
- LADSPA & LV2 plugins supported, apart from SPA
- Native API via UNIX Socket (D-Bus available; extensible)
Behavior

- PulseAudio is nasty with crash handling
  - Restarts automatically

- PipeWire doesn’t inherit this behavior
  - The service is meant to be started & restarted by systemd, using socket activation
  - It is up to the session/policy manager to restore connections, if necessary
ALSA UCM

• PipeWire wants to use ALSA UCM (Use Case Manager) to configure sound cards
  - Allows dynamic reconfiguration of ports
  - Allows power on/off on parts of the sound hardware
  - Hides ALSA controls complexity

• Open to discussion on improving it or making it optional

• Currently not implemented (in TODO list)
Who is behind this

- Author: Wim Taymans
  - Well-known old GStreamer developer (since the very early days) & ex-maintainer
  - Sponsored by: Red Hat

- Embraced by PulseAudio developers
  - Seen as the next generation of PulseAudio
  - PulseAudio will eventually be phased out

- Welcomed by ALSA and Jack developers
PipeWire in AGL
The idea

Application

direct call or via libpipewire

Restricted client fd

4A High Level
-> Device Access API

4A HAL

AGL PipeWire Session
Policy Manager

request

PipeWire

Bluetooth & misc
remote device
provider

ALSA

Unicens

Other
backends
Why?

• Stick to a (to-be) widely used & widely tested audio system
  - Implemented, maintained and supported by PulseAudio & GStreamer experts

• Provide flexibility for vendors to implement certain processing in hardware
  - by SPA plugins or external device providers
  - without having to implement a full-blown replacement for the 4A softmixer
  - without writing AGL-specific software
Why?

• Provide more advanced capabilities for free
  - Advanced mixing configurations
  - Dynamic audio routing
  - Lower latency for applications that require it
  - Software effect plugins (echo canceler, equalizer, you name it...)
  - Standard Linux Bluetooth audio implementation (from PulseAudio)
  - Airplay, DLNA
  - RTP streaming to other car nodes with synchronization
Why?

• Extend device arbitration to non-audio devices
  - Video capture devices (ex. camera for video calls)
  - Hardware encoders / decoders
  - Media-based sensors (ex. camera for AI)
Why?

- Zero-copy, processing & power efficiency
  - Current 4A softmixer requires copying buffers back and forth through the kernel:
    app → ALSA loop device → softmixer → real ALSA device
  - AVIRT solves that, but unnecessarily introduces AGL-specific kernel code
Why?

- Better security on the audio connection
  - Currently applications can just open an ALSA device without asking the 4A HL
  - SMACK will allow it if the application is labeled to be capable of opening this device (ex. multimedia application opening the “multimedia” sound device)
  - In PipeWire, the session/policy manager will be the one to make the final decision
  - We can arbitrate access to the socket via a binding-like mechanism
  - Also solves the problem of media frameworks not being aware of the AGL security mechanisms and 4A
Why?

- Avoid using complicated ALSA plugins
  - Old, ugly codebase
  - Hard to use and therefore hard to maintain a “softmixer” that works well
  - Limited functionality
  - Not as efficient
Proposal

• Do a minimal demo
  - The least modifications required to get PipeWire to provide the audio backend

• Work with upstream
  - Provide missing bits in PipeWire
  - Decouple functionality from PulseAudio and make it available
  - Ensure that all AGL specific requirements are supported as early as possible
Thank you!