Building a video conference (WebRTC) controller with Elixir

Or how Elixir was introduced into VoiSmart

Hello!

- ➔ **Who:**
  Matteo Brancaleoni
- ➔ **Where:**
  VoiSmart www.voismart.it
- ➔ **As:**
  Software Engineer
- ➔ **Where #2:**
  GH: @xadhoom
  TW: @mbrancaleoni
  E: mbrancaleoni@voismart.it

About us

15 years into the VoIP industry, building PBX and voice termination turnkey solutions using open source softswitches, like Asterisk, FreeSwitch.

Mostly coded with Python and the Twisted Networking engine, doing multi process dialog & coordination with RabbitMQ.

In 2017 we jumped into the video conference bandwagon.

Telco is hard...

- ➔ **Many concurrent tasks running**
  Call control, logging, device states, …
- ➔ **Many identical tasks running**
  Without shared data between them
- ➔ **Events everywhere**
  Each call is very chatty about what is doing

Telco is realtime...

- ➔ **You cannot wait to handle a call**
  Everything served in few milliseconds
- ➔ **Async, async, async!**
  A phone system is async, several things may happen when you're serving a call
- ➔ **Python is not the right tool here**
  But using Twisted and RabbitMQ helped a lot in being async and distributing work

A video conference system has the same requirements ?

Yes! It does :)

Let’s see why and how we handled it, better.

Requirements

➔ **Web based conference**
Because our roots are in the www

➔ **Should carry video**
It's a *video* conference

➔ **Should carry audio**
And should connect the PSTN

➔ **Should carry chat messages**
For quick text snippets sharing

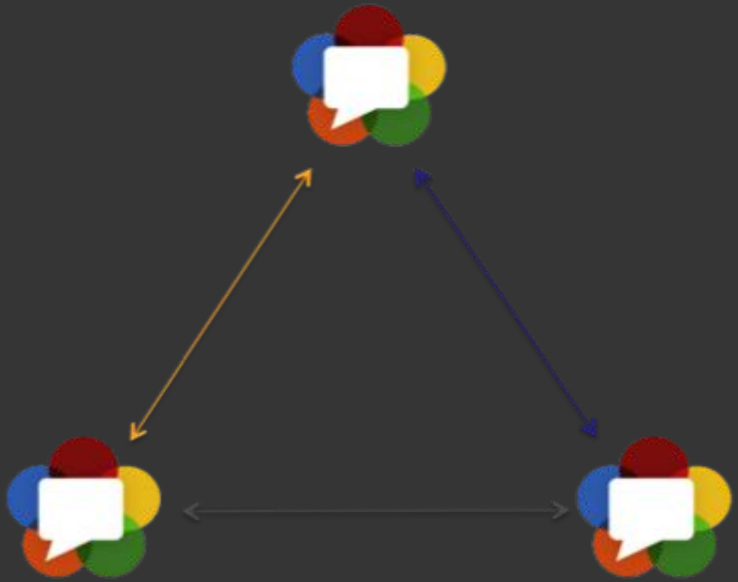
WebRTC

Simple APIs for rich
Real Time multimedia communications.

But signalling is out of scope for WebRTC.

(so just supporting webRTC means almost nothing…)

Mesh



Pros

Lowest delay

Best compatibility

Cons

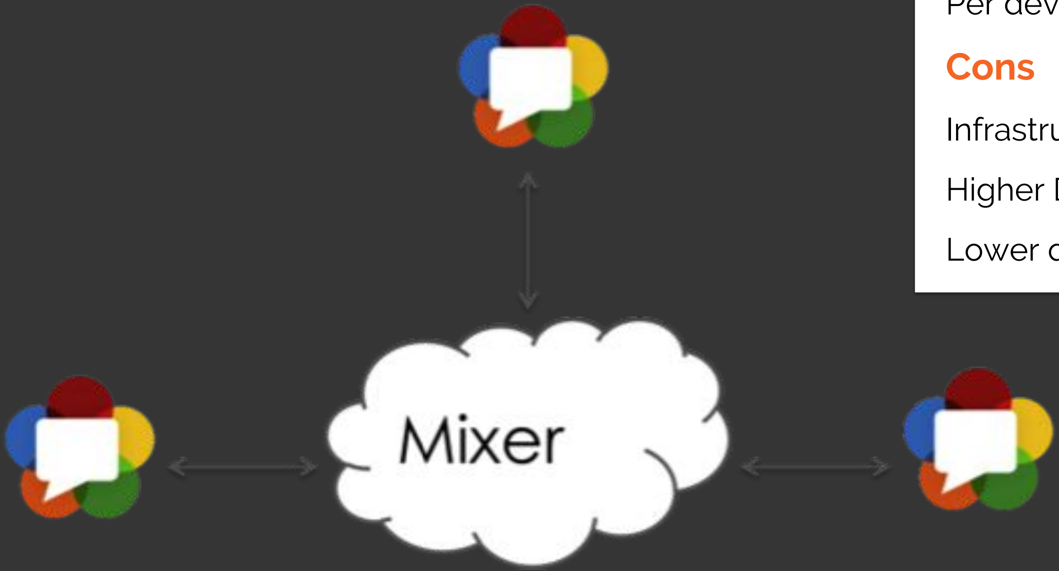
Does not scale

High CPU usage

High bandwidth

Image courtesy of https://webrtchacks.com

MCU



Pros

Low bandwidth

Per device stream

Cons

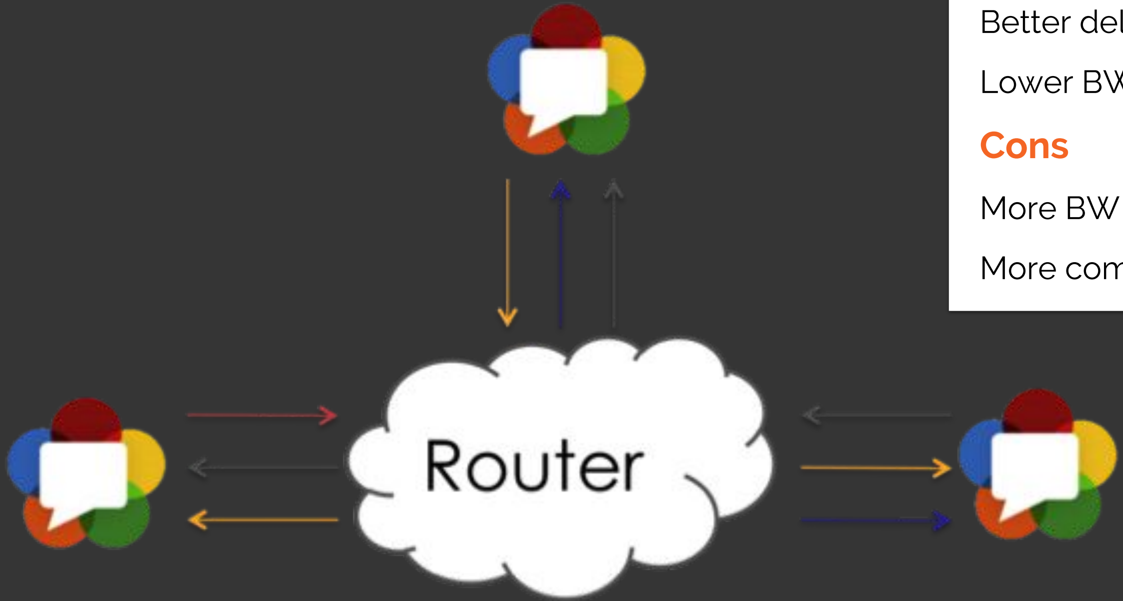
Infrastructure cost

Higher Delay

Lower quality

Image courtesy of https://webrtchacks.com

SFU



Pros

Scalable

Better delay

Lower BW than Mesh

Cons

More BW than MCU

More complex clients

Image courtesy of https://webrtchacks.com

Signalling

A way to communicate Session Descriptions to others

You have to build your own protocol and transports

Signalling #2

Not only webRTC stuff, but also:

Instant messages

Audio events

Call control (kick, mute, join, quit…)

Signalling #3

Not only client 2 server, but also inside the server:

Controlling the SFU (janus)

Controlling the audio bridge (freeSwitch)

Signalling between clients

this is Realtime stuff

Continuous flow of asynchronous events, from multiple sources, that must be handled concurrently

Things may also fail

- Lost connection to the clients (common)
- Lost connection to the media servers
- webRTC is still in development, a client may crash us
- Programming error :(

Tools needed for the job

- ➔ **A tool to ease async message passing between functional units**
- ➔ **Every unit may do “something” while waiting for events**
  So the tool should allow me to create live “loops” easily
- ➔ **Delays on some dialogs should not slow things down**
- ➔ **Issues on some units should not crash the system**

Elixir to the rescue!

Because is Erlang after all

Our architecture

Or what we learned so far

Use umbrella apps

Split your services in separate apps to ease development

- One for messages
- One for video
- One for audio
- One for auth
- One for logging
- And web, and config APIs, etc

Log everything

With a correlation ID

And a machine parsable format

Authorization & Authentication

JWT between apps, using :joken

JWTs contains all {A,M,F} tuples for the logged user

Zuul, a service to handle users/permission

Each service registers/updates own permissions
via Zuul on app start

Roles handling

Group permission into roles

Delegated to a subapp, depends on the others

On system start, registers/updates roles via Zuul

defguarded

A macro that checks JWT tokens

Put your claims into the token and let the macro check if you can call the function

(a near miss with elixir 1.6 defguard/defguardp)
(split entry point with real implementation or tests will suffer)

The token cache/opaquer

A revoking mech for JWTs

JWTs can also be long

(Putting them into Authorization header may become an issue)

Map them with an opaque string

Born with in-memory storage, easily migrated to Ecto,
because it is a GenServer and the public APIs did not change

—

Client communications

:phoenix websockets for live events and room protocol

One channel dispatching to several modules

GraphQL APIs for the rest, with :absinthe

(basically, only 2 endpoints for everything)

Identify your processes

Create a process only if needed

(they're async, you should handle state, messages and crashes,
so do that only if really needed)

Most of the time you can pass some data to a module and keep it in the calling process.

Processes are useful

For long running tasks (e.g. token expire)

For persistent connections

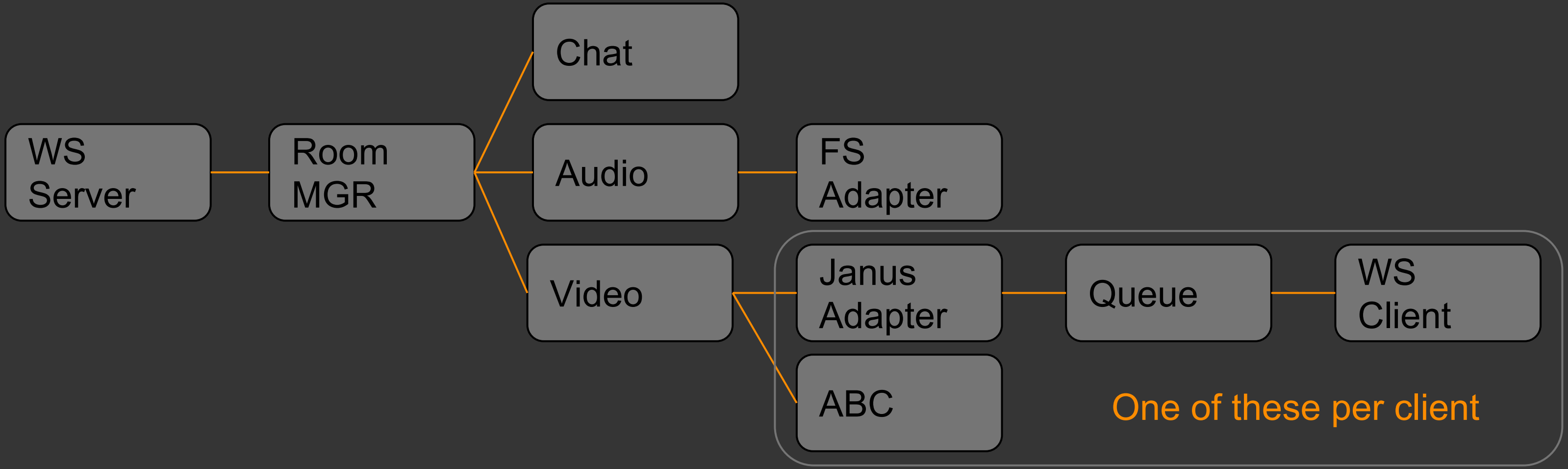
To keep temporary state (which may be used later/shared)

To create a controlled crash chain

To handle disconnections
(client disconnects does not close the room and notifies others)

For out-of-band processing (Adaptive Bitrate Controller)

how a room looks like



Going :global

As the old adage goes
"Always wrap process lookup in a module"

Moved from Registry to :global in few hours,

Distributed and fault tolerant Video Conference!

Testing this stuff

"Mock as a noun"

we chat a lot with external entities, so the need of mocking modules that mimic the behaviour in normal and error conditions

Testing this stuff #2

Aka (partial) integration tests

Because services are not isolated when deployed

An umbrella app for integration tests

It depends on all the others, started only in :test env

Testing this stuff #3

Continuous delivery to the instance
we use everyday for ourself and for demos

Using :edeliver in GitLab pipelines

Deploying

We don’t do services, only ship software packages

Packaged as RPM with Koji (Fedora build server)

Release built with :distillery, :conform and included runtime

Building in a disconnected env is problematic for Elixir

Upload already fetched deps, rebar(3) and hex to build it

Future?

A SIP bridge to let SIP video calls interact with the SFU based webRTC

(mixing video, on the fly layouts and so on…)

Session Recorder

Using :ffmpex, a nice frontend to ffmpeg in elixir

OSS Contributions

:ecto unsafe_fragment/1 by @xadhoom
:swoosh 2 feature PRs and 2 fixes PRs by @davec82/@xadhoom
:event_socket_outbound new package by @davec82
:elixir_mod_event features and fixes by @davec82/@xadhoom
:ffmpex improvements by @xadhoom
dialyzer fixes :phoenix, :gen_state_machine by @flaviogrossi
:websocket_client reconnection fixes by @flaviogrossi
Janus SSL for rabbitMQ connections by @flaviogrossi

Open Telecom Platform

Everything into a Telco application fits into what Erlang provides. A feeling hard to explain, but strong.

By extension also Elixir, with a nicer syntax and a lot of higher level libs that ease development.

Demo time!

Thank you.

Made with love at VoiSmart by
Matteo Brancaleoni @xadhoom
Davide Colombo @davec82
Flavio Grossi @flaviogrossi
Maybe You ? We're also hiring :)

Questions?