Liquidsoap 2.5.x

What's New

Liquidshop 2026

Agenda

1. Language quality of life
2. OCaml 5 and multicore
3. Streaming engine performance
4. JACK rewrite
5. Subtitles
6. New switching paradigm
7. Native Icecast server
8. The Liquidsoap book

1. Language Quality of Life

Implicit int → float

Works at **call sites** where the parameter type is known to be `float`:

```
fade.in(duration=3, s)        # was: duration=3.
sin(0)                        # was: sin(0.)
random.float(min=0, max=10)
```

Does **not** work where the compiler needs to unify types across branches:

```
if b then 1. else 2 end   # error — branches must agree
[1., 2, 3.]               # error — list elements must agree
```

When in doubt, `float(x)` still works everywhere.

`getter(t)` and `ref(t)` type annotations

Previously there was **no syntax** to explicitly annotate getter or ref types.

This mattered when the typechecker couldn't infer what you intended:

```
# Force a value to be treated as a getter:
volume = (0.8 : getter(float))

# Force a method to be treated as a ref:
state = (x.foo : ref(int))
```

When the typechecker can't figure it out, you can now tell it directly.

Better runtime errors

`Invalid_value` exceptions now include a full call stack:

```
At script.liq, line 5, char 4-22:
s = single("test.mp3")

Error 7: Invalid value:
That source is fallible.
This value was passed through the following call stack:
at script.liq, line 2, char 2-33,
at script.liq, line 6, char 0-12
```

Debugging scripts becomes easier.

2. OCaml 5 and Multicore

Tracking OCaml 5 GC regressions

Liquidsoap currently runs on OCaml 4.14. Switching to OCaml 5.x (multicore) has been blocked by **GC pacing regressions**: the garbage collector's default parameters caused significantly higher CPU and memory usage compared to 4.14 for our workload.

We reported and tracked two issues:

- ocaml/ocaml#13123 — initial regression report (4.14 → 5.1)
- ocaml/ocaml#14533 — pacing behaviour in 5.5+

The good news: pacing appears fixed in OCaml 5.5+. Default parameters may still need tuning — we're following up. An excellent talk on the broader multicore debugging effort: The Saga of Multicore OCaml (Jane Street).

What multicore means for Liquidsoap

Liquidsoap's streaming pipeline is largely **sequential by nature**: decode a frame, pass it through operators one by one, send it to outputs. There is not much to parallelize there.

The real CPU bottleneck is inside **decoders and encoders** — and for those, parallelism already happens inside FFmpeg itself.

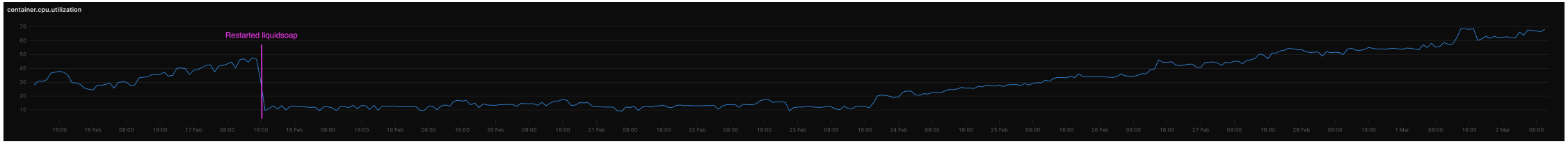
What multicore OCaml does enable is running **independent clocks and outputs in parallel** — multiple streams, multiple encoding pipelines — without threads competing under a single lock. That's the gain we're after.

3. Streaming Engine Performance

(Also backported to the next 2.4.x stable release)

It started with a bug report

A user reported **CPU slowly climbing over days until it hit 100% of a core**. Through a patient back-and-forth they helped us gather the right data: reproduction script, OS-level graphs, memtrace output, Linux perf reports on the compiled binary.



Memtrace showed no memory leak. The culprit was in the logs:

```
[clock:2] There are currently 104 sources, possible source leak!
```

Each crossfade transition created new `source.dynamic` sub-sources that were **never released**. After thousands of transitions, the clock was iterating over thousands of stale entries on every tick.

The fix — and then some

Sub-clock leak on source deregistration was fixed. Linux perf reports then revealed further hot-path inefficiencies:

- Clock sync-source propagation: **O(n) → O(1)** per tick
- Frame length computation: pre-computed instead of recalculated each tick
- Format/kind dispatch: int-indexed array instead of queue scan

CPU is now stable. No script changes required.

3. JACK Rewrite

What changed

Complete rewrite using native OCaml C bindings. No more `bjack` dependency.

JACK now integrates with the same sync-source clock mechanism as ALSA, SRT, and PulseAudio. The JACK processing thread directly drives Liquidsoap's clock, giving true hardware-timed streaming.

JACK operates at ultra-low latency, well below Liquidsoap's default 20ms frame — a good opportunity to push the new clock architecture to its limits.

5. Subtitles

Subtitles as a first-class content type

Subtitles join **audio** and **video** as a native Liquidsoap content type.

- Native SRT decoding
- FFmpeg encode/decode: `%subtitle`, `%subtitle.copy` (passthrough)
- `on_subtitle` — react to subtitle events in a script
- `subtitles.map` — transform subtitles
- `subtitles.insert` — dynamically inject subtitles into a stream

Why it matters beyond "subtitles"

Content model parity: everything FFmpeg can carry, Liquidsoap can now express and handle natively.

Live accessibility: receive a stream, inject auto-generated captions via `subtitles.insert`, serve it — all in one script.

```
s = input.harbor("live")
s = subtitles.insert(s)
# inject a subtitle at any time:
s.insert_subtitle({text="Hello!", duration=3., forced=false, format="text"})
output.icecast(%ffmpeg(..., %subtitle), host="...", s)
```

6. New Switching Paradigm

Per-source composition methods

(Open PR — feedback wanted!)

The old pain

Every radio operator has fought this:

```
# The old way — fragile, non-obvious
radio = fallback(
  track_sensitive=false,   # needed for live sources, but applies to ALL
  transitions=[
    fun (_, b) -> fade.in(b),   # must manually enumerate
    fun (a, _) -> fade.out(a),
    fun (a, b) -> sequence([fade.out(a), fade.in(b)])
  ],
  [live, music, backup]
)
```

Flat parameters that apply to all sources equally. Easy to get wrong, hard to explain.

The new model

Each source knows what it is and carries its own switching intent:

```
live  = input.harbor("live")       # composition_type = "live"
music = playlist("/path/to/music") # composition_type = "file"

# Live cuts in immediately.
# Music waits for a track boundary.
# Fades handled automatically.
radio = fallback([live, music, backup])
```

This just works.

Per-source overrides

Full control when you need it, attached to the source:

```
# Force a specific source to always wait for track boundary
live.track_sensitive := true

# React when this source is selected or leaves
live.on_select(fun ({starting}) ->
  fade.in(starting)
)

music.on_leave(fun ({source = s}) -> source.skip(s))
```

Behavior is attached to the source, not scattered across the operator call.

Why this is a big deal

- The most common newcomer confusion — "why doesn't my live source cut in immediately?" — **goes away by default**
- Per-source `on_select` / `on_leave` callbacks enable fine-grained automation without fragile transition lists
- The model matches how radio operators actually think: *this source is live, that one is a file*

This is in active review. Does this match how you think about switching? Are there cases this doesn't cover?

7. Native Icecast Server

`icecast.server`

(Live demo — beta, ready for testing at scale)

The simplest setup

```
icecast.server(port=8000, password="hackme")
```

Liquidsoap listens on port 8000. DJs point Butt, Mixxx, or any Icecast-compatible client at it directly.

No separate Icecast daemon required.

Migrating from an existing Icecast setup

Point at your existing XML config:

```
icecast.server(config="/etc/icecast.xml")
```

XML configuration support

Status	Options
✅	`listen-socket`, `source-password`, `burst-size`, `burst-on-connect`, `source-timeout`, `client-timeout`, `tls-certificate`, `tls-key`, `mount`
⚠️ parsed, not enforced	`limits/clients`
❌ not yet	`max-listeners`, `hidden`, `public`, `intro`, `relay`, `directory`, `fileserve`, `logging`
🚫 incompatible	`fallback-override`, `fallback-when-full`

React to source connections

```
server = icecast.server(port=8000, password="hackme")

server.on_connect(fun (r) ->
  log.important("DJ connected on #{r.mount}")
)

server.on_disconnect(fun (r) ->
  log.important("DJ disconnected from #{r.mount}")
)
```

Full scripting power available on every mount event.

Why it's architecturally different from Icecast

Classic Icecast is a relay — it passes bytes. When a live source disconnects, listeners get an HTTP redirect to the fallback mount. Some players handle it. Many don't. Everyone notices.

Liquidsoap's server: fallback lives *inside the source pipeline*. The output — and the listeners — never change. No redirect. No disconnect.

```
DJ disconnects → source.dynamic switches inside → listeners hear nothing
```

The game changer: `dedicated_encoder`

With a shared encoder (default), listeners who connect mid-stream get the stream at whatever point the encoder happens to be. With format-sensitive containers, late joiners often get garbled or broken playback.

```
icecast.server(
  port=8000,
  password="hackme",
  dedicated_encoder=true
)
```

What `dedicated_encoder` does

Each listener that connects gets their own encoder instance, initialized from a **clean frame boundary**.

Every listener gets a properly-framed stream from byte 0, regardless of when they join.

With copy-based encoders (`%ffmpeg` with `%audio.copy` / `%video.copy`): the cost is one lightweight remux per listener. The listener experience difference is significant. Scalability under large listener counts still needs validation.

`dedicated_encoder` in context

	Default	`dedicated_encoder=true`
Encoder instances	1 shared	1 per listener
Late joiners	mid-stream, may glitch	clean from byte 0
CPU cost (copy encoder)	baseline	minimal overhead
CPU cost (full re-encode)	baseline	high — avoid

Use copy encoders (`%audio.copy` , `%video.copy`) with `dedicated_encoder=true` .

The full picture

```
# One line to replace your Icecast daemon
server = icecast.server(
  port=8000,
  config="/etc/icecast.xml",
  dedicated_encoder=true
)

# Full Liquidsoap scripting on every mount
server.on_connect(fun (r) -> log.important("#{r.mount}: DJ on air"))
server.on_disconnect(fun (r) -> log.important("#{r.mount}: DJ left"))
```

Incoming stream → demux → Liquidsoap pipeline (fallbacks, metadata, subtitles, …) → remux → listeners, each with a clean stream.

The Liquidsoap Book

A new edition of the book

2.5.x brings enough new features and breaking changes that we're targeting a **new edition of the book** to release alongside it.

The Liquidsoap book is the main learning resource for new users — getting it right for 2.5.x matters.

Still missing: Icecast server, subtitles, per-source composition, source introspection. Clock model partially updated.

How to contribute

The book source is on GitHub: **savonet/book**

We mainly need **readers**:

- Read a chapter, try the scripts, tell us what breaks
- Tell us what doesn't make sense or what's missing
- Open an issue — you don't need to write a fix

If you've struggled to understand something, that's exactly the feedback we need.

Summary

Feature	Status
Language QoL (int/float, errors)	✅ 2.5.0
Streaming engine performance	✅ 2.5.0 + 2.4.x
JACK rewrite	✅ 2.5.0
Subtitles	✅ 2.5.0
Per-source composition	🔄 In review
Native Icecast server	✅ 2.5.0 (beta)
OCaml 5 / multicore	🔄 Tracking upstream
New book edition	🔄 In progress

Thank you

Questions?