Ludit

Aug 19, 2023

Contents:

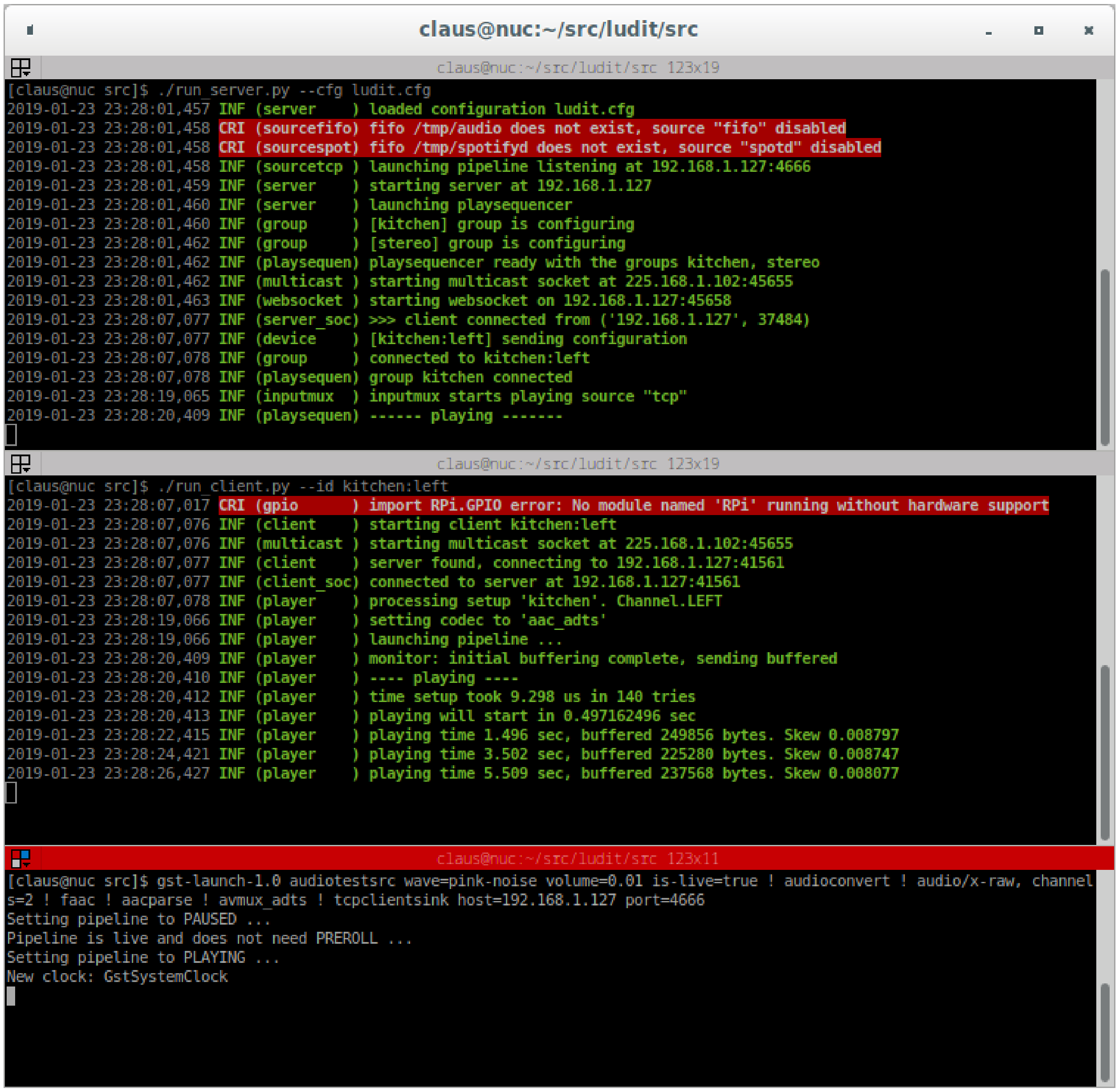
CHAPTER 1

Quick start

Probably the fastest and or simplest way to get Ludit playing is on a x86 PC. Most notably it skips the fancy audio sources and simply plays noise generated locally with gstreamer. Since everything will run on a single PC only one client is started as a second client might be unable to open Alsa. Regarding audio this means that only a single channel is playing (if two clients are running on the same default Alsa device this would have given a mono playback). Once the server and a client are running it will be possible to play with the audio settings such as crossover and equalizer via a webpage.

Note that since everything is running on a single computer there is no time drift to worry about (not that it would ever be an issue for a quick test run)

3 terminals will be needed so Terminator will come in handy in case you don't know it already. It will look like the following image, from top to bottom with the server, a client and a gstreamer pipeline.



```
[claus@nuc src]$ ./run_server.py --cfg ludit.cfg
2019-01-23 23:28:01,457 INF (server     ) loaded configuration ludit.cfg
2019-01-23 23:28:01,458 CRI (sourcefifo) fifo /tmp/audio does not exist, source "fifo" disabled
2019-01-23 23:28:01,458 CRI (sourcespot) fifo /tmp/spotifyd does not exist, source "spotd" disabled
2019-01-23 23:28:01,458 INF (sourcetcp ) launching pipeline listening at 192.168.1.127:4666
2019-01-23 23:28:01,459 INF (server     ) starting server at 192.168.1.127
2019-01-23 23:28:01,460 INF (server     ) launching playsequencer
2019-01-23 23:28:01,460 INF (group      ) [kitchen] group is configuring
2019-01-23 23:28:01,462 INF (group      ) [stereo] group is configuring
2019-01-23 23:28:01,462 INF (playsequen) playsequencer ready with the groups kitchen, stereo
2019-01-23 23:28:01,462 INF (multicast ) starting multicast socket at 225.168.1.102:45655
2019-01-23 23:28:01,463 INF (websocket ) starting websocket on 192.168.1.127:45658
2019-01-23 23:28:07,077 INF (server_soc) >>> client connected from ('192.168.1.127', 37484)
2019-01-23 23:28:07,077 INF (device     ) [kitchen:left] sending configuration
2019-01-23 23:28:07,078 INF (group      ) connected to kitchen:left
2019-01-23 23:28:07,078 INF (playsequen) group kitchen connected
2019-01-23 23:28:19,065 INF (inputmux  ) inputmux starts playing source "tcp"
2019-01-23 23:28:20,409 INF (playsequen) ------ playing -------
```

```
[claus@nuc src]$ ./run_client.py --id kitchen:left
2019-01-23 23:28:07,017 CRI (gpio       ) import RPi.GPIO error: No module named 'RPi' running without hardware support
2019-01-23 23:28:07,076 INF (client     ) starting client kitchen:left
2019-01-23 23:28:07,076 INF (multicast ) starting multicast socket at 225.168.1.102:45655
2019-01-23 23:28:07,077 INF (client     ) server found, connecting to 192.168.1.127:41561
2019-01-23 23:28:07,077 INF (client_soc) connected to server at 192.168.1.127:41561
2019-01-23 23:28:07,078 INF (player     ) processing setup 'kitchen'. Channel.LEFT
2019-01-23 23:28:19,066 INF (player     ) setting codec to 'aac_adts'
2019-01-23 23:28:19,066 INF (player     ) launching pipeline ...
2019-01-23 23:28:20,409 INF (player     ) monitor: initial buffering complete, sending buffered
2019-01-23 23:28:20,410 INF (player     ) ---- playing ----
2019-01-23 23:28:20,412 INF (player     ) time setup took 9.298 us in 140 tries
2019-01-23 23:28:20,413 INF (player     ) playing will start in 0.497162496 sec
2019-01-23 23:28:22,415 INF (player     ) playing time 1.496 sec, buffered 249856 bytes. Skew 0.008797
2019-01-23 23:28:24,421 INF (player     ) playing time 3.502 sec, buffered 225280 bytes. Skew 0.008747
2019-01-23 23:28:26,427 INF (player     ) playing time 5.509 sec, buffered 237568 bytes. Skew 0.008077
```

```
[claus@nuc src]$ gst-launch-1.0 audiotestsrc wave=pink-noise volume=0.01 is-live=true ! audioconvert ! audio/x-raw, channel
s=2 ! faac ! aacparse ! avmux_adts ! tcpclientsink host=192.168.1.127 port=4666
Setting pipeline to PAUSED ...
Pipeline is live and does not need PREROLL ...
Setting pipeline to PLAYING ...
New clock: GstSystemClock
```

A number of requirements are listed as 'Common installs' on the *Software installations* page. From the 'Server installs' simple-websocket-server is also needed.

1.1 Server

Enter ./src. The first thing to do is to make a configuration file.

```
./run_server.py --newcfg > ludit.cfg
```

Have a look at the file, probably there is a group called 'kitchen' with two speakers caller 'left' and 'right'. These names will be needed for launching the clients. Now start the server:

```
./run_server.py --cfg ludit.cfg
```

Note that if the server and client(s) are running on different computers then only one NIC should be up on each.

1.2 Client

In a seperate terminal enter ./src and launch a client:

```
./run_client.py --id kitchen:left
```

Client and server should now have connected automatically (they discover each other via multicast).

1.3 Audio

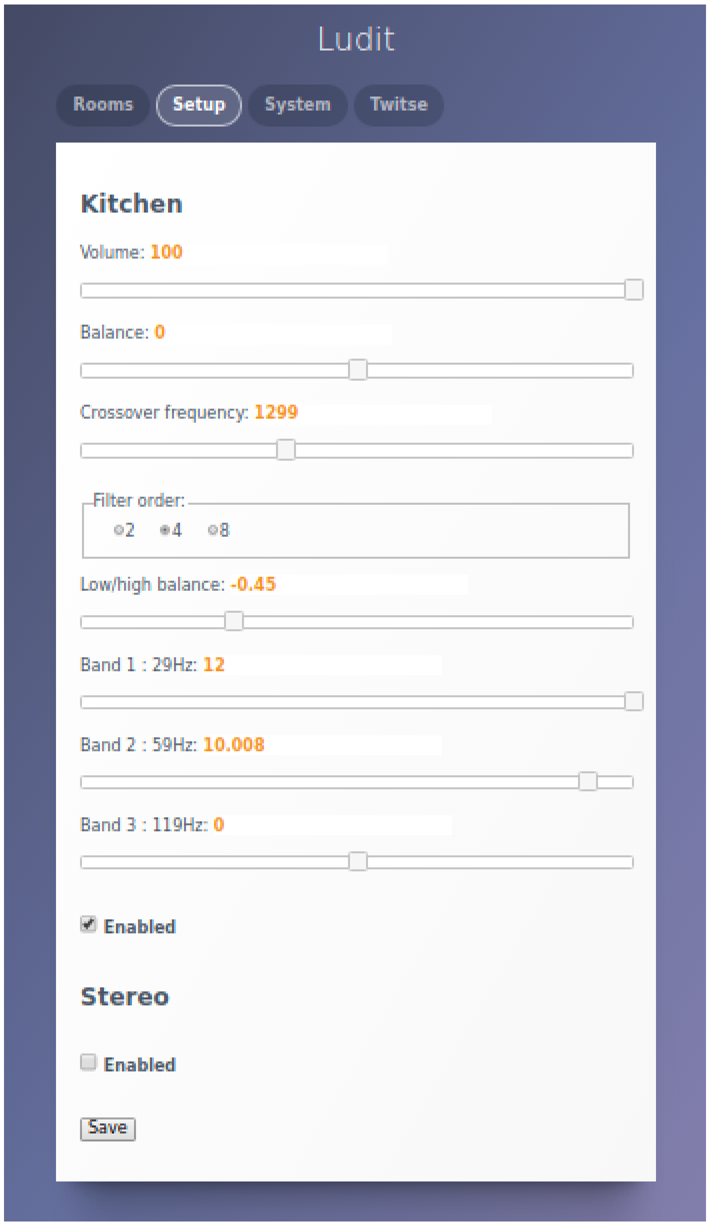
In the third and last terminal launch the following gstreamer pipeline. Note that the volume is turned way down to prevent audio shock. Increase it to actually hear anything:

```
gst-launch-1.0 audiotestsrc wave=pink-noise volume=0.01 is-live=true ! audioconvert !␣
↪audio/x-raw, channels=2 ! faac ! aacparse ! avmux_adts ! tcpclientsink host=
↪<hostname or ip> port=4665
```

On the PC audio output the woofer signal will be in one channel and the tweeter signal in the other. It will sound horrible. The audiotestsrc source can be replaced with a gstreamer source playing a local file or streaming web radio if the noise gets too much.

1.4 Web

Finally go to ./web and make a copy of ludit_local.js.template called ludit_local.js and fill in the correct ip or hostname. Then open index.html in a webbrowser and the first two tabs should be operational as they connect directly to the Ludit server.



Kitchen

Volume: 100

Balance: 0

Crossover frequency: 1299

Filter order: 2 [✓ 4] 8

Low/high balance: -0.45

Band 1 : 29Hz: 12

Band 2 : 59Hz: 10.008

Band 3 : 119Hz: 0

[✓] Enabled

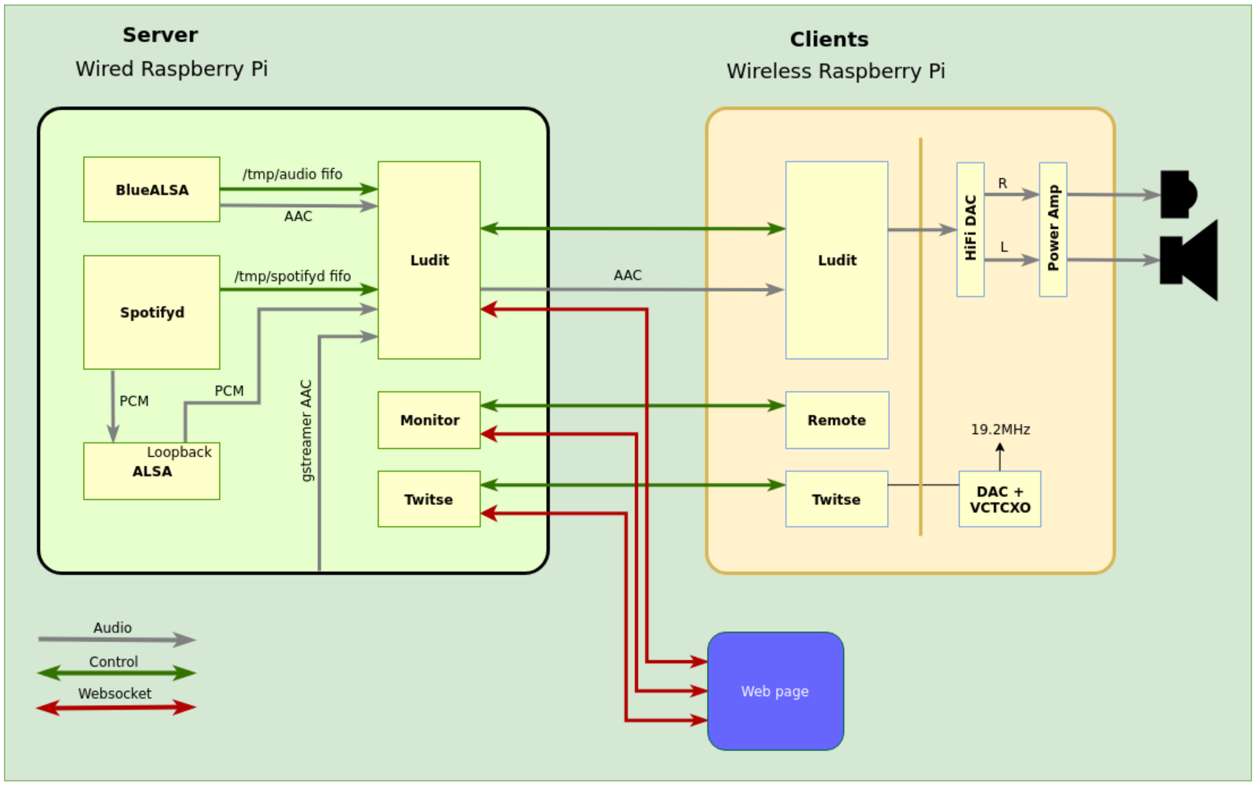
Stereo

[ ] Enabled

CHAPTER 2

System overview

Ludit is both the name of the audio player project on github but it is also more generally used to encompass the entire audio player system. The current complete Ludit audio system as it looks today is shown in the following image. The two main components are the server and the client dealing with routing and playing audio. There is a web page which connects to the server and is used for e.g. audio adjustments and selecting which groups are playing and which is not.



2.1 Server:

Pending…

2.2 Client:

The client shown in the yellowish square to the right in the image is a mono player so there will be two of these in a stereo group. Stuff running on the raspberry pi is shown to the left of the vertical line, and stuff on the right is located on the carrier board called Luhab which is part of the Twitse project. Besides a DAC + VCTCXO for controlling the clock to the raspberry pi the Luhab carrier also features a PCM5102A audio DAC.

2.3 Web:

Pending…

CHAPTER 3

Software installations

The package lists below are at least a starting point for installing the Ludit dependencies. On a computer never used for development before the lists will most likely not be exhaustive. Hopefully things will later crash and fail in a way that indicates what could be missing.

3.1 Common installs

These installs are common for both server and clients

Arch:

```
python-pip libfdk-aac faad2 faac kate git subversion openssh rsync gstreamer gst-
↪python gst-plugins-good gst-plugins-bad gst-plugins-ugly gst-libav python-pybluez␣
↪python-websockets
```

On a fresh install this will end up as nearly 1GB of storage used.

Ubuntu:

```
gstreamer1.0-libav gstreamer1.0-plugins-bad gstreamer1.0-plugins-ugly python-gst-1.0␣
↪ubuntu-restricted-extras aac-enc libfdk-aac-dev autoconf libtool libasound2␣
↪libasound2-dev bluez libbluetooth-dev glib-2.0-dev libgtk2.0-dev libsbc-dev libsbc1␣
↪python3-pip python3-websockets
```

3.1.1 Others

connectable installable via pip:

```
pip3 install connectable --upgrade
```

3.2 Client software installs

The client attempts to interact with gpio on a raspberrypi, to keep it from complaining on a raspberrypi install rpi-gpio python bindings. If rpi-gpio are unavailable (which they will always be on a non-rpi) then gpio will just be disabled.

3.3 Server software installs

simple-websocket-server can be installed via pip:

```
sudo pip install git+https://github.com/dpallot/simple-websocket-server.git
```

3.4 Known problems

Bluez exception using python 3.10 (seen on arch)

- SystemError: PY_SSIZE_T_CLEAN macro must be defined for '#' formats
- UnicodeDecodeError: 'utf-8' codec can't decode byte 0xff in position 4: invalid start byte

Install pybluez from Blaok fork (after uninstalling whatever pybluez might already be installed):

```
$ git clone https://github.com/Blaok/pybluez.git && cd pybluez
# python setup.py install
```

CHAPTER 4

Client types

There are two types of clients, single channel and stereo.

4.1 Single channel

Single channel clients are straightforward and simple. They come in pair of twos in order to play a stereo signal. They will play either the left or right channel and thus each client will need to know which channel to pick. A server group for two single channel clients could look like this:

Server configuration:

```
"devices": [
    {
        "channel": "left",
        "name": "leftdevicename"
    },
    {
        "channel": "right",
        "name": "rightdevicename"
    }
  ],
  "name": "groupname"
```

This is a part of the default json configuration that the server will print out with –newcfg.

The two clients in a group like this will call in with their names and the server will inform them about which channel to play. The straight forward way for each standard single channel client to identify itself is to supply a '–id group:name' on the command line which will use default settings for everything else. Alternatively they can specify a client configuration file instead in case the defaults won't do, see client configuration.

4.2 Stereo

It is possible to make a stereo speaker as well. The usecase will be a speaker that both runs as a low latency soundbar for a tv as well as being a normal streaming playing group. Channel will now be ‘stereo’.

Server configuration:

```
"devices": [
    {
      "channel": "stereo",
      "name": "devicename"
    }
  ],
  "name": "groupname"
```

As for the single channel a stereo client can be started with –id or by specifying a local configuration file with –cfg.

4.3 Client configuration

This is a template configuration for a client printed by a client with –newcfg:

```
"alsa": {
    "devices": [
        "hw:0",
        "hw:1"
    ]
},
"device": "devicename",
"group": "groupname",
"multicast": {
    "ip": "225.168.1.102",
    "port": "45655"
},
"version": "0.3"
```

The “alsa” part is optional, it can be used to override the system default alsa device if needed. The only case where it is required to be present and contain exactly two entries is for a stereo device using two separate soundcards. For a single channel device or a stereo device using e.g. a 5.1 surround soundcard it should only contain a single ‘devices’ entry if the default should be overwritten.

CHAPTER 5

Server audio source setup

The currently supported audio sources are

- spotifyd - Spotify over LAN/Wifi
- bluealsa - A2DP bluetooth sink
- gstreamer - Used for testing
- alsa - Alsa input on the server
- realtime - Local client in soundbar mode

5.1 Audio source: spotifyd

Spotifyd enables Ludit as an 'Spotify Connect' audio player in e.g. the Spotify list called 'Connect to device' seen by selecting 'Devices Available' during playing. There are ready made binaries for RPI, on x86 follow the upstream spotifyd instructions.

PCM audio from spotifyd is recorded with an Alsa loopback device. Install ./config/modules-load.d/raspberrypi.conf from the repository in /etc/modules.d on the server. This will make the snd-aloop kernel module load at boot which setups the loopback device. Check that the loopback device is device 1 and the onboard bcm2835 is device 0.

aplay -l:

```
card 0: ALSA [bcm2835 ALSA], device 0: bcm2835 ALSA [bcm2835 ALSA]
    Subdevices: 7/7
    Subdevice #0: subdevice #0
    Subdevice #1: subdevice #1
    .. etc
card 0: ALSA [bcm2835 ALSA], device 1: bcm2835 ALSA [bcm2835 IEC958/HDMI]
    Subdevices: 1/1
    Subdevice #0: subdevice #0
card 1: Loopback [Loopback], device 0: Loopback PCM [Loopback PCM]
    Subdevices: 8/8
```

(continues on next page)

(continued from previous page)

```
    Subdevice #0: subdevice #0
    Subdevice #1: subdevice #1
    Subdevice #2: subdevice #2
    .. etc
card 1: Loopback [Loopback], device 1: Loopback PCM [Loopback PCM]
    Subdevices: 8/8
    Subdevice #0: subdevice #0
    Subdevice #1: subdevice #1
    Subdevice #2: subdevice #2
    .. etc
```

For starting spotifyd manually have a look at the systemd file in ./systemd/ludit_spotifyd.service.template. It shows how to create a /tmp/spotifyd fifo and an example spotifyd launch line.

5.2 Audio source: BlueALSA

Use the fork here. Building instructions can be found on upstream.

Get bluetooth running as the first thing, including pairing and trusting the source devices.

For starting bluealsa manually have a look at the systemd file in ./systemd/bluealsa.service.template (in the bluealsa fork repository !). Like for spotifyd it shows how to create a /tmp/audio fifo and a typical spotifyd launch line.

If anonymous users should be able to play over bluetooth without any trusting or pairing then use a bluetooth autoconnect script like this one: (not tried this specific one but it looks nice) https://gist.github.com/oleq/24e09112b07464acbda1#file-a2dp-autoconnect

5.3 Audio source: Mopidy

It would be kind of rude not to add Mopidy as an audio source since Mopidy uses gstreamer and exposes its playing pipeline directly in its configurationfile. Mopidy plays just about everything but for Ludit integration it has only been tested with a standard MPD client. So the state of the Mopidy audio source in this project will realisticly be something like ‘under development’.

The Mopidy playing pipeline in ~/.config/mopidy/mopidy.conf should be changed to:

```
output = audioconvert ! audio/x-raw, channels=2 ! faac ! aacparse ! avmux_adts !␣
↪tcpclientsink host=<server> port=4666 sync=true
```

Mopidy sends general play state events on a websocket that Ludit needs to subscribe to. There are 4 configuration values in the Ludit configurationfile that needs to get adjusted:

```
mopidy_ws_enabled': 'true'
mopidy_ws_address': ip where Mopidy is running
mopidy_ws_port': the http port in the Mopidy configuration file
mopidy_gst_port': the tcpclientsink port in the Mopidy playing pipeline above
```

While developing Mopidy refused the webconnection from the Ludit server. A quick hack is to edit ‘handlers.py’ in the Mopidy sources. Edit the check_origin function to end with

> #if parsed_origin and parsed_origin not in allowed_origins: # logger.warn(‘HTTP request denied for Origin “%s”’, origin) # return False return True

For reference see https://github.com/mopidy/mopidy/pull/1712/commits/6e9ed9e8a9d4734671756ceeebf2059657ea2ab5. What the real fix is remains to be figured out.

5.4 Audio source : gstreamer

There can be any number of gstreamer inputs configured in the server configuration file. The "gstreamer" value is a list of inputs to be initialized and it is located under "sources". The only constraint is that all enabled gstreamer inputs need to have an unique port assigned.

An example of a single PCM input listening on port 4777:

```
"gstreamer": [
    {
        "enabled": "true",
        "format": "pcm",
        "port": "4777",
        "samplerate": "48000"
    }
],
```

From any LAN computer it is now possible to test if the above works with the following gstreamer pipeline:

```
gst-launch-1.0 audiotestsrc volume=0.01 ! audioconvert ! audio/x-raw, channels=2 ! /
audioresample ! audio/x-raw, format=S16LE, rate=48000 ! tcpclientsink host=<server IP>
↪ port=4777
```

The *Quick start* also uses a gstreamer input for testing out.

5.5 Audio source : alsa

Listens to an alsa input device on the server and uses a noise gate to automatically start and stop playing whenever there is a signal. This source is experimental.

Tip: To quickly check that there is indeed audio present on a given device (when nothing works), then arecord can act as a commandline vu meter:

arecord -f cd -d 0 -D hw:0 -vv /dev/null

5.6 Audio source : realtime

This is a rather convoluted audio source. The aim is to allow a client to run with minimal latency from a local input source and play it back locally as well. As such it is running against the spirit of Ludit as it for a start isn't really a Ludit audio source as it doesn't run on the server. It only plays locally on a single client and it is not broadcast over the network to other clients. The only usecase would be a stereo or a soundbar that should be able to both operate as a normal Ludit client with audio streamed from the server, but also play audio from a local video source in realtime. And even then it only makes sense if the audio processing in Ludit is truly needed due to e.g. preserve the workings of the crossover filter and/or any equalization filters. For playing a local realtime stereo signal the client should be configured as a stereo device which requires it to have two stereo alsa devices for 4 channel playback matching two two-way crossovers.

If a client is running in realtime mode its idle state is to listen for local audio and start playing if there is any. If the server starts streaming this will always have priority over the local audio and the local audio will only be able to resume when the server stops streaming. This is the simplest possible setup since it does not require the server to even know that there is a realtime client present.

If a client should run in realtime mode it has to be started with a local configuration file. The server can’t help with setting up a realtime client.

As for what the latency actually is then its okay for watching video. Purists requiring near zero latency (or better..) will most likely have left reading about Ludit by now anyway.

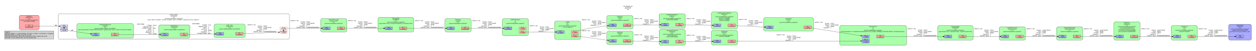
The automatic starting and stopping of local audio is done the same way as for the normal alsa audio source described above.

CHAPTER 6

Audio processing

6.1 Mono pipeline

The following image is generated by gstreamer from a running single channel pipeline (source in client/pipeline.py)



Sorry for the awfull single line layout.

CHAPTER 7

RPI setup

7.1 config.txt

Below is a starting point for a /boot/config.txt matching a Raspberry Pi 3 B+. Most importantly is enabling turbo mode to prevent the cpu from changing its frequency dynamically. This will have a negative effect on Twitse time measurements.

Next the RPI's are overclocked. They are not exactly speed kings and they can use a little boost. Common sense would dictate that overclocking must have a positive effect on the Twitse time measurements but this haven't really been verified.

/boot/config.txt:

```
# See /boot/overlays/README for all available options

gpu_mem=16

dtparam=i2c1=on            # client only
dtparam=i2c_arm=on         # client only
dtparam=spi=on             # client only
dtoverlay=hifiberry-dac    # client only

force_turbo=1
arm_freq=1200
core_freq=500
sdram_freq=500
over_voltage=2
over_voltage_sdram=2

initramfs initramfs-linux.img followkernel
```

/boot/cmdline.txt

Disable the 'audit' kernel logs: audit=0

Watch for processes writing to disk: iotop -o -b -d 10

(This is from an Arch installation)

CHAPTER 8

Introduction

Ludit is an audioplayer made for an wireless audio system using Twitse for time synchronisation. Ludit consists of a central wired server and seperate wireless stereo speakers where the server can act as e.g. a bluetooth A2DP headset for whatever music players that might run on whatever devices. Ludit does not have a music player app by itself.

Ludit requires the involved computers to be in hardware time sync since Ludit does not know how to correct audio with regard to crystal drift. The Twitse project is at least one way to keep computers in full hardware time sync over a WLAN.

Ludit is intended to get the best out of size constained loudspeakers in everyday setups. It is not too concerned about hifi but happily allows a heavy dose of electronic bass lift to get smaller loudspeakers play an acceptable bass on e.g. 3" or 4" bass drivers in 'small' (closed) enclosures. The price to pay is that Ludit speakers in such a setup are not really suited for playing insanely loud since the bass lift burns away quite some power used to battle the small enclosure without giving a equivivalent high sound pressure. But its nice to get the bass drum back. Obviously Ludit can just as well play through speakers that need no bass lift at all.

The following image shows one of two speakers made for the kitchen. It is very much in Ludits dna to play on two way systems where an electronic crossover can use right and left channel on a soundcard for tweeter and woofer. The carrier board underneath the raspberry pi is part of the twitse project although it sports a PCM5102A audio dac.

:alt: kitchen_speaker.jpg :width: 300px

CHAPTER 9

Links

The Raspberry Pi: Audio out through I2S. Analysis of the native I2S from a raspberry pi which happens to be rather jitterish as it struggles to produce a 44.1 kHz samplerate. Be aware when using syncroneous DACs:

http://www.dimdim.gr/2014/12/the-rasberry-pi-audio-out-through-i2s/