Arching Kaos

Kaotisk Hund

April 6, 2021

Abstract

Little words about what follows... arching ... kaos !!!

Contents

Chapter 1

Introduction

Chapter 2

Theory

In following, we will overview some basic concepts such as radiowaves, what they are and how the are transmitted, radio stations through history, which was their role and which is this today, what do they need to operate and some theoritical organisation models of a hypothetical, commercial or not, radio station.

2.1 Radiowaves

Radio waves are radiated by electric charges undergoing acceleration[**first**].[1]. They are generated artificially by time varying electric currents, consisting of electrons flowing back and forth in a metal conductor called an antenna,[6][7] thus accelerating. In transmission, a transmitter generates an alternating current of radio frequency which is applied to an antenna. The antenna radiates the power in the current as radio waves. When the waves strike the antenna of a radio receiver, they push the electrons in the metal back and forth, inducing a tiny alternating current. The radio receiver connected to the receiving antenna detects this oscillating current and amplifies it.

As they travel farther from the transmitting antenna, radio waves spread out so their signal strength (intensity in watts per square meter) decreases, so radio transmissions can only be received within a limited range of the transmitter, the distance depending on the transmitter power, antenna ra-

[1]Kraus, John D. (1988). Antennas, 2nd Ed. Tata-McGraw Hill. p. 50. ISBN 0074632191.; Serway, Raymond; Faughn, Jerry; Vuille, Chris (2008). College Physics, 8th Ed. Cengage Learning. p. 714. ISBN 978-0495386933.

diation pattern, receiver sensitivity, noise level, and presence of obstructions between transmitter and receiver. An omnidirectional antenna transmits or receives radio waves in all directions, while a directional antenna or high gain antenna transmits radio waves in a beam in a particular direction, or receives waves from only one direction.

Radio waves travel through a vacuum at the speed of light, and in air at very close to the speed of light, so the wavelength of a radio wave, the distance in meters between adjacent crests of the wave, is inversely proportional to its frequency.

The other types of electromagnetic waves besides radio waves; infrared, visible light, ultraviolet, X-rays and gamma rays, are also able to carry information and be used for communication. The wide use of radio waves for telecommunication is mainly due to their desirable propagation properties stemming from their large wavelength.[7] Radio waves have the ability to pass through the atmosphere, foliage, and most building materials, and by diffraction can bend around obstructions, and unlike other electromagnetic waves they tend to be scattered rather than absorbed by objects larger than their wavelength.

Τα ραδιοκύματα ανακαλύφθηκαν από τον ... το ... με [...] (κάποιον τρόπο. Τα πρώτα πειράματα που έγιναν ήταν σχετικά με την [...][2].

2.2 Radio Stations

A radio broadcasting station is usually associated with wireless transmission, though in practice broadcasting transmission (sound and television) take place using both wires and radio waves. The point of this is that anyone with the appropriate receiving technology can receive the broadcast.[17] Use of a sound broadcasting station

In line to ITU Radio Regulations (article1.61) each broadcasting station shall be classified by the service in which it operates permanently or temporarily.

[2]πηγή εδώ

2.2.1 History

2.2.2 Operation

2.2.3 Work Models

2.2.4 Today

2.3 Music

Chapter 3

Into the era of digital technology

3.1 Some stuff

3.2 Internet

3.3 Collaborative management

Chapter 4

arching-kaos-docs

Documentation for installing and operating arching-kaos

4.1 Description

Arching Kaos is a decentralized radio station.

4.2 Features

Among with a stream of mixes, there are some features that Arching Kaos provides.

These are:

- installing and operating a web radio station,
- an API for uploading mixtape-like content,
- listing of all radio mixtapes
- registering uploaders,
- a web site that renders based on live information through the API,
- an IRC network, that also appears on the radio's website,
- a decentralized sosial network for sharing the latest uploads and/or news and

- decentralized filesystem storage for sharing and helping each other on file hosting.

4.3 Repository usage

This is where the stream of changes are stored. You can clone them using `git` to your local filesystem like this:

```
git clone <url>
```

Edit install.sh to match your preferences. There is a couple of variables being set there which are replaced by some default values. Most of the values can work from the default with no editing of the file, but it's good to have a look at it before running it anyway.

Run `./install.sh` to configure, install and start `arching-kaos`.

4.4 How it works?

We can find out how it works but mostly we need to know what the problems are that we are going to solve.

4.4.1 Problems to solve

1. 24/7 music

For that, we play all the mixtapes from a playlist in repeat, scanning regularly for content.

2. Multiple contributors

In a radio station and for a quality time and stream of music, it's needed to have producers that they produce the content. So, we need to be able to accept people's contributions to the radio station, in order to play them and further share them.

3. File storage

As people host their files into their computers, so does a server. We can utilize the existence of a file in two places or more places with p2p technology. (See 4. File Sharing)

4. File sharing

As a matter of fact, p2p technology is used for hosting information on multiple peers which can be synchronized through the network. These include IPFS, the InterPlanetery FileSystem which helps us on transfering files big files with no limits, works as p2p so our downloads can't fail as it rechecks each chunk of the file for its consistency and provides an HTTP gateway for accessing files.

5. Point of view

Another point is how the radio station and the so formentioned mixtapes are going to make their way into publicity! So, apparently, since we speak for a web radio, a web page would be perfect for that.

6. Communications

Music is about communication. A radio too. How people reach each other? A network of users could be established for exchanging messages. So far, IRC does a pretty good job on this. So we 'll be making use of an IRC server in order to exchange instant messages.

7. Socialization

Instant messaging is enough for getting information "in the moment", but what about the updates of the station? There is a need for a place that people can be informed about the latest mixtapes and the news of the radio station.

8. Pack all these

Since, so far we made a list of 7 problems, of course a solution to all these by one package is a matter of consideration. Organizing and optimizing different components and ensure that will work similarly on different environments is a priority. So we make an image of each component and run containers with docker.

4.4.2 Quick overview

1. It's based on icecast2 server to produce a stream.

We just initialize a very simple but configured from our `./install.sh` script Icecast v2 docker image.

2. Uses IPFS to upload, provide links and sync mixes.

We use IPFS as a gateway to retrieve IPFS links to the local File System of our working environment. We need to save in order to provide a "safe source" for Liquidsoap (See 8. Liquidsoap). We also pin the links to our IPFS repository.

3. Uses DAT and Torrent to produce sync options.

Whenever a mixtape is synced to the radio station, a DAT repository (and a torrent should but not yet) is created and stored in the mixtape list.

4. Uses SSB to produce newsletter on SSB network.

Whenever a mixtape is synced, SSB network is informed about it's artist, title and IPFS link to listen, in #arching-kaos-mixes.

5. Uses ReactJS to produce a webpage for the radio station.

For creating a web interface, ReactJS framework was used, along with some external libraries, like `node-fetch` to fetch statistics (artist and title) for the currently playing mixtape on the streaming radio station and for creating a list of all the mixtapes to listen on demand.

6. Uses expressJS to create and run an API that holds a list of shows and an IP whitelist for authenticating uploaders.

ExpressJS is a nodeJS library that helps us create an HTTP server instance which can be manually routed. What I do in there, is providing routes that activate certain procedures. We use this for registering uploaders, upload mixtapes and provide the mixtape list to whoever applies for it.

7. Uses CJDNS network for unique IP per uploader.

As most of the devices connected to the internet are mostly routed through a NAT network, it's not possible to have different uploaders registered through their IPs. CJDNS creates a p2p mesh network, which is based on public/private key encryption. Every peer (user of this program) has a unique cryptographic key pair which consists of a public and a private key. The private one is used to sign outgoing packets and decrypt incoming ones, while the public one is used to encrypt packets for the particular peer. The public key is also transformed to a private IPv6 on fc00::/7 block which can be then used for actually send and receive packets from and to the network.

8. Uses Liquidsoap to provide inputs to icecast2.

We already have a stream server, icecast v2, set up, waiting to server streams to clients. But actually, there are no streams. What we are going to about it, is use Liquidsoap for playing a playlist containing the mixtapes, which is regularly refreshed and reload, to icecast's specific mount point so listeners can reach to listen to.

9. Uses charybdis-ircd for communication.

Many different IRC daemons/servers were investigated to see what fits best, in terms of connectivity, maintainance and the ability to make a network of IRC servers. I choose charybdis-ircd for being the most complete IRCv3. It can also connect to other charybdis-ircd instances which is good option for creating a network of IRC servers to prevent downtime and centrality problems.

4.5 Environment preparation

The final testing got part on a Gentoo Linux, a Linux distribution by Gentoo Foundation Inc, but is easily adjusted to Fedora Linux which is sponsored by Red Hat Inc. Ubuntu Linux which is sponsored by Canonical Inc, is compatible as well, but most of the automatic tools are bundled with Fedora's package manager, `dnf`.

4.5.1 Off to go!

We are on a freshly installed Fedora Linux, that's not a requirement, it can be any other distribution but I 'll use this as a reference.

We need to install basic packages to get the tools will use later. These three things, `cjdns`, `docker-ce`, `docker-compose`, `git`, `bison`, `automake`, `libtool`, `m4` and `nginx`.

#TODO Make sure there is a script for all these, at least on Fedora, plus, remember what `bison` was needed for... charybdis?

After installing these, we need to clone this repository, if not already cloned, and change directory to it

```
git clone https://git.arching-kaos.net/kaotisk/arching-kaos.git
cd arching{-}kaos
```

The installation makes a directory `./storage` with all the data.

Note: both the projects' files and data files are used. The `./storage` folder is excluded from the repository but not `./etc`. If you are contributing be sure not to publish passwords.

4.6 How the installation works

After cloning the repository, you can review the `./install.sh` file, in order to configure basic parts of arching-kaos.

We will now go into a deep analysis of the `./install.sh` script and see what it does and also explain the values that can be edited in there.

We start with a small greeting to the user:

```
#!/bin/sh
echo "Getting the basics done..."
echo "Initializing and updating modules..."
```

4.6.1 Submodules initiation

In the very first part of `./install.sh` we just pull the git modules that we want to include in our project. To achieve that, we change our working directory to `./modules` where the modules are supposed to get downloaded.

```
cd modules
```

We are going to start with `arching-kaos-api`.

```
git submodule init arching-kaos-api
git submodule update arching-kaos-api
```

We continue with `arching-kaos-radio`.

```
git submodule init arching-kaos-radio
git submodule update arching-kaos-radio
```

Add the `arching-kaos-generic`.

```
git submodule init arching-kaos-generic
git submodule update arching-kaos-generic
```

Also the `arching-kaos-ssb`.

```
git submodule init arching-kaos-ssb
git submodule update arching-kaos-ssb
```

The `arching-kaos-irc`.

```
git submodule init arching-kaos-irc
git submodule update arching-kaos-irc
```

And the `docker-dat-store`

```
git submodule init docker-dat-store
git submodule update docker-dat-store
```

We inform that we are done with the above

```
echo "Done!"
```

And we change back to our root directory (project's one).

```
cd ..
```

4.6.2 Configuring ./etc

Now, what we are going to do is simply replace variables placed in various files across the project to achieve correct settings before we run anything.

```
echo "Configuring ./etc ..."
```

IRC

We start with charybdis IRC daemon settings. Please adjust to your needs.

```
echo "...1/4 charybdis"
```

There are certain settings that need to be configured in charybdis. Our main approach is to have everything ready to get into a working state. In `./install.sh` the following blocks from `./etc/charybdis/ircd.conf` are going to be edited. Comments and defaults are stripped. See actual file for more information and help.

- Server information

```
serverinfo {
  name = "{$IRC_NAME}";
  sid = "{$IRC_SID}";
  description = "{$IRC_DESCRIPTION}";
  network_name = "{$IRC_NETNAME}";
  vhost = "{$PUBLIC_IPV4}";
  # In case you want to use a public IPv6 you may want
  # to uncomment the following line and also set an IPv6
  # in the appropriate sed command.
  #vhost6 = "{$PUBLIC_IPV6}";
  vhost6 = "{$CJDNS_IPV6}";
};
```

- Administrator information

```
admin {
  name = "{$ADMIN_NAME}";
  description = "{$ADMIN_DESCRIPTION}";
  email = "{$ADMIN_EMAIL}";
};
```

- Listening settings

```
listen {
  host = "{$PUBLIC_IPV4}";
  sslport = 6697;
  host = "{$CJDNS_IPV6}";
  port = 6667;
};
```

- Operator settings

```
auth {
  password = "{$IRC_AUTH_PASSWORD}";
  class = "opers";
};
```

- Administrator settings

```
operator "god" {
  user = "*god@{$CJDNS_IPV6}";
  password = "{$GOD_IRC_PASSWORD}";
  snomask = "+Zbfkrsuy";
  flags = ~encrypted;
  privset = "admin";
};
```

We use `sed` to set the variables ({$variables}) to values.

```
sed -i.bak -e 's/{$IRC_NAME}/irc.arching-kaos.net/' \
  etc/charybdis/ircd.conf
sed -i.bak -e 's/{$IRC_SID}/44Q/' etc/charybdis/ircd.conf
sed -i.bak -e 's/{$IRC_DESCRIPTION}/A friendly IRC server/' \
  etc/charybdis/ircd.conf
sed -i.bak -e 's/{$IRC_NETNAME}/irc.arching-kaos.net/' \
  etc/charybdis/ircd.conf
sed -i.bak -e 's/{$PUBLIC_IPV4}/127.0.0.1/g' \
  etc/charybdis/ircd.conf
# In case you have a public IPv6 you should just insert
# it between the double slashes.
# sed -i.bak -e 's/{$PUBLIC_IPV6}//g' etc/charybdis/ircd.conf
sed -i.bak \
  -e \
  's/{$CJDNS_IPV6}/fc42:7cfa:b830:e988:f192:717f:6576:ed12/g' \
  etc/charybdis/ircd.conf
sed -i.bak -e 's/{$ADMIN NAME}/kaotisk/' \
  etc/charybdis/ircd.conf
sed -i.bak -e 's/{$ADMIN_DESCRIPTION}/some descr/' \
  etc/charybdis/ircd.conf
sed -i.bak -e 's/{$ADMIN_EMAIL}/kaotisk@arching-kaos.com/' \
  etc/charybdis/ircd.conf
sed -i.bak -e 's/{$IRC_AUTH_PASSWORD}/somepass/' \
  etc/charybdis/ircd.conf
sed -i.bak -e 's/{$GOD_IRC_PASSWORD}/somepass/' \
  etc/charybdis/ircd.conf
```

Icecast2

And we continue with icecast2 in which we set in the same way the following variables in `./etc/icecast2/icecast.xml` file. As the suffix of the filename implies, its an XML[1] file, hence it's consisting of tags with values (e.g. <tag>value</tag>).

```
echo "...2/4 icecast"
```

- Location We can set our instance's location.

```
<location>{$LOCATION}</location>
```

- Administrator email We can set an administrator email. It shows up on the icecast web page under the Server Information section of page Version.

```
<admin>{$ADMIN_EMAIL}</admin>
```

- Authentication

[1]https://www.w3.org/XML

```
<authentication>
  <!-- Sources log in with username 'source' -->
  <source-password>{$ICECAST_SOURCE_PASSWORD}</source-password>
  <!-- Relays log in with username 'relay' -->
  <relay-password>{$ICECAST_RELAY_PASSWORD}</relay-password>
  <!-- Administrators log in with username 'admin' alias -->
  <admin-user>admin</admin-user>
  <admin-password>{$ICECAST_ADMIN_PASSWORD}</admin-password>
</authentication>
```

- Hostname

```
<hostname>{$ICECAST_HOSTNAME}</hostname>`
```

- Setting our webradio domain name to be able to access icecast's stats

```
<http-headers>
<header name="Access-Control-Allow-Origin" \
  value="{$RADIO_WEBSITE_BASEURL}" />
</http-headers>
```

So we do `sed` again. These are the variables that you can edit accordingly in `./install.sh`.

```
sed -i.bak -e 's/{$LOCATION}/earth/' etc/icecast2/icecast.xml
sed -i.bak -e 's/{$ADMIN_EMAIL}/kaotisk@arching-kaos.com/' \
  etc/icecast2/icecast.xml
sed -i.bak -e 's/{$ICECAST_SOURCE_PASSWORD}/hackme/' \
  etc/icecast2/icecast.xml
sed -i.bak -e 's/{$ICECAST_RELAY_PASSWORD}/hackme/' \
  etc/icecast2/icecast.xml
sed -i.bak -e 's/{$ICECAST_ADMIN_PASSWORD}/hackme/' \
  etc/icecast2/icecast.xml
sed -i.bak -e \
's/{$ICECAST_HOSTNAME}/icecast.arching-kaos.local/' \
  etc/icecast2/icecast.xml
sed -i.bak -e \
  's/ \
  {$RADIO_WEBSITE_BASEURL}/ \
  http:\/\/radio.arching-kaos.local/' \
  etc/icecast2/icecast.xml
```

Liquidsoap

There goes liquidsoap settings also. We basically need to edit two lines:

```
echo "...3/4 liquidsoap"
```

- Source icecast password

```
output.icecast(
  %vorbis,
  radio,
  mount="demo.ogg",
  host="localhost",
  password="{$ICECAST_SOURCE_PASSWORD}",
  url="http://radio.arching-kaos.com",
  description="Arching Kaos radio is a collaborative webradio."
)
```

- Live stream access

```
live = input.harbor(
  "live",
  port=3210,
  user="source",
  password="{$LIVE_SOURCE_PASSWORD}"
)
```

Which we also edit with `sed`.

```
sed -i.bak -e 's/{$ICECAST_SOURCE_PASSWORD}/hackme/' \
  etc/liquidsoap/radio.liq
sed -i.bak -e 's/{$LIVE_SOURCE_PASSWORD}/hackmetoo/' \
  etc/liquidsoap/radio.liq
```

NGINX and intergration

```
echo "...4/4 nginx"
```

We use nginx to serve our pages through the network. `./etc/conf.d` files as well source files from the submodules `./modules` that use the same variables are set in the following way. - API

```
sed -i.bak -e 's/{$API_SERVER_NAME}/api.arching-kaos.local/g' \
  etc/nginx/conf.d/api.conf \
  modules/arching-kaos-radio/src/ShowList.js \
  modules/arching-kaos-radio/src/Menu.js
```

- Documentation

```
sed -i.bak \
  -e 's/{$DOCS_SERVER_NAME}/docs.arching-kaos.local/g' \
  etc/nginx/conf.d/api.conf \
  etc/nginx/conf.d/docs.conf
```

- Generic domain name

```
sed -i.bak -e 's/{$DOMAIN_NAME}/arching-kaos.local/g' \
  etc/nginx/conf.d/default.conf \
  modules/arching-kaos-radio/src/Signature.js
```

- Icecast

```
sed -i.bak \
  -e 's/{$ICECAST_SERVER_NAME}/icecast.arching-kaos.local/g' \
  etc/nginx/conf.d/icecast.conf \
  modules/arching-kaos-radio/src/App.js \
  modules/arching-kaos-radio/src/Menu.js \
  modules/arching-kaos-radio/src/NowPlaying.js
```

- IPFS

```
sed -i.bak \
  -e 's/{$IPFS_SERVER_NAME}/ipfs.arching-kaos.local/g' \
  etc/nginx/conf.d/ipfs-gateway.conf \
  modules/arching-kaos-api/config.js
```

- IRC
  - Server settings

```
sed -i.bak \
  -e 's/{$IRC_SERVER_NAME}/irc.arching-kaos.local/g' \
  etc/nginx/conf.d/irc.conf \
  etc/thelounge/config.js
```

  - Client settings

```
sed -i.bak \
  -e 's/{$IRC_CLIENT}/http:\/\/127.0.0.1:9000/g' \
  modules/arching-kaos-radio/src/Chat.js \
  modules/arching-kaos-irc/index.html
```

- Radio

```
sed -i.bak \
  -e 's/{$RADIO_SERVER_NAME}/radio.arching-kaos.local/g' \
  etc/nginx/conf.d/radio-arching.conf \
  modules/arching-kaos-radio/src/Header.js
```

- SSB

```
sed -i.bak -e 's/{$SSB_SERVER_NAME}/ssb.arching-kaos.local/g' \
  etc/nginx/conf.d/ssb.conf \
  etc/ssb-pub-data/config
```

- Opentracker

```
sed -i.bak -e \
   's/{$TRACKER_SERVER_NAME}/tracker.arching-kaos.local/' \
   etc/nginx/conf.d/tracker.conf
```

API

We, then, create a folder where the `arching-kaos-api` files will reside when we will run it. And also, copy some sample files for getting the API ready.

```
echo "Create API directories"
# sh ./modules/arching-kaos-api/api-dir.sh
# Going the custom way again
export ARCHING_KAOS_API_DIR=$PWD/storage/.arching-kaos-api
mkdir -p $ARCHING_KAOS_API_DIR/downloads
cp modules/arching-kaos-api/ipList.json-sample \
  $ARCHING_KAOS_API_DIR/ipList.json
cp modules/arching-kaos-api/shows.json-sample \
  $ARCHING_KAOS_API_DIR/shows.json
```

4.6.3 Starting docker images or build them in some cases

In folder `./scripts/` files starting with `docker-` are small scripts that start certain images for our project to work.

```
echo "Getting docker scripts ready ..."
echo "Proceeding arching-kaos installation ..."
```

Documentation

So we basically want to have our source documentation available so we start it from a script.

```
echo "Starting docs..."
sh ./scripts/docker-arching-kaos-docs.sh
echo "... done"
```

Icecast2

```
echo "Starting icecast..."
sh ./scripts/docker-icecast.sh
echo "... done"
```

IPFS

```
echo "Starting ipfs..."
sh ./scripts/docker-ipfs.sh
echo "... done"
```

Opentracker

```
echo "Starting opentracker..."
sh ./scripts/docker-opentracker.sh
echo "... done"
```

SSB

```
echo "Starting ssb..."
sh ./scripts/docker-ssb-create.sh
echo "... done"
```

Liquidsoap

```
echo "Starting liquidsoap..."
sh ./scripts/docker-liquidsoap.sh
echo "... done"
```

API

```
echo "Starting API..."
cd modules/arching-kaos-api
sh ./install.sh
cd ../..
sh ./modules/arching-kaos-api/run.sh
echo "... done"
```

Web interface for the radio

```
echo "Starting webpage..."
cd modules/arching-kaos-radio
./start.sh
echo "... done"
cd ../..
```

dat-store

```
echo "Starting dat-store..."
cd modules/docker-dat-store
sh ./build.sh
sh ./start.sh
echo "... done"
cd ../..
```

thelounge

```
echo "Starting thelounge..."
sh ./scripts/docker-thelounge.sh
echo "... done"
```

4.6.4 Building last components

IRC (charybdis)

```
echo "Setting up IRC"
sh ./scripts/charybdis-simple-install.sh
cp etc/charybdis/ircd.conf $HOME/ircd/etc/ircd.conf
cp etc/charybdis/ircd.motd $HOME/ircd/etc/ircd.motd
echo "Starting IRC..."
$HOME/ircd/bin/charybdis
#TODO Insert crontab @reboot
```

4.6.5 Patching everything into NGINX web proxy

```
echo "Starting NGINX..."
docker run --name nginx \
  --restart always \
  -d --network=host \
  -v $PWD/etc/nginx/conf.d:/etc/nginx/conf.d \
  -v $PWD/modules/arching-kaos-generic:/srv/generic \
  -v $PWD/modules/arching-kaos-irc:/srv/irc \
  -v $PWD/modules/arching-kaos-ssb:/srv/ssb \
  nginx
```

4.6.6 Edit SSB landpage

Execute the following to get your SSB's servers public id:

```
docker exec -it sbot sbot whoami > my_ssb_ident
```

and save it to file `my_ssb_ident`. Also execute to get a code for 100 invites it:

```
docker exec -it sbot \
  sbot invite.create 100 > invite_code
```

apply the info above in `./modules/arching-kaos-ssb/index.html`.

4.7 There you go!

Our application is successfully installed and running!!

```
echo "Voila!"
```