

Chromium OS audio

CRAS audio server

Why another audio server?

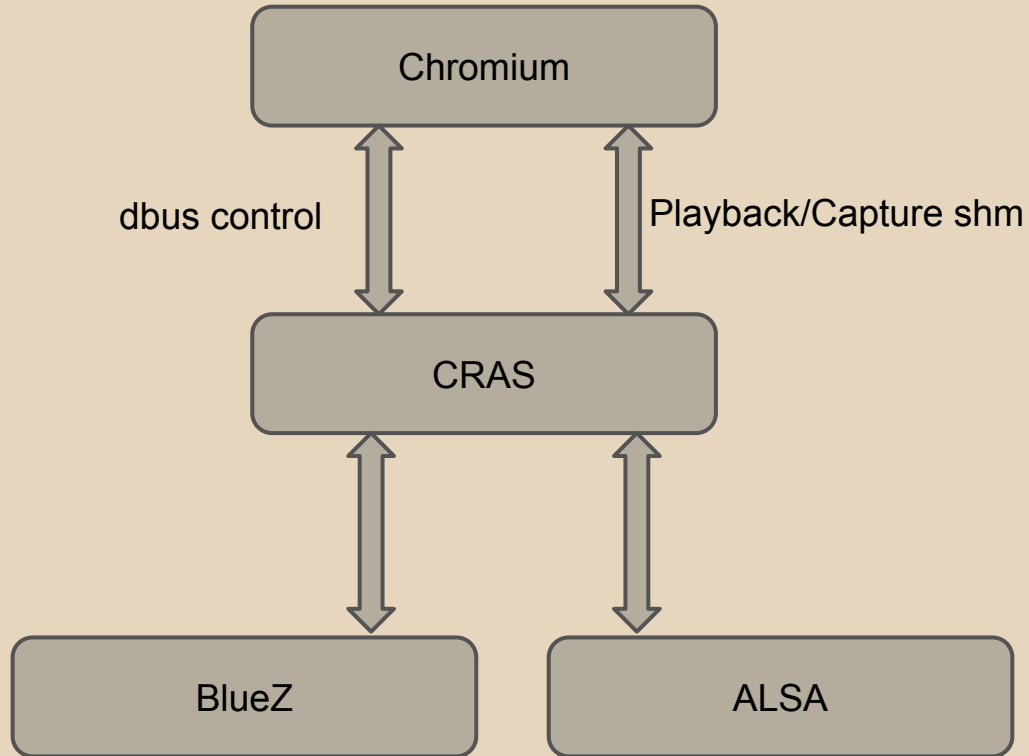
low end hardware (1 core atom, or Tegra 2)

optimize for one user (chrome)

dynamic stream re-routing

maintainability, code size, security

Basic Audio Flow



Client Library API

```
int cras_client_create(struct cras_client **client);
```

```
int cras_client_run_thread(struct cras_client *client);
```

```
struct cras_stream_params *cras_client_unified_params_create(  
    enum CRAS_STREAM_DIRECTION direction, /* direction - CRAS_STREAM_OUTPUT or CRAS_STREAM_INPUT  
*/  
    unsigned int block_size, /* block_size - The number of frames per callback(dictates latency). */  
    enum CRAS_STREAM_TYPE stream_type, /* not currently used */  
    uint32_t flags, /* not used either. */  
    void *user_data, /* user_data - Pointer that will be passed to the callback. */  
    cras_unified_cb_t unified_cb, /* unified_cb - Called for each block size samples */  
    cras_error_cb_t err_cb, /* err_cb - Called when there is an error with the stream. */  
    struct cras_audio_format *format); /* format - Specifies bits per sample, num chan, sample rate */
```

```
int cras_client_add_stream(struct cras_client *client, cras_stream_id_t *stream_id_out, struct cras_stream_params  
*config);
```

Server side features

Timer Based Wake-ups based on stream level

Device Sample Rate Estimation

Mixing, DSP, and format conversion

Volume level tuning

Device synchronization

One audio thread

Wake up timing

Wakes up each stream based on a timer

Timer rate set based on block size

Adjusted based on estimated device clock

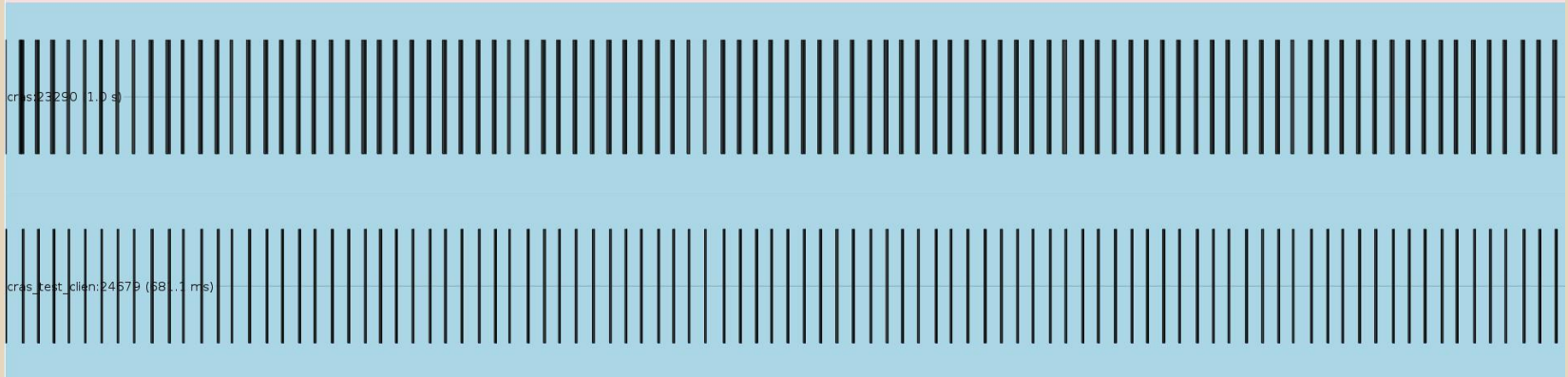
Underrun handling

Don't let one stream cause all to glitch

Scheduling jitter, real time threads help

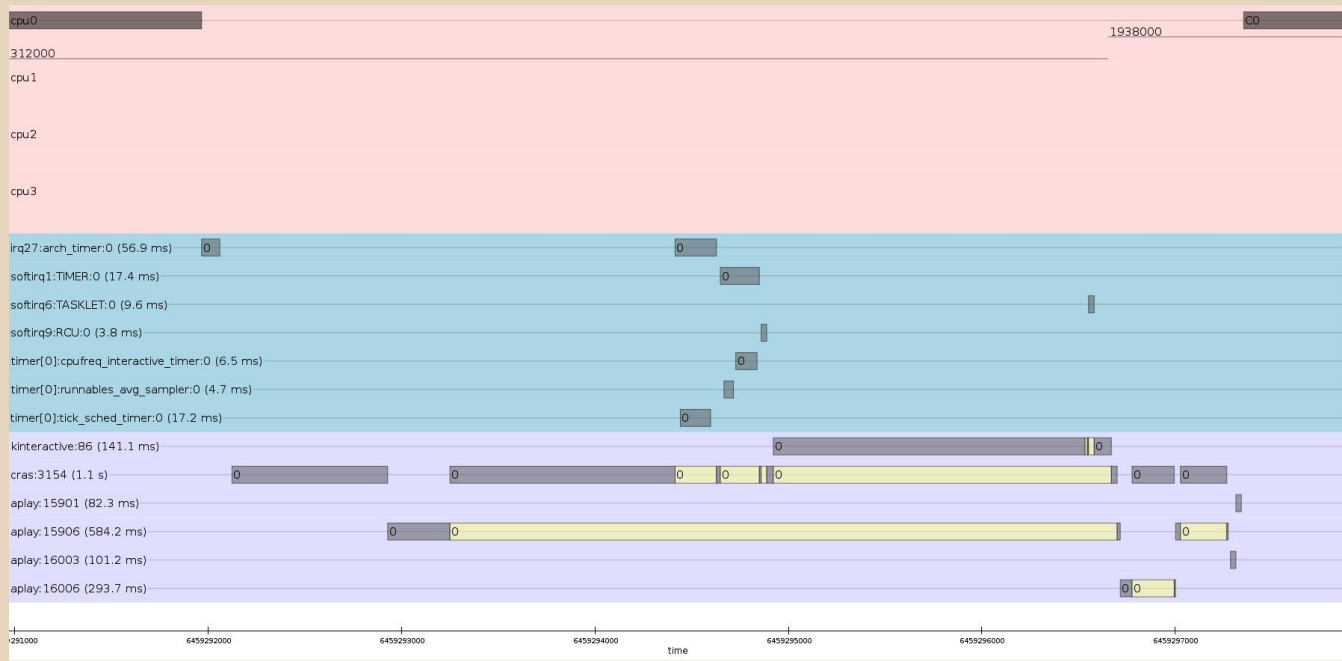
Timing picture

One 44.1kHz stream (good)



Timing glitch

Interactive governor preempts



Device Rate Estimation

By measuring the buffer level at known times estimate the sample rate of the device

Use this estimated sample rate to calculate how long each stream should sleep

Buffer Management

Avoid copying audio

Shared memory between client and server

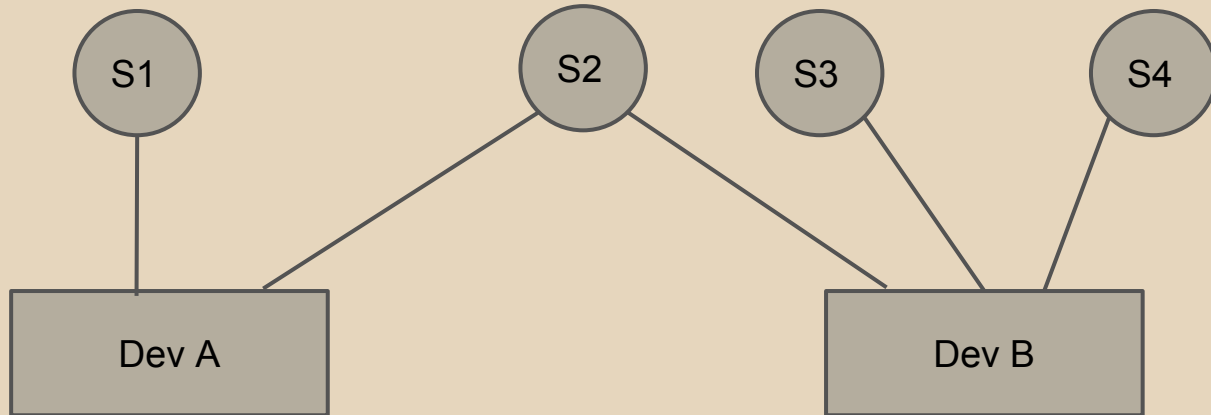
Server moves audio directly from shm to
mmaped device buffer

Format conversion still needs an extra copy.

avoid copying audio continued

Buffer write/read point management is tricky

n devices reading from the same stream and n streams writing to the same device



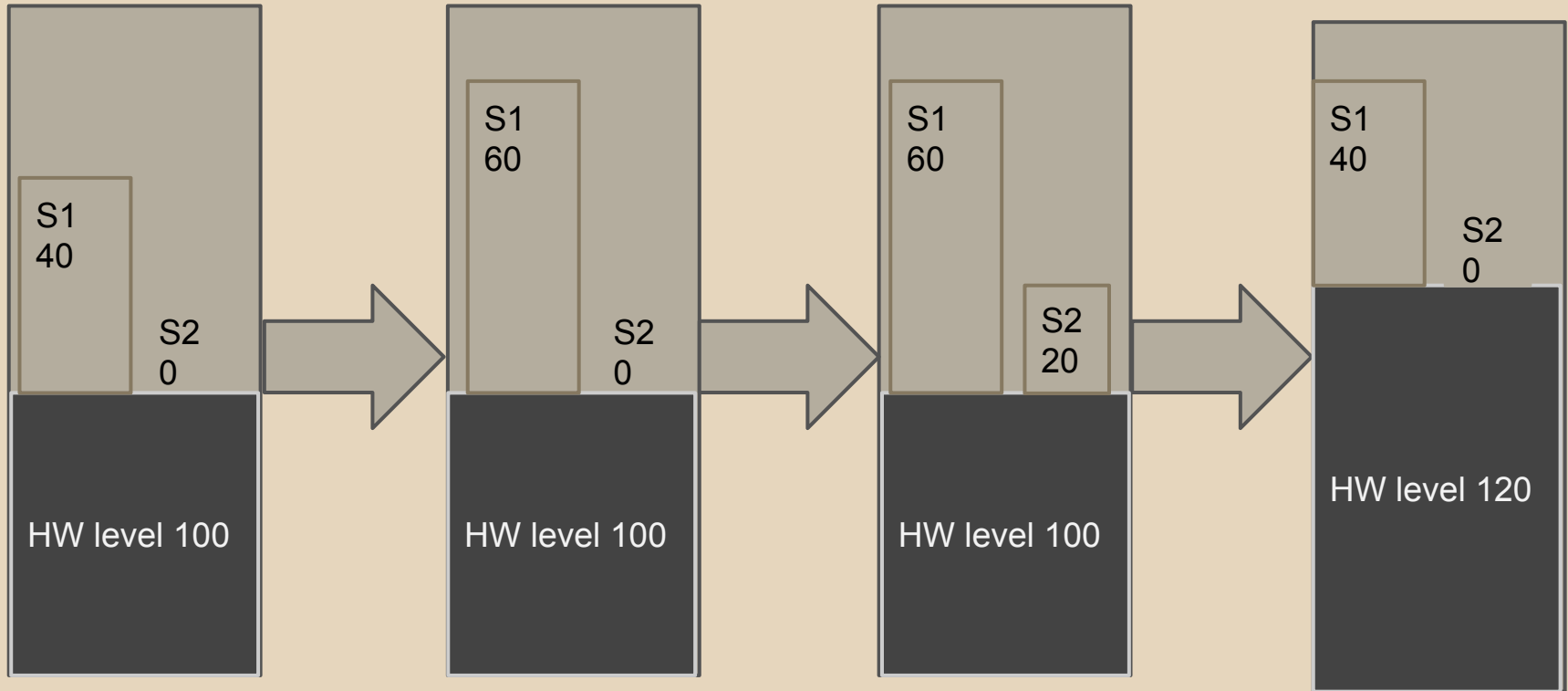
Write pointer management

Each device tracks offset of all streams attached to it.

Similarly each stream tracks its offset into each device it is attached to.

offsets are updated after all users have mixed

Two output stream example



Output Processing

Output processing

Need Speaker EQ

Each system is different

Different OEMs want different tunings

DSP speed

Output and input processing are on the critical path.

Heavily optimized with neon and sse versions.

Keep it simple with a three band DRC and a 10 band eq per channel.

DSP tuning

Can listen in real time on un-tuned device, WebAudio blocks are equivalent to the blocks used in the optimized DSP.

Generated config file, hands off tuning for core engineering team.

audio-tuning.appspot.com

Audio Tuning

audio-tuning.appspot.com

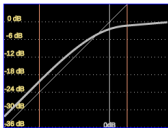
Audio Source

URL: Local Audio File

0:00

Enable DRC Enable EQ Show FFT Swap DRC/EQ

DRC



Threshold

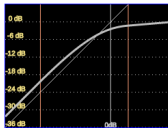
Knee

Ratio

Boost

Attack

Release



Start From

Threshold

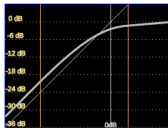
Knee

Ratio

Boost

Attack

Release



Start From

Threshold

Knee

Ratio

Boost

Attack

Release

EQ Left

Hz	47Hz	94Hz	188Hz	375Hz	750Hz	1500Hz	3000Hz	6000Hz	12000Hz	24000Hz
18dB										
12dB										
6dB										
0dB										
-6dB										
-12dB										
-18dB										

Screenshot taken
Click to view

Copy to clipboard

volume-tuning.appspot.com

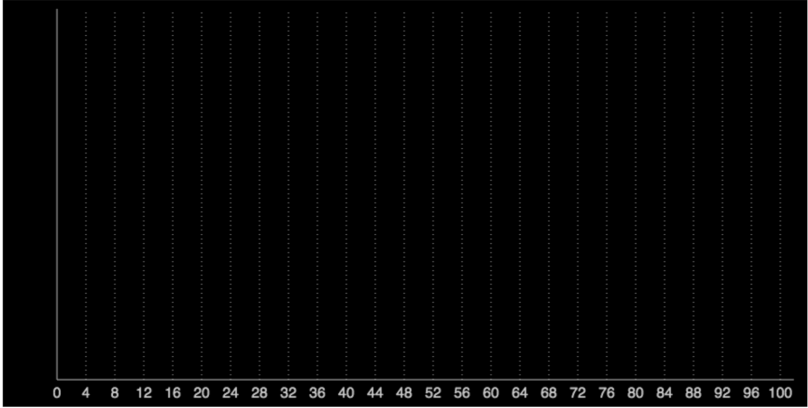
Browser tabs: Fwd: i2s-out ALSA driver | x | Volume Tuning | x | LinuxCon + CloudOpen | x | CRAS ELC 2014 - Google | x | x

Address bar: volume-tuning.appspot.com

Volume Tuning

Min: dB
Max: dB
Step: dB

0:	<input type="text"/> dB
4:	<input type="text"/> dB
8:	<input type="text"/> dB
12:	<input type="text"/> dB
16:	<input type="text"/> dB
20:	<input type="text"/> dB
24:	<input type="text"/> dB
28:	<input type="text"/> dB
32:	<input type="text"/> dB
36:	<input type="text"/> dB
40:	<input type="text"/> dB
44:	<input type="text"/> dB
48:	<input type="text"/> dB
52:	<input type="text"/> dB
56:	<input type="text"/> dB
60:	<input type="text"/> dB
64:	<input type="text"/> dB
68:	<input type="text"/> dB
72:	<input type="text"/> dB
76:	<input type="text"/> dB
80:	<input type="text"/> dB
84:	<input type="text"/> dB
88:	<input type="text"/> dB
92:	<input type="text"/> dB
96:	<input type="text"/> dB
100:	<input type="text"/> dB



0 4 8 12 16 20 24 28 32 36 40 44 48 52 56 60 64 68 72 76 80 84 88 92 96 100

[Download config](#)

UCM

ALSA UCM usage

Use a single ‘HiFi’ verb.

Have devices for headphones, external microphones, and HDMI.

A few non-standard values.

ALSA UCM example

```
SectionVerb {
    Value {
        OutputDspName "speaker_eq"
    }
    EnableSequence [
        cdev "hw:Venice2"
        cset "name='Left Speaker Mixer Left DAC Switch' on"
        cset "name='Right Speaker Mixer Right DAC Switch' on"
        ...snip...
        cset "name='Headphone Left Switch' on"
        cset "name='Headphone Right Switch' on"
        cset "name='Speaker Left Switch' on"
        cset "name='Speaker Right Switch' on"
        cset "name='Speakers Switch' on"
    ]
    DisableSequence [
    ]
}
```

ALSA UCM example headphones

```
SectionDevice."Headphone".0 {
    Value {
        JackName "NVIDIA Tegra Venice2 Headphone Jack"
        OutputDspName ""
    }
    EnableSequence [
        cdev "hw:Venice2"
        cset "name='Speakers Switch' off"
        cset "name='HP Left Out Switch' on"
        cset "name='HP Right Out Switch' on"
    ]
    DisableSequence [
        cdev "hw:Venice2"
        cset "name='HP Left Out Switch' off"
        cset "name='HP Right Out Switch' off"
        cset "name='Speakers Switch' on"
    ]
}
```


ALSA UCM example microphone

```
SectionDevice."Mic".0 {
    Value {
        JackName "NVIDIA Tegra Venice2 Mic Jack"
        CaptureControl "MIC2"
    }
    EnableSequence [
        cdev "hw:Venice2"
        cset "name='Int Mic Switch' off"
        cset "name='DMIC Mux' ADC"
        cset "name='Mic Jack Switch' on"
    ]
    DisableSequence [
        cdev "hw:Venice2"
        cset "name='Mic Jack Switch' off"
        cset "name='DMIC Mux' DMIC"
        cset "name='Int Mic Switch' on"
    ]
}
```

External Device Support

USB/Bluetooth

Bluetooth chip attached through USB not i2s

USB devices go through ALSA

Bluetooth through a Bluez created socket
A2DP/HFP/HSP

USB/Bluetooth transfer size

Main challenge is granularity of transfers

Data is sent over USB in URB sized chunks

For Bluetooth one MTU can be > 500 samples

No way to detect the size from user space

Have to pad buffers to ensure enough is ready

HDMI audio output

Auto routing decision is difficult

EDID parsing helps

Docked mode

Improvements for embedded systems

Process hop eliminated on one user systems

Add local streams

Make timing and device management a separate library

Avoid waking up

Could improve a lot here

Don't try to synchronize streams at all

Favor accuracy of stream callbacks over wakeup aggregation.

Performance Measurement

CPU usage

measure # of instructions over 5 seconds of playback, averaged three runs each.

```
perf stat -p <server pid>,<player pid> -r 3 -a sleep 5
```

All tests were on a TegraK1 chromebook.
Crouton was used to run pulseaudio

perf output

```
localhost / # perf stat -p 12912,12922 -r 3 -a sleep 5
```

Performance counter stats for process id '12912,12922' (3 runs):

```
 940.900007 task-clock (msec)    #    0.188 CPUs utilized    ( +- 26.80% ) [100.00%]
    4,937 context-switches      #    0.005 M/sec            ( +- 1.18% ) [99.98%]
     6 cpu-migrations          #    0.007 K/sec            ( +- 92.22% )
     0 page-faults              #    0.000 K/sec
398,970,008 cycles              #    0.424 GHz              ( +- 10.34% )
<not supported> stalled-cycles-frontend
<not supported> stalled-cycles-backend
 85,783,097 instructions        #    0.22 insns per cycle    ( +- 6.28% )
 21,426,180 branches            #   22.772 M/sec            ( +- 9.23% )
 2,620,234 branch-misses        #   12.23% of all branches    ( +- 0.85% )

5.005028610 seconds time elapsed ( +- 0.06% )
```

Single 44.1kHz wav file with aplay

```
aplay -D<plugin> -B20000 filename.wav
```

plugin	instructions (millions)	task-clock (msec)
hw:1	11.2	294
pulse	188.9	1686.4
cras	85.8	940.9

One 44.1kHz one 48kHz

```
aplay -D<plugin> -B20000 44_1k.wav
```

```
aplay -D<plugin> -B20000 48k.wav
```

plugin	instructions (millions)	task-clock (msec)
pulse	576.2	1755.1
cras	247.1	837.8

Native clients at 44.1kHz

```
pacat --rate 44100 --latency 1764 --raw /dev/zero
```

```
ctc --playback_file /dev/zero --block_size 441 --rate 44100
```

pacat	85.1	1506.0
ctc	27.4	673.5

Native clients 44.1kHz and 48kHz

pacat	356.1	1613.0
ctc	102.0	1506.0