Scuola universitaria professionale della Svizzera italiana
Dipartimento tecnologie innovative

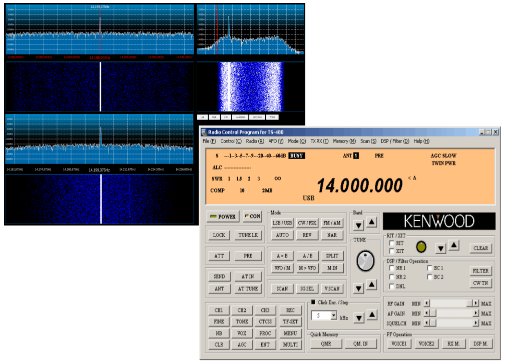
SUPSI

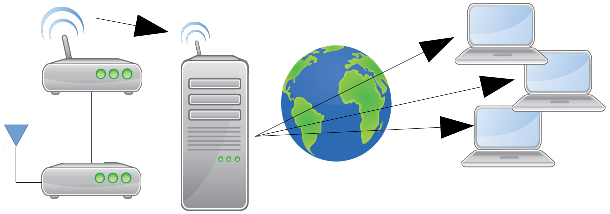
Software Defined Radio with Remote Head and Internet Clients

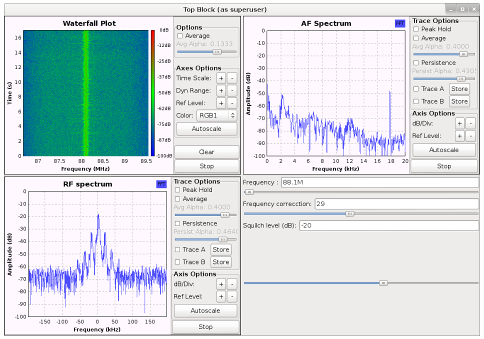
Studente/i	Relatore	Correlatore
Giovanni Franza	Tiziano Leidi	Loris Grossi
Corso di laurea	Modulo / Codice Progetto	Anno
Master of Science in Informatics	Tesi	2016
Committente	Data	
	05/09/2017	

Agenda

- Idea
  - Challenges / Framework
- Development
  - Tools / Devices / Computers /Software
- Technologies
  - USB / UDP / Websocket / Graphic & Audio context / DSP elaboration
- Demo
- Measures
- From Here



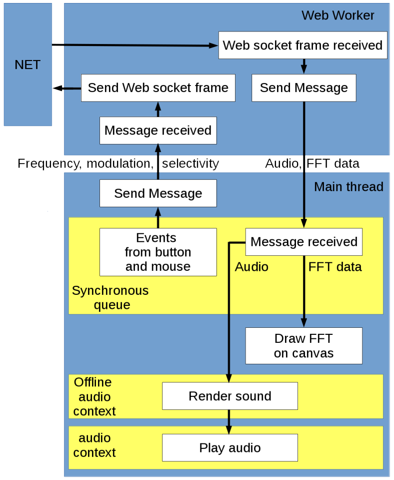


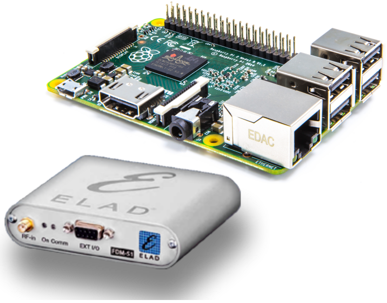


Frequency : 88.1M

Frequency correction: 29

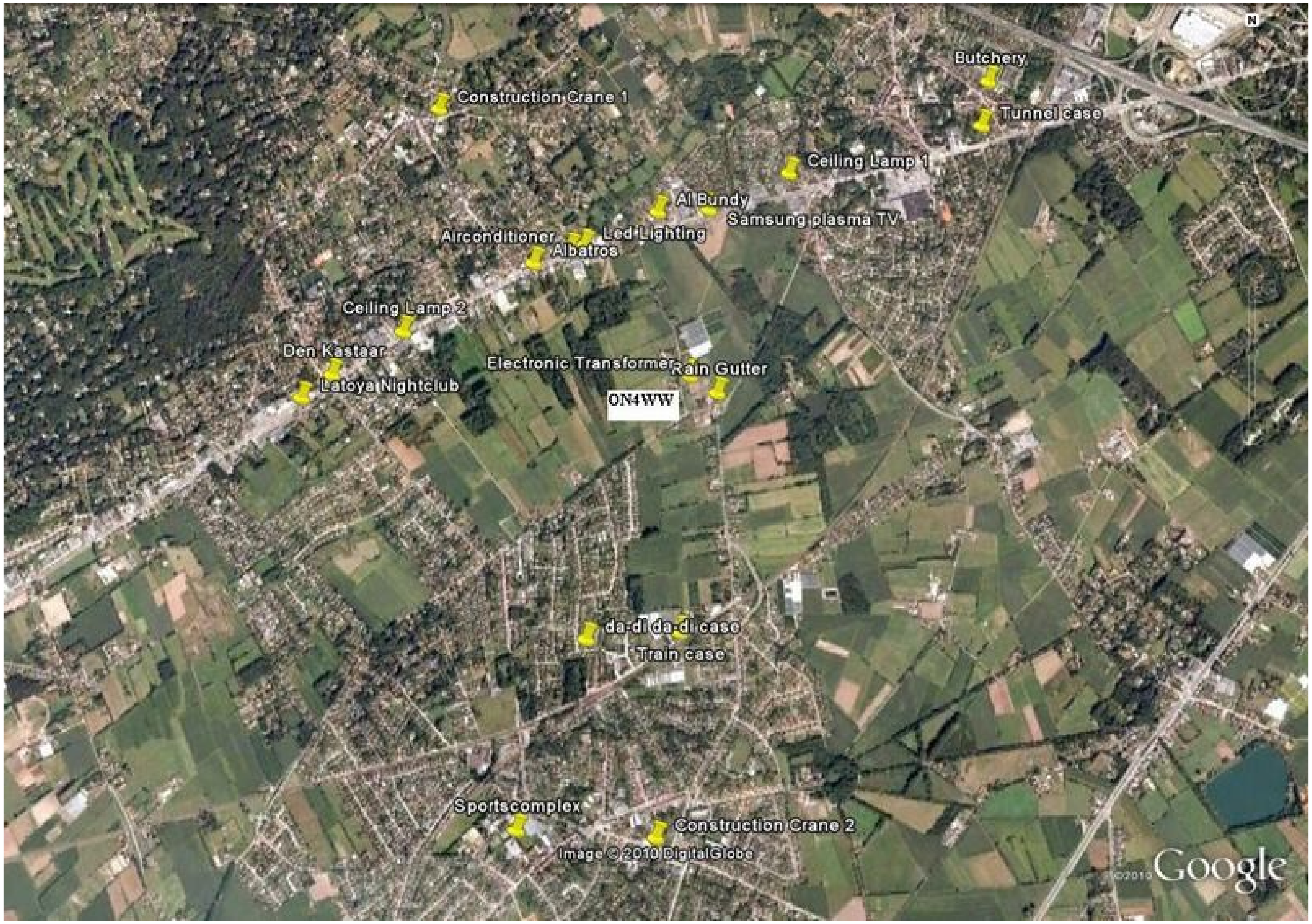
Squelch level (dB): -20



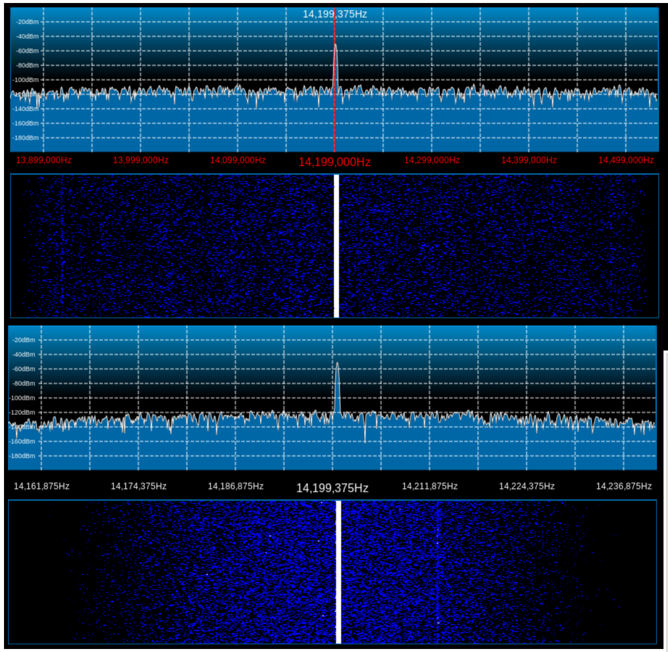


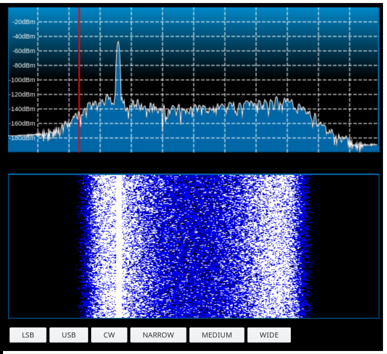
Idea

Is a crowded town the best position for a radio receiver ?

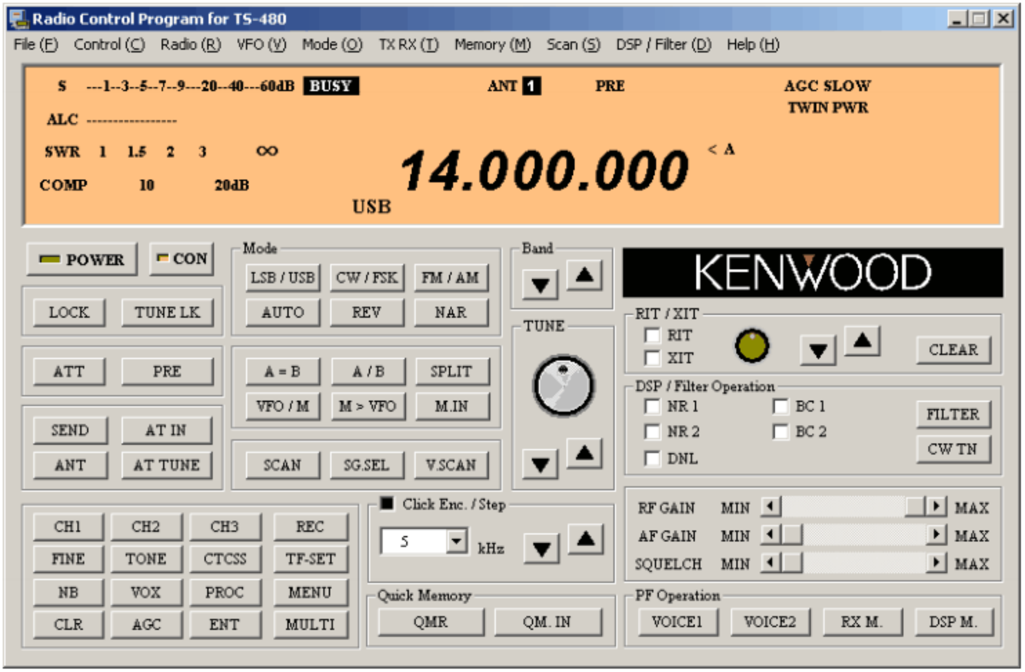


Challenges - UI





Can a SDR have the same UI of a normal Radio receiver ?


Radio Control Program for TS-480

S ---1--3--5--7--9---20--40---60dB BUSY ANT 1 PRE AGC SLOW TWIN PWR

ALC ----------------

SWR 1 1.5 2 3 ∞

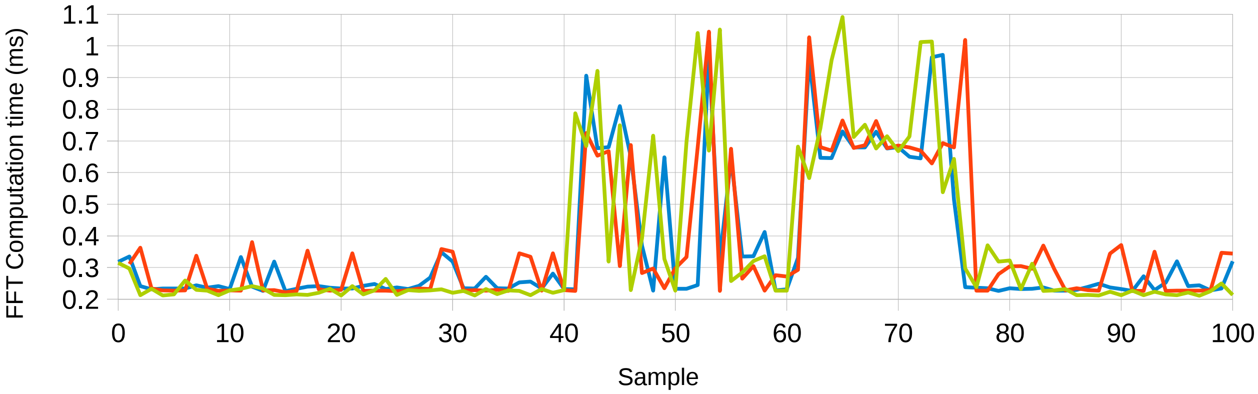
COMP 10 20dB

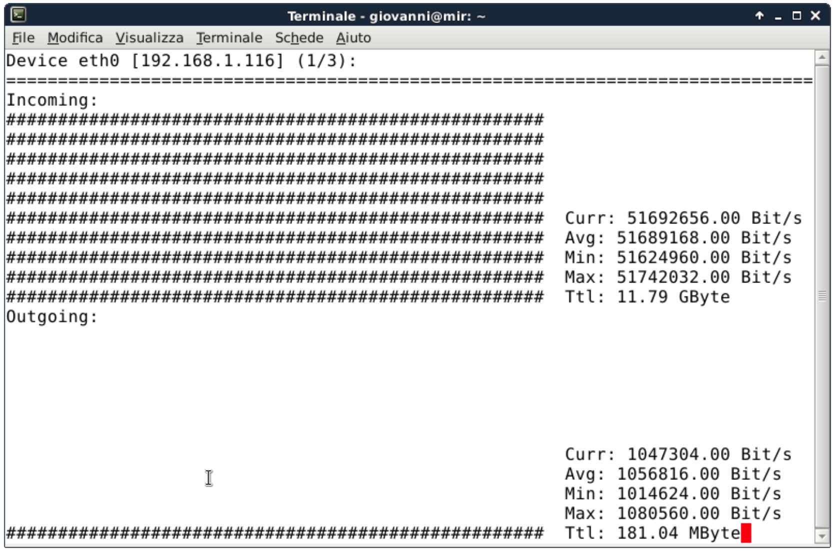
14.000.000 < A

USB

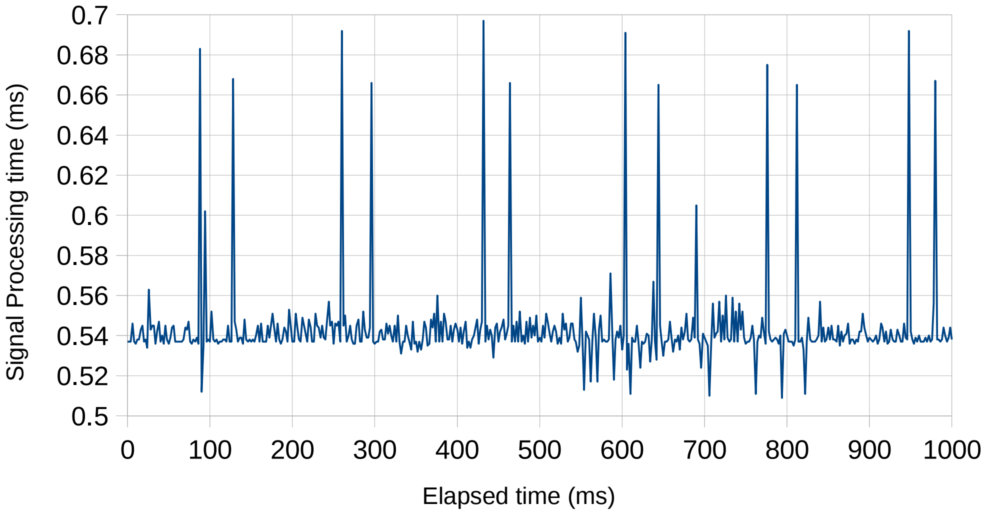
Challenge - OS

Can a non native RT system host a SDR program ?



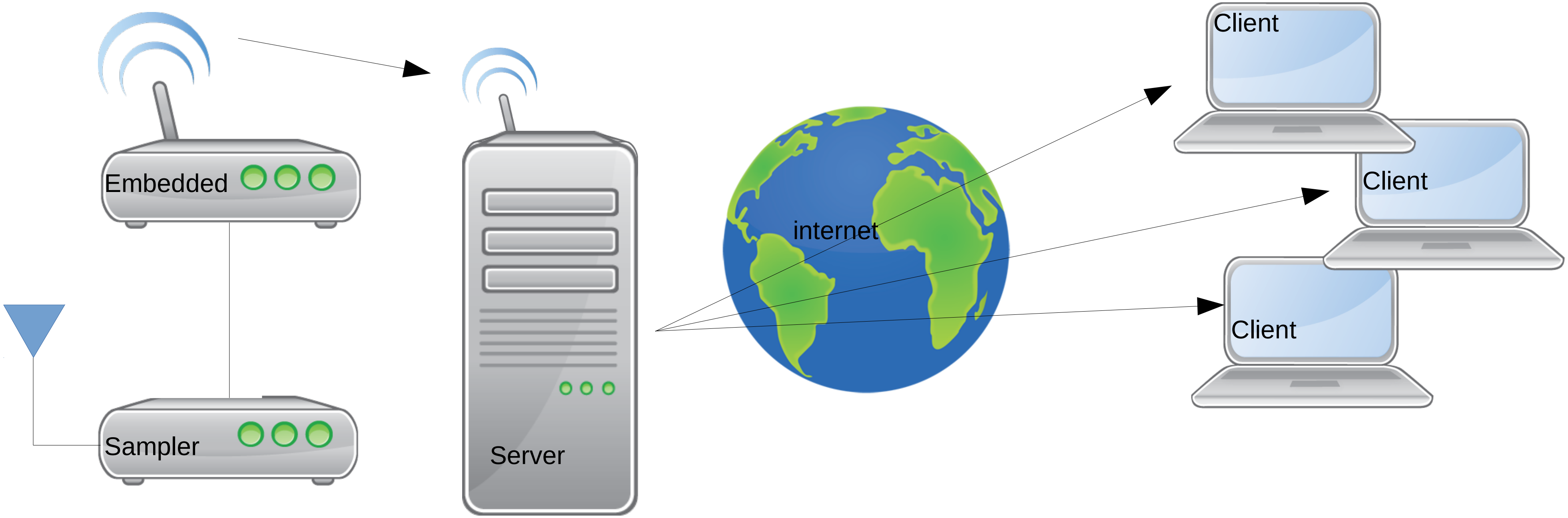


```
Device eth0 [192.168.1.116] (1/3):
==============================================================================
Incoming:
####################################################
####################################################
####################################################
####################################################
####################################################
####################################################  Curr: 51692656.00 Bit/s
####################################################  Avg: 51689168.00 Bit/s
####################################################  Min: 51624960.00 Bit/s
####################################################  Max: 51742032.00 Bit/s
####################################################  Ttl: 11.79 GByte
Outgoing:






                                                      Curr: 1047304.00 Bit/s
                                                      Avg: 1056816.00 Bit/s
                                                      Min: 1014624.00 Bit/s
                                                      Max: 1080560.00 Bit/s
####################################################  Ttl: 181.04 MByte
```



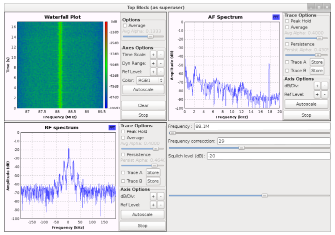
Framework

- Sampler: together with an embedded on the top of a hill
- Server: a PC in a webfarm nearby
- Clients: a browser anywhere



Development

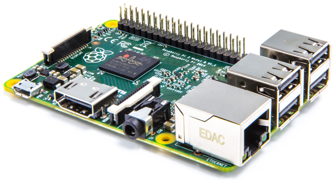
- Hardware: COTS components off the shelf
  - Samplers
  - Embedded
  - PC
- Development: Spiral model
- Software: KISS keep it simple and stupid

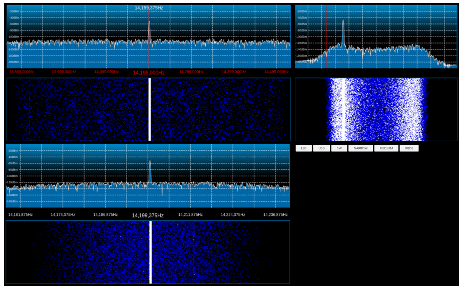


Frequency : 88.1M

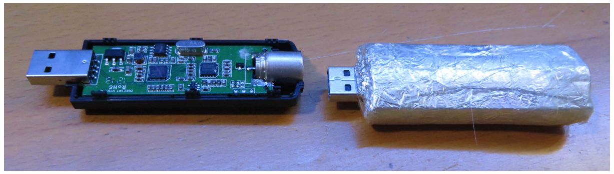
Frequency correcction: 29

Squlch level (dB): -20





Devices

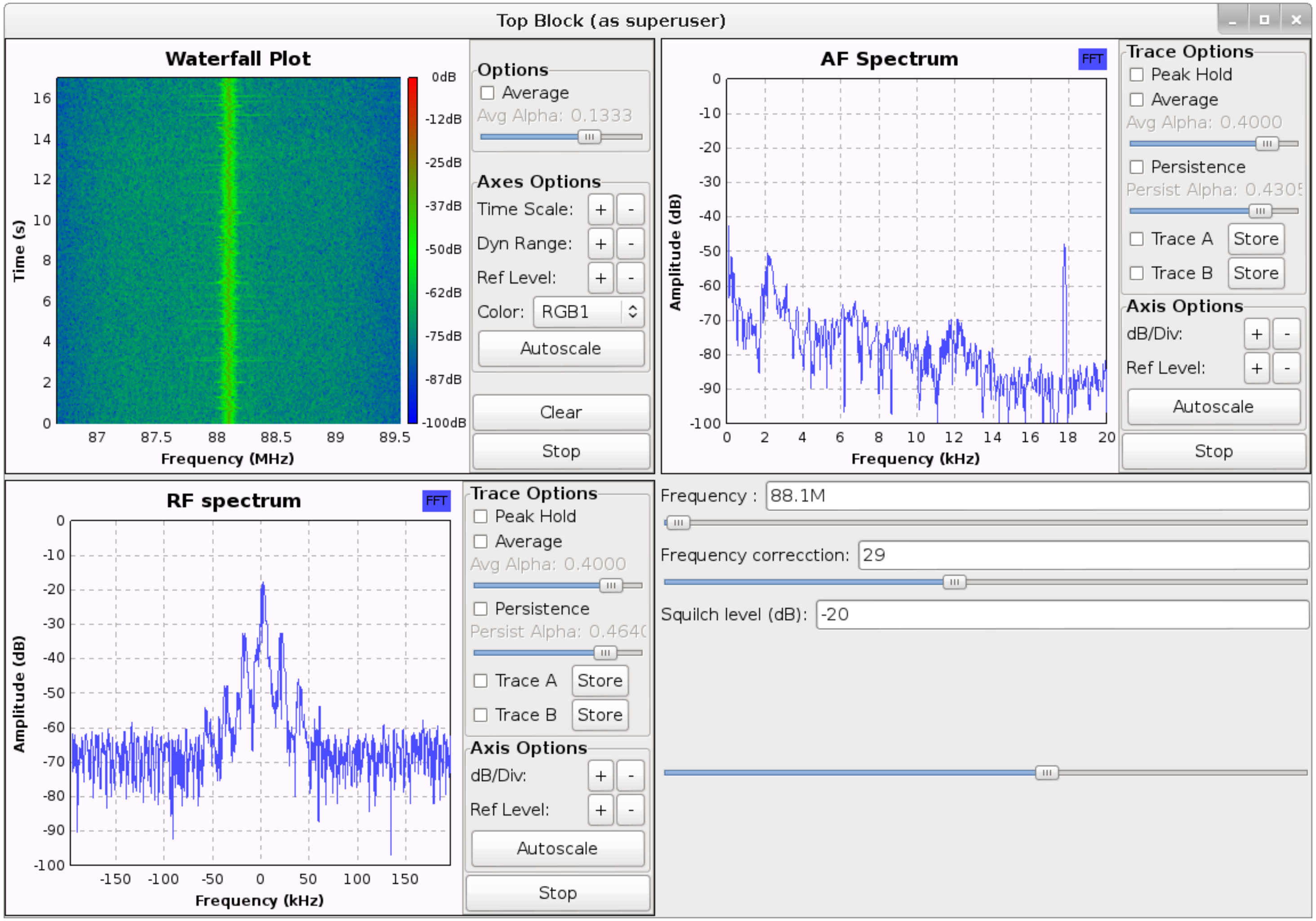


OFF ON
WARNING!
DO NOT REMOVE FRONT OR REAR PANELS. REFER SERVICING TO QUALIFIED PERSONNEL.
HF
VHF
ANTENNA
SE0F2L
EXT I/O
ELAD
Made in Italy

- Philosophy: COTS (components off the shelf)
- At the beginning very cheap RTL dongles
  - Tested with GNU Radio
- Then, thanks to an agreement with ELAD, much better and documented samplers S1 and S2
  - First, developed GNU Radio drivers
  - Then USB→UDP tunnelling program

Tools

GNU Radio as a reference and a test bed



Top Block (as superuser)

Frequency : 88.1M

Frequency correcction: 29

Squilch level (dB): -20

Waterfall Plot options: □ Average, Avg Alpha: 0.1333, Color: RGB1

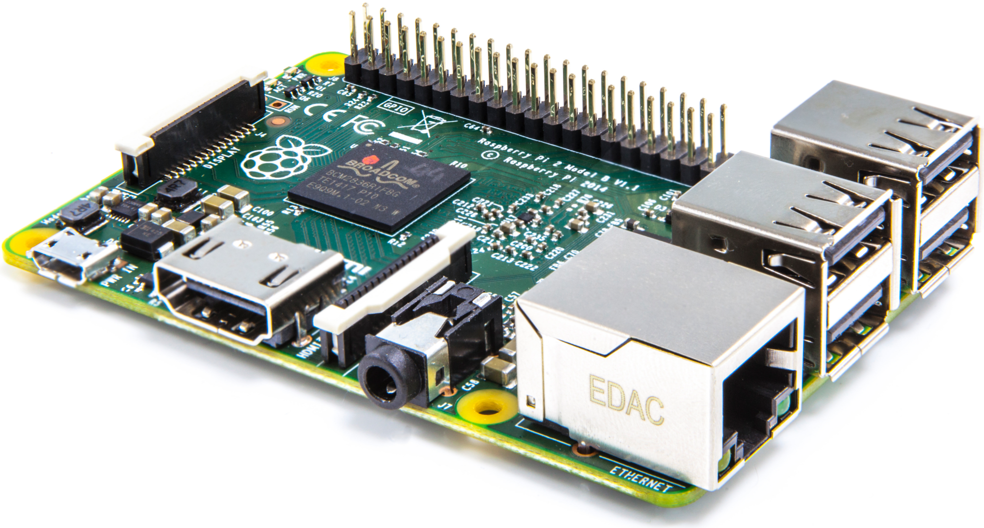
AF Spectrum trace options: □ Peak Hold, □ Average, Avg Alpha: 0.4000, □ Persistence, Persist Alpha: 0.4305, □ Trace A, □ Trace B

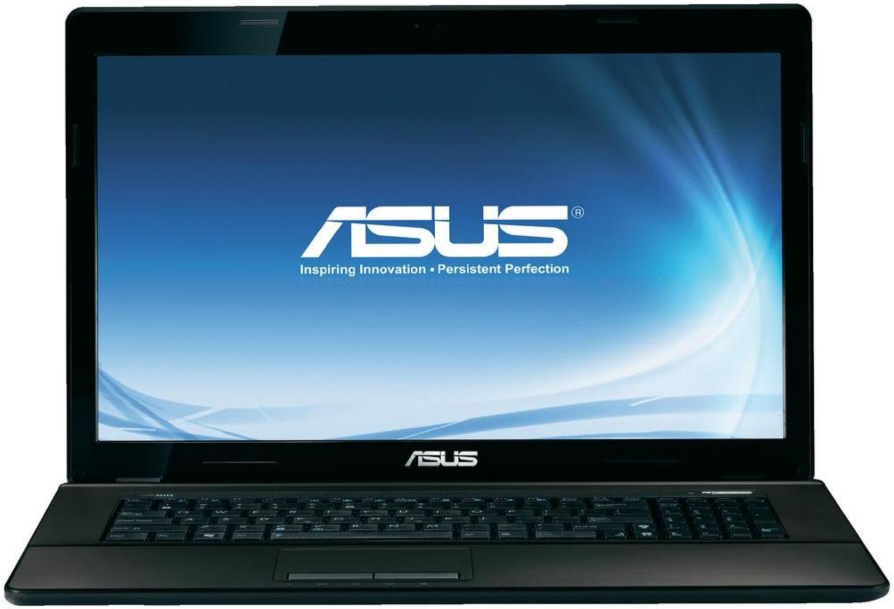
RF spectrum trace options: □ Peak Hold, □ Average, Avg Alpha: 0.4000, □ Persistence, Persist Alpha: 0.4640, □ Trace A, □ Trace B

Computers

Also here, COTS

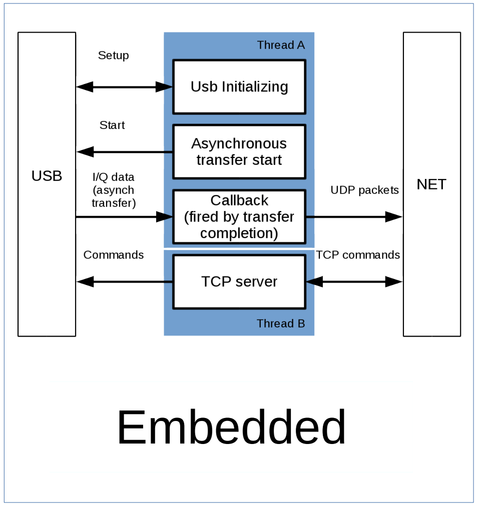
Raspberry PI 2 as embedded that hosts USB→UDP bridge



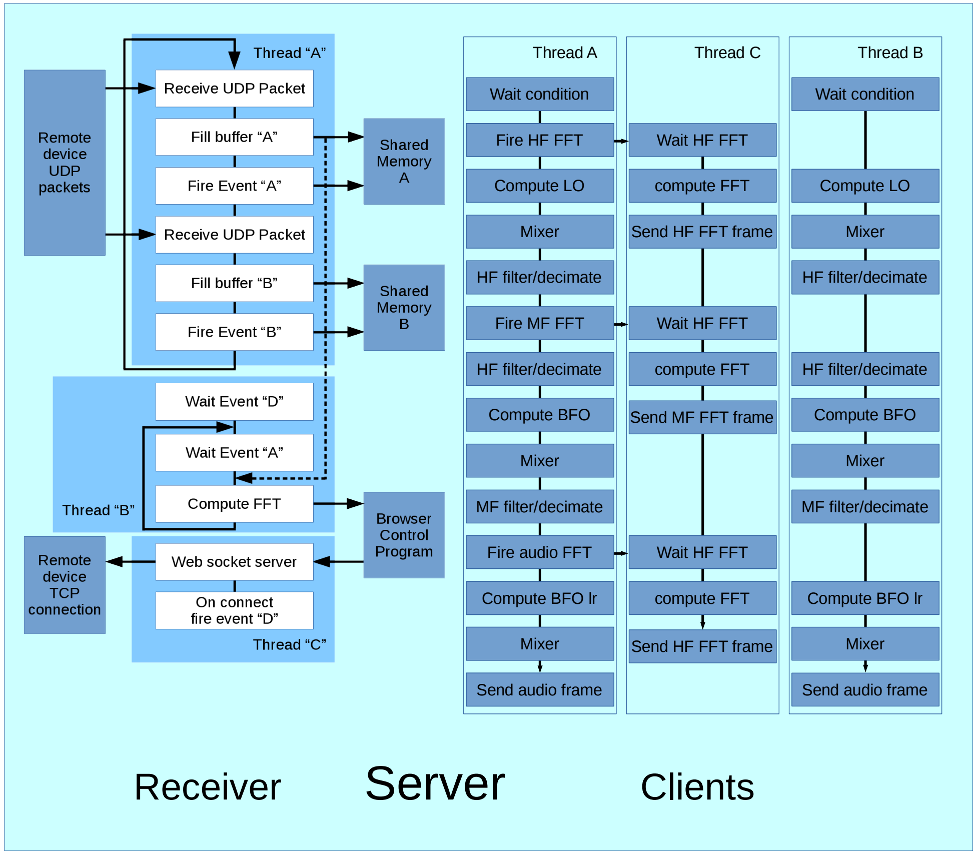


Computer with I7 Intel hyperthreaded quad core CPU as “server”

Software


Thread A
Setup
Usb Initializing
Start
Asynchronous transfer start
USB
I/Q data (asynch transfer)
Callback (fired by transfer completion)
UDP packets
NET
Commands
TCP commands
TCP server
Thread B

Embedded


Thread "A"
Remote device UDP packets
Receive UDP Packet
Fill buffer "A"
Fire Event "A"
Receive UDP Packet
Fill buffer "B"
Fire Event "B"
Shared Memory A
Shared Memory B
Wait Event "D"
Wait Event "A"
Thread "B"
Compute FFT
Browser Control Program
Remote device TCP connection
Web socket server
On connect fire event "D"
Thread "C"

Thread A
Wait condition
Fire HF FFT
Compute LO
Mixer
HF filter/decimate
Fire MF FFT
HF filter/decimate
Compute BFO
Mixer
MF filter/decimate
Fire audio FFT
Compute BFO lr
Mixer
Send audio frame

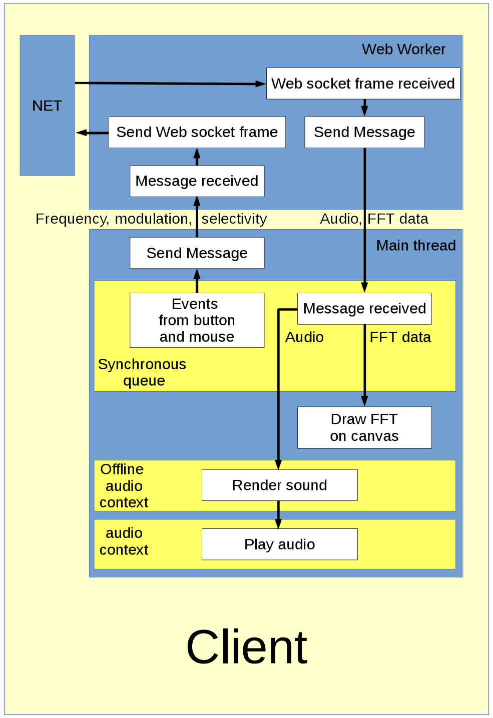
Thread C
Wait HF FFT
compute FFT
Send HF FFT frame
Wait HF FFT
compute FFT
Send MF FFT frame
Wait HF FFT
compute FFT
Send HF FFT frame

Thread B
Wait condition
Compute LO
Mixer
HF filter/decimate
HF filter/decimate
Compute BFO
Mixer
MF filter/decimate
Compute BFO lr
Mixer
Send audio frame

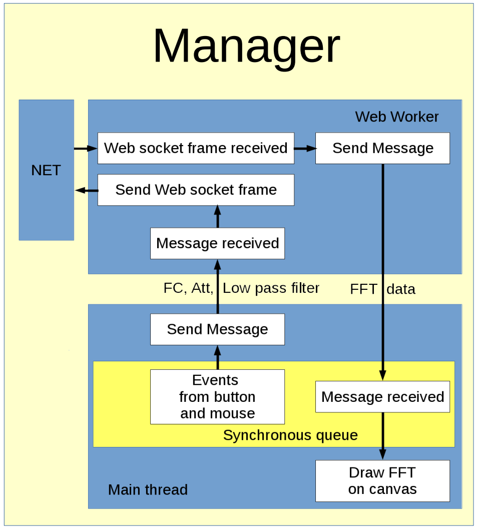
Receiver

Server

Clients

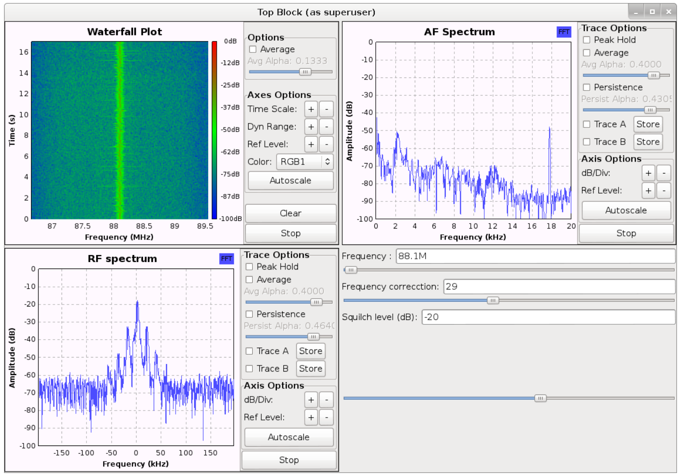

Web Worker
NET
Web socket frame received
Send Web socket frame
Send Message
Message received
Frequency, modulation, selectivity
Audio, FFT data
Main thread
Send Message
Events from button and mouse
Message received
Audio
FFT data
Synchronous queue
Draw FFT on canvas
Offline audio context
Render sound
audio context
Play audio

Client


Manager
Web Worker
NET
Web socket frame received
Send Message
Send Web socket frame
Message received
FC, Att, Low pass filter
FFT data
Send Message
Events from button and mouse
Message received
Synchronous queue
Draw FFT on canvas
Main thread

First phase

- Before any work an RTL-SDR test was done using GNURadio
  - To verify RTL-SDR usability
- Before testing GNURadio was studied
  - To be able to use it correctly
- Some flowgraphs were developed
  - They resulted useful to test RTL-SDR
- RTL-SDR discarded in favor of ELAD samplers
  - Development of modules useful for project (callbacks structure reused)



Waterfall Plot options: ☐ Average; Avg Alpha: 0.1333; Color: RGB1

AF Spectrum trace options: ☐ Peak Hold; ☐ Average; Avg Alpha: 0.4000; ☐ Persistence; Persist Alpha: 0.4305; ☐ Trace A Store; ☐ Trace B Store

RF spectrum trace options: ☐ Peak Hold; ☐ Average; Avg Alpha: 0.4000; ☐ Persistence; Persist Alpha: 0.4640; ☐ Trace A Store; ☐ Trace B Store

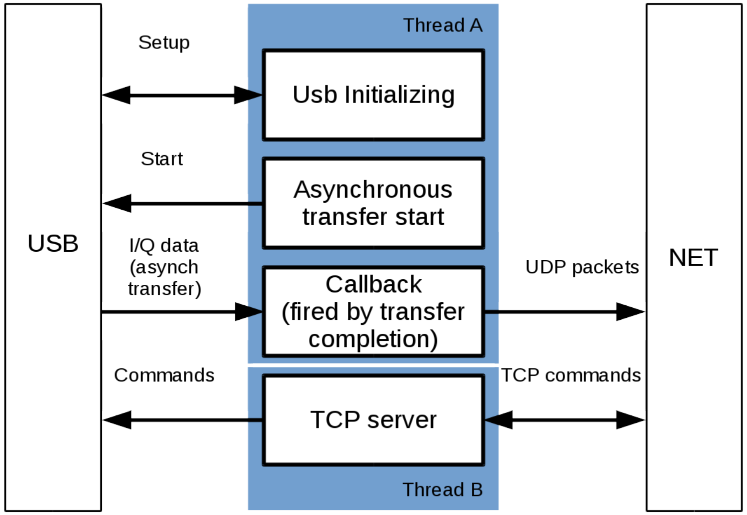
Frequency : 88.1M

Frequency correcction: 29

Squilch level (dB): -20

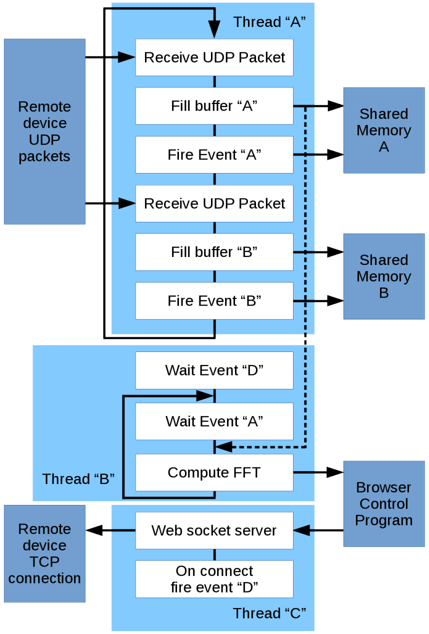
Embedded

- USB initialization (device dependent)
  - For some devices also FPGA setup
- Data transfer via callback
  - To avoid data loss callbacks and USB data transfer overlaps
- Network data flow via UDP
  - No data loss, verified
- Commands received and confirmed using TCP
  - To exploit its characteristics.


USB
Setup
Start
I/Q data (asynch transfer)
Commands
Thread A
Usb Initializing
Asynchronous transfer start
Callback (fired by transfer completion)
TCP server
Thread B
UDP packets
TCP commands
NET

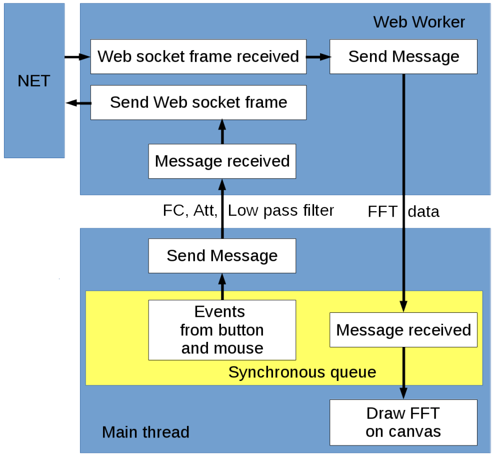
Receiver

- It receives UDP packets
- It fills shared memories
  - Two banks, with associated events
  - To allow clients to work on a bank at a time
- It talks with Manager (web application)
  - Via Websocket
- It computes FFT
  - To allow manager to draw spectrogram and waterfall



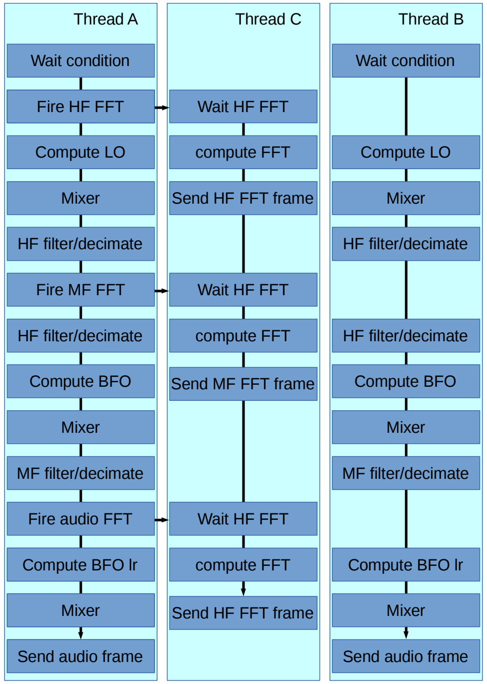
Manager

- Web worker to escape javascript monothreading
  - Events/messages bwtween worker and main thread
- Graphic context to design graphic elements
  - Layered graphics to reduce computation load
- Websocket communications



Client

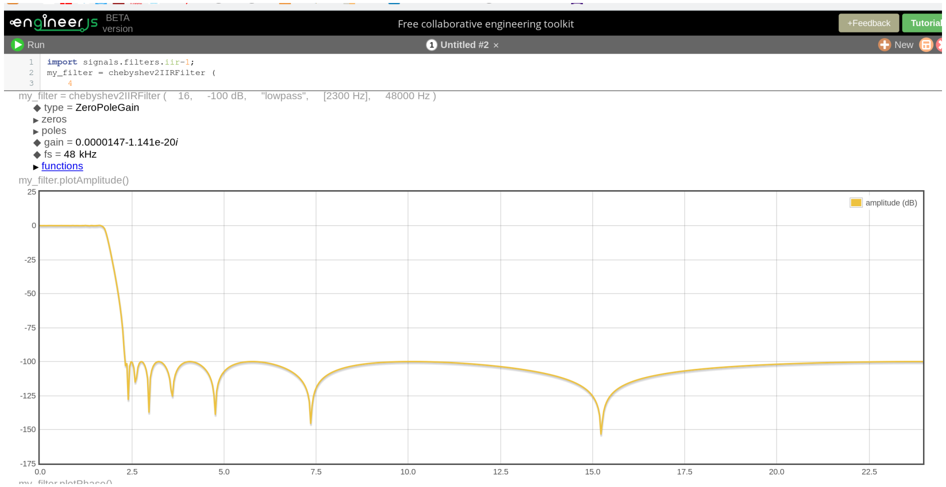
- One client for each external request
  - Developed as TCP server daemon
- Two threads to manage data processing
  - All “normal superhet” processing: VFO, mixer, MF filtering, BFO, audio
- One thread to manage FFT
  - Periodically computed at three different levels: input, intermediate, audio
- One thread to manage Websocket



Thread A	Thread C	Thread B
Wait condition		Wait condition
Fire HF FFT	Wait HF FFT	
Compute LO	compute FFT	Compute LO
Mixer	Send HF FFT frame	Mixer
HF filter/decimate		HF filter/decimate
Fire MF FFT	Wait HF FFT	
HF filter/decimate	compute FFT	HF filter/decimate
Compute BFO	Send MF FFT frame	Compute BFO
Mixer		Mixer
MF filter/decimate		MF filter/decimate
Fire audio FFT	Wait HF FFT	
Compute BFO lr	compute FFT	Compute BFO lr
Mixer	Send HF FFT frame	Mixer
Send audio frame		Send audio frame

Data processing

- VFO / mixer to center to 0Hz the wanted band
- Antialiasing filter and decimation to spread the band and make “downsampling gain”
- BFO / mixer to center to 0Hz the middle of wanted baseband
- Filtering and decimation to fit to a reasonable sampling rate
- BFO / mixer to move filtered band to baseband
- Classic FIR / IIR
- Classic “prostapheresis” mixer



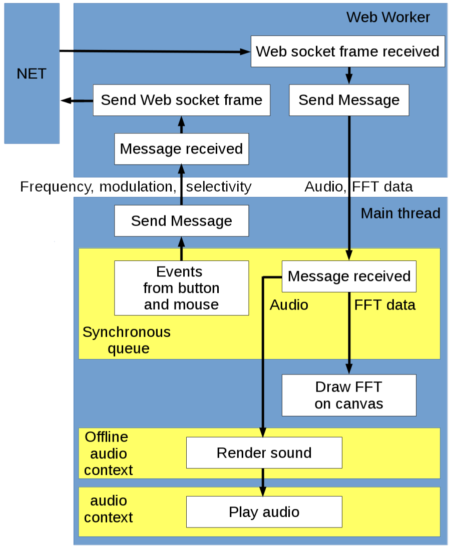
```
import signals.filters.iir-1;
my_filter = chebyshev2IIRFilter (
    4
```

```
my_filter = chebyshev2IIRFilter (   16,    -100 dB,    "lowpass",    [2300 Hz],    48000 Hz )
  ◆ type = ZeroPoleGain
  ▸ zeros
  ▸ poles
  ◆ gain = 0.0000147-1.141e-20i
  ◆ fs = 48 kHz
  ▸ functions
my_filter.plotAmplitude()
```

my_filter.plotPhase()

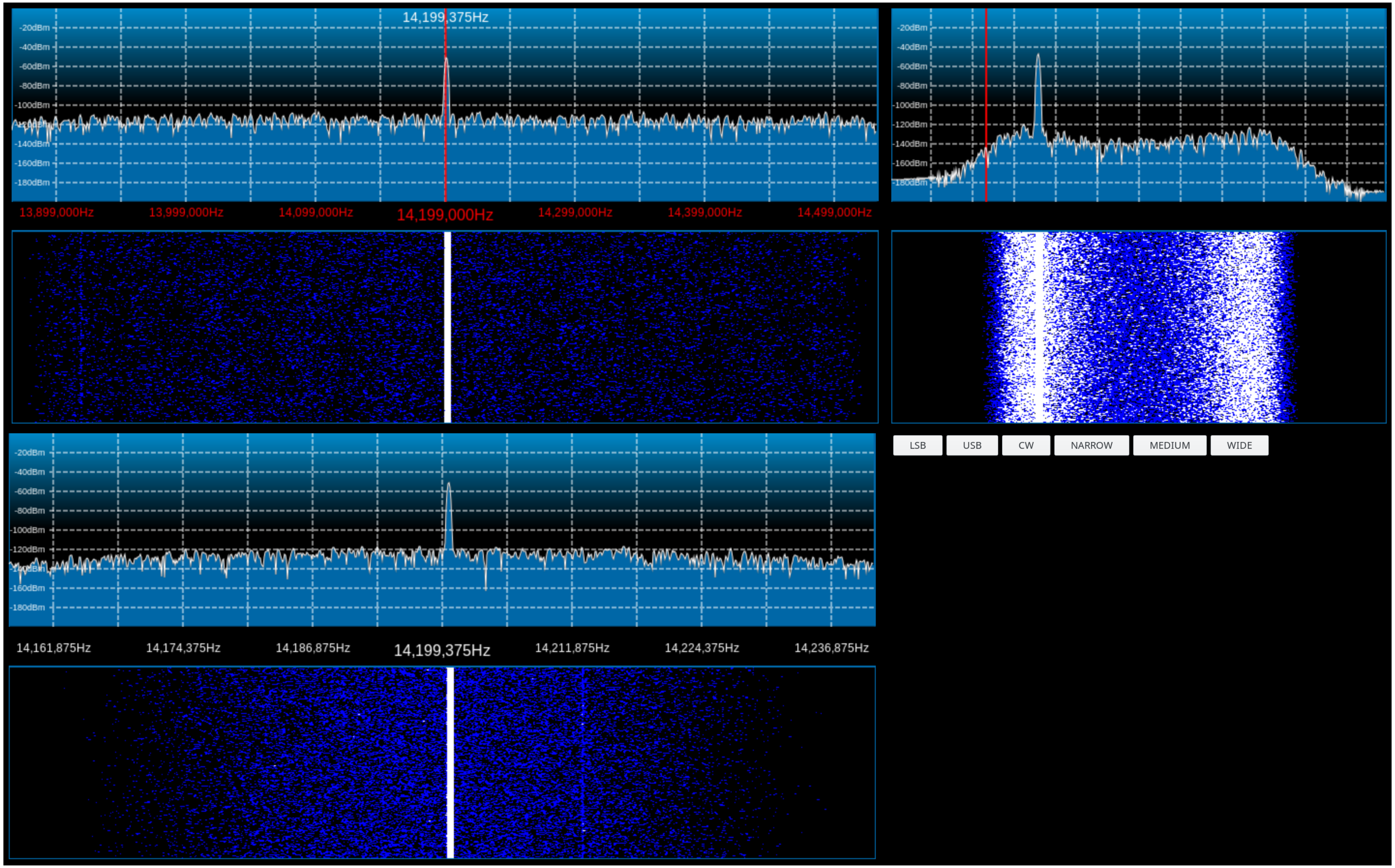
Web client

- Webworker that manages Websocket
  - Uses messages to talk to main thread
- Graphic context that draws spectrograms
- Audio context that plays audio
  - After offline audio context sound conversion
- Offline audio context



Demo

Client

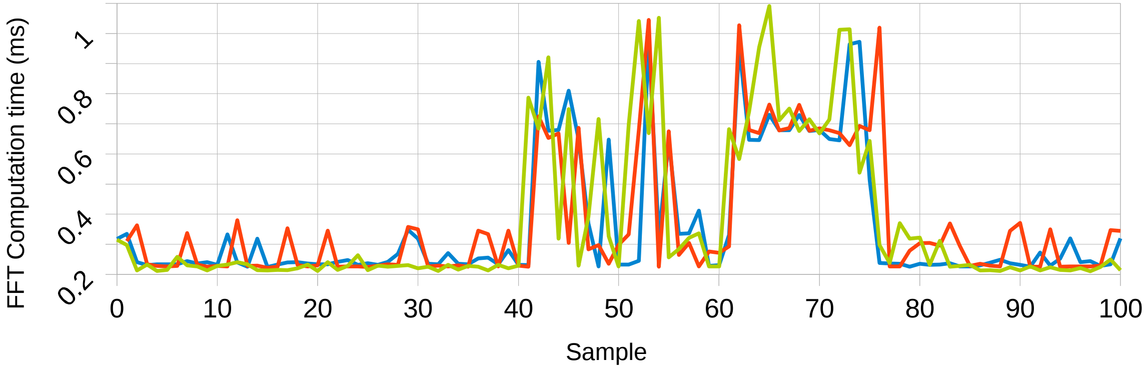

14,199,375Hz
-20dBm
-40dBm
-60dBm
-80dBm
-100dBm
-120dBm
-140dBm
-160dBm
-180dBm
13,899,000Hz
13,999,000Hz
14,099,000Hz
14,199,000Hz
14,299,000Hz
14,399,000Hz
14,499,000Hz
LSB
USB
CW
NARROW
MEDIUM
WIDE
14,161,875Hz
14,174,375Hz
14,186,875Hz
14,199,375Hz
14,211,875Hz
14,224,375Hz
14,236,875Hz

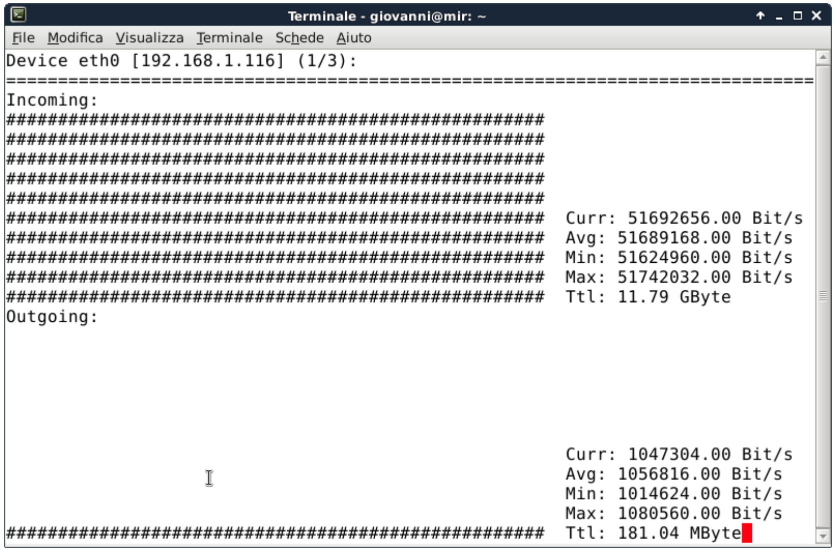
Measures

50Mb/s on bridge

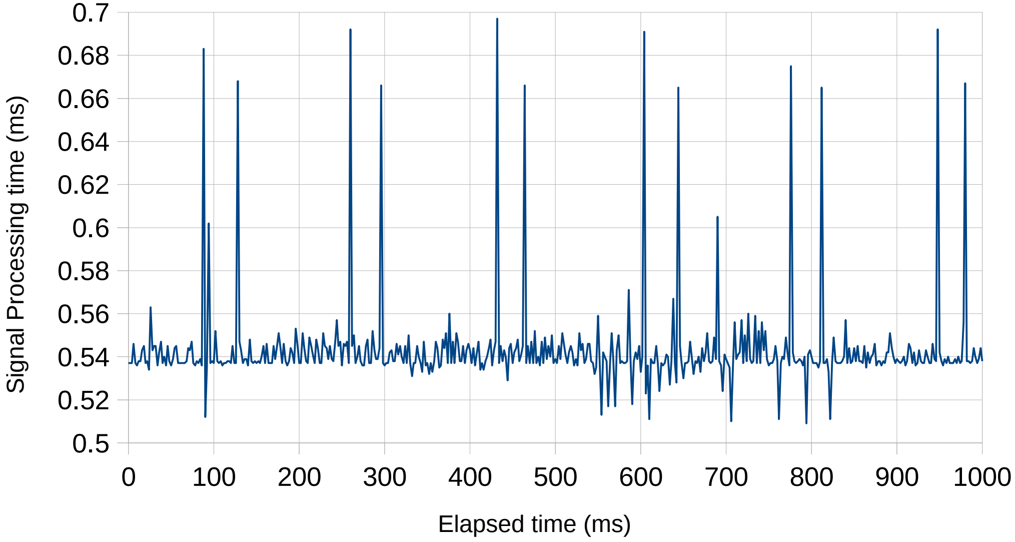
1Mb/s each client

4 simultaneous clients





```
Device eth0 [192.168.1.116] (1/3):
==============================================================================
Incoming:
####################################################
####################################################
####################################################
####################################################
####################################################
####################################################  Curr: 51692656.00 Bit/s
####################################################  Avg: 51689168.00 Bit/s
####################################################  Min: 51624960.00 Bit/s
####################################################  Max: 51742032.00 Bit/s
####################################################  Ttl: 11.79 GByte
Outgoing:






                                                      Curr: 1047304.00 Bit/s
                                                      Avg: 1056816.00 Bit/s
                                                      Min: 1014624.00 Bit/s
                                                      Max: 1080560.00 Bit/s
####################################################  Ttl: 181.04 MByte
```



From here

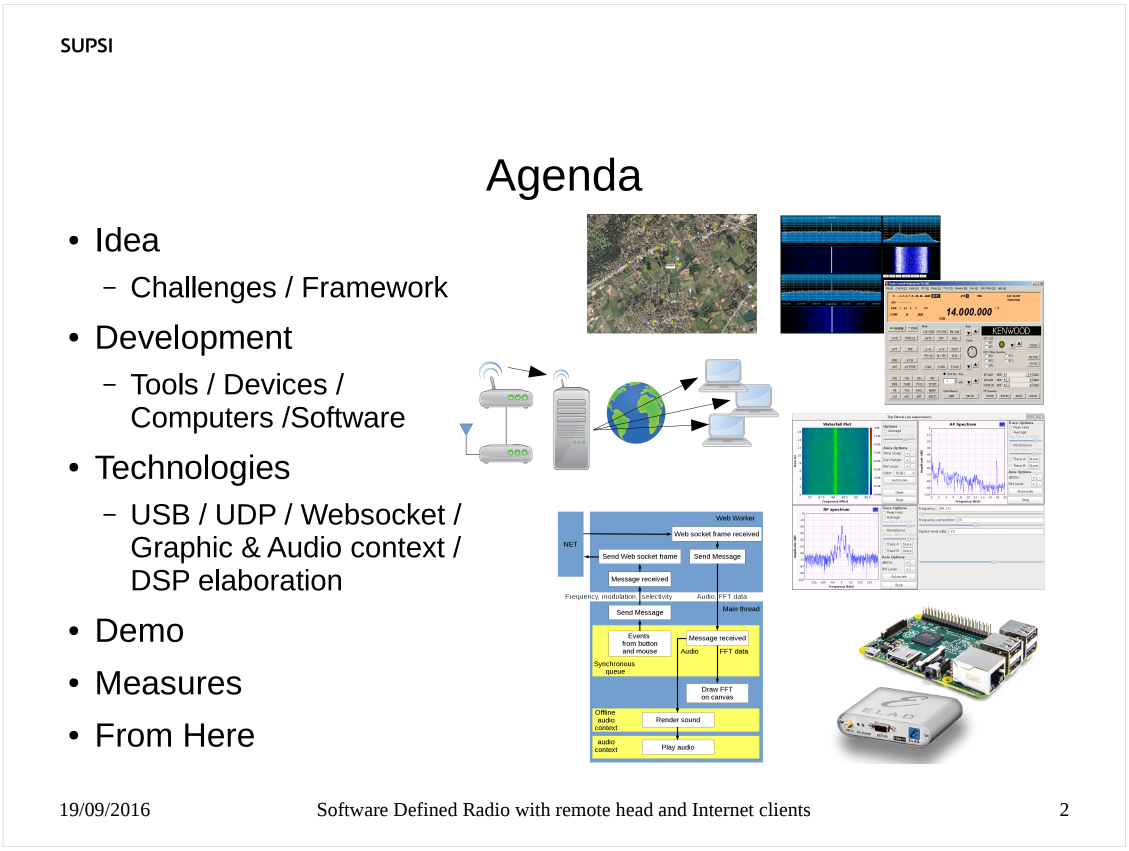
- Change UDP data stream format
- Change Audio format (codec)
- CIC filters for decimator
- FFT filters for Bandwidth
- Other modulations
- AGC, Volume control

Scuola universitaria professionale della Svizzera italiana
Dipartimento tecnologie innovative

SUPSI

Software Defined Radio with Remote Head and Internet Clients

Studente/i	Relatore	Correlatore
Giovanni Franza	Tiziano Leidi	Loris Grossi
Corso di laurea	Modulo / Codice Progetto	Anno
Master of Science in Informatics	Tesi	2016
Committente	Data	
	05/09/2017	



SUPSI

Agenda

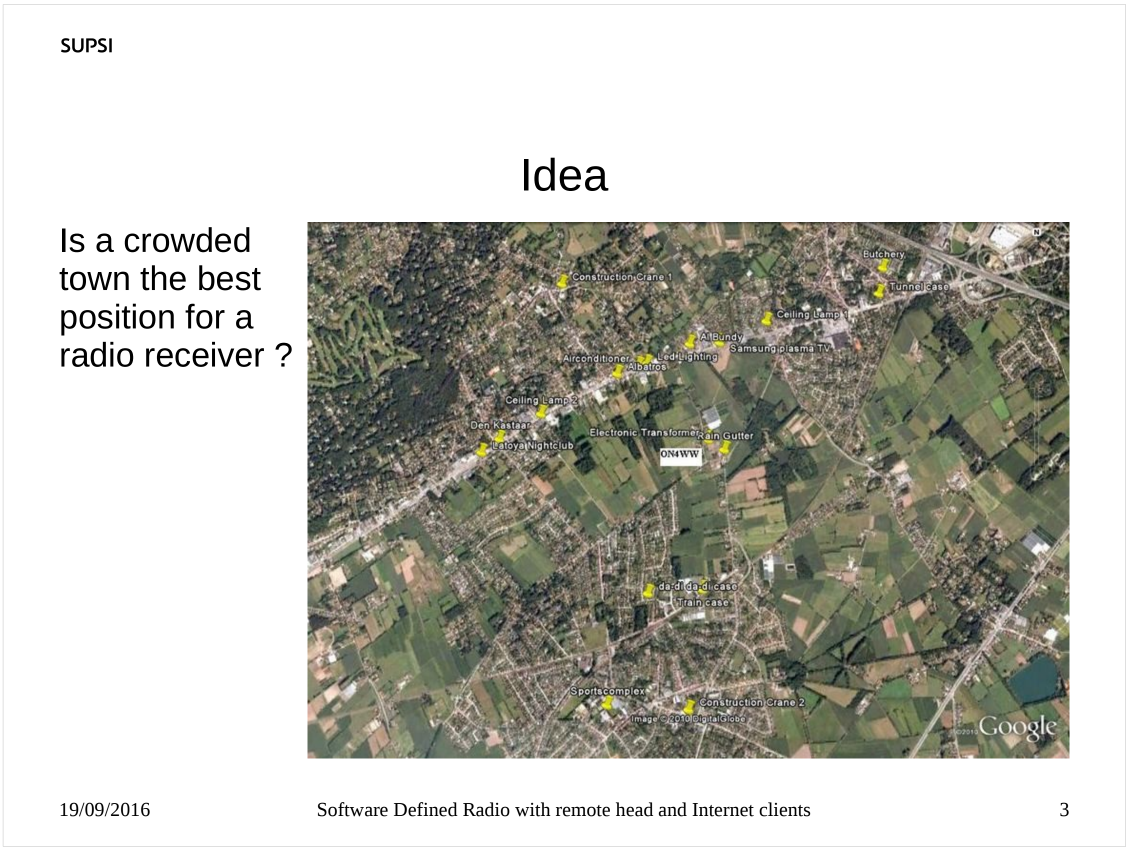
- Idea
  - Challenges / Framework
- Development
  - Tools / Devices / Computers /Software
- Technologies
  - USB / UDP / Websocket / Graphic & Audio context / DSP elaboration
- Demo
- Measures
- From Here

Why I do this project

How I developed it

Which are the results

What I plan to do after it



SUPSI

Idea

Is a crowded town the best position for a radio receiver ?

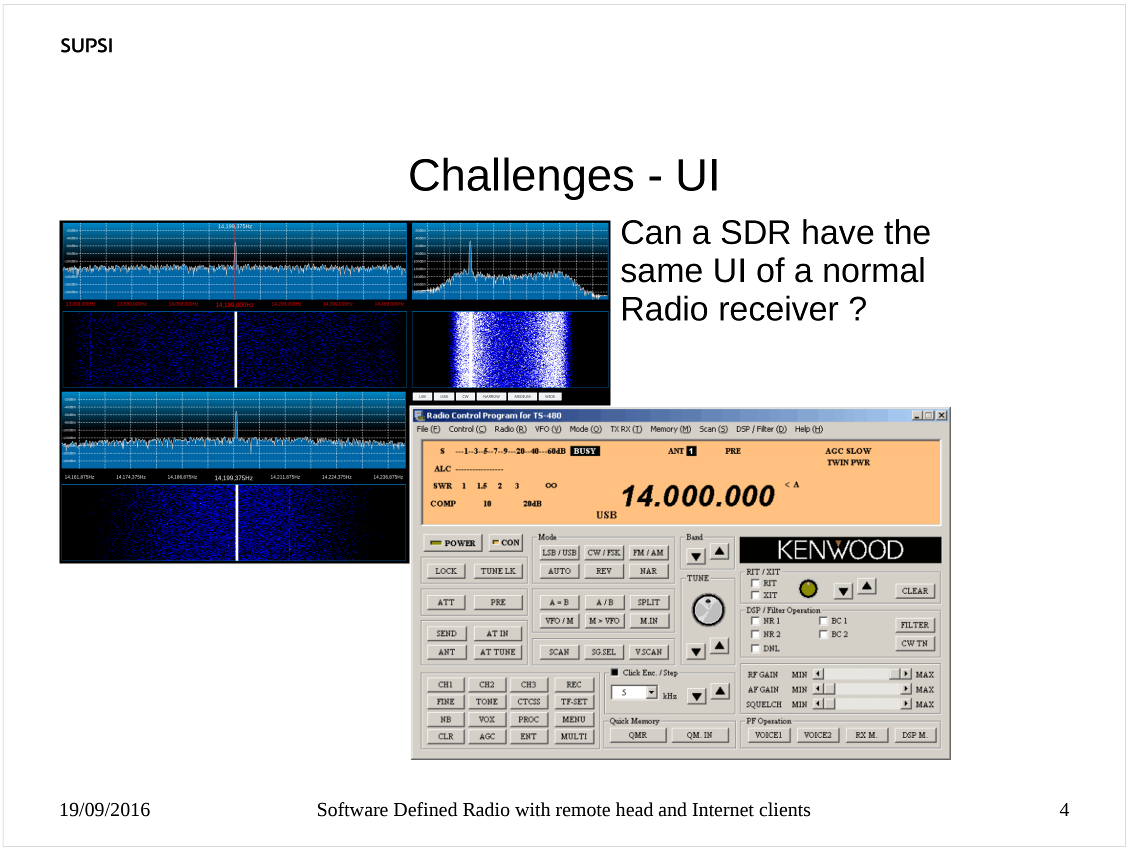
19/09/2016 Software Defined Radio with remote head and Internet clients 3

Old good times, when cities were electrically silent, are gone, a lot of noise is generated by any sort of appliances.

“Normal” people fears antennas, ORNI + building regulations could make antennas installation a nightmare.

Even when on holidays, in a distant location, an OM wishes to use his/her radio without issues related to customs.

The solution can be remotization.



SUPSI

Challenges - UI

Can a SDR have the same UI of a normal Radio receiver ?

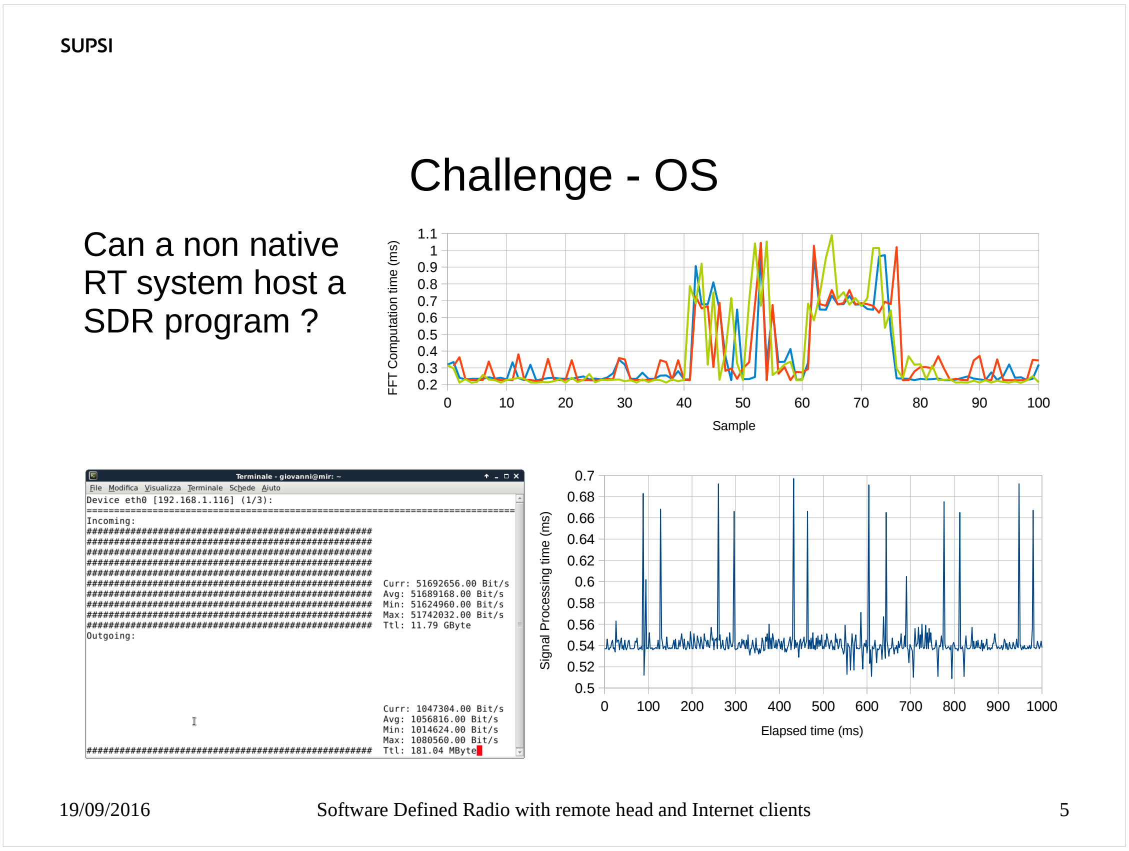
There are challenges in the development of such an idea.

One aspect of the challenge is to build something substancially new, not simply a copy of the front panel of a good old radio.

SDR, with FFT, changes the scenario of a radio receiver: no more “blind tuning” but “click and point”.

Don’t throw the child with the dirty water: a sort of “knob” is really useful also because OMs are used to tune “rotating their wrist”.

UI based on web browser, to exploit their ubiquity.



SUPSI

Challenge - OS

Can a non native RT system host a SDR program ?

Terminale - giovanni@mir: ~
File Modifica Visualizza Terminale Schede Aiuto

```
Device eth0 [192.168.1.116] (1/3):
==============================================================================
Incoming:
###################################################
###################################################
###################################################
###################################################
###################################################
###################################################  Curr: 51692656.00 Bit/s
###################################################  Avg: 51689168.00 Bit/s
###################################################  Min: 51624960.00 Bit/s
###################################################  Max: 51742032.00 Bit/s
###################################################  Ttl: 11.79 GByte
Outgoing:




                                                     Curr: 1047304.00 Bit/s
                                                     Avg: 1056816.00 Bit/s
                                                     Min: 1014624.00 Bit/s
                                                     Max: 1080560.00 Bit/s
###################################################  Ttl: 181.04 MByte
```

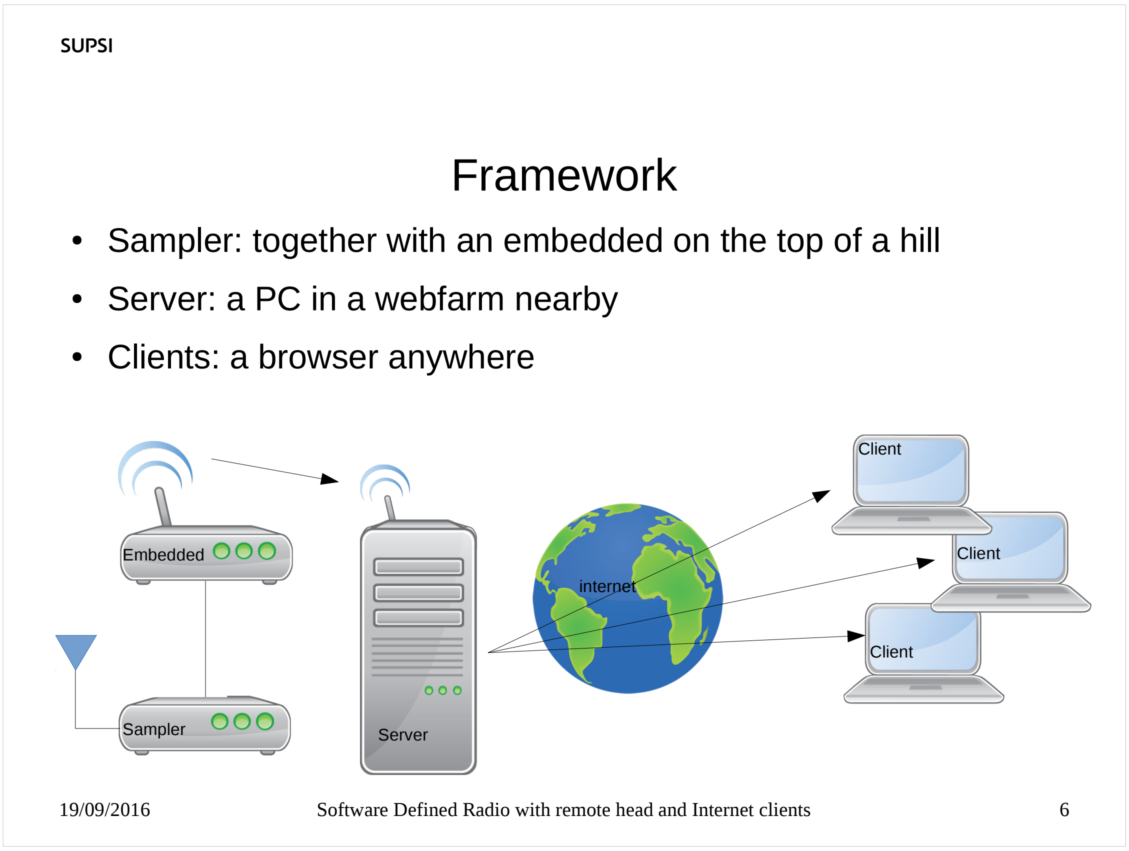
I’m a “linux addicted”, but Linux is not completely RT: is it possible to use it for SDR?

How much other parts of OS can interfere with operations?

From the measurements we see that there are glitches, but we could limit their effects.

Another challenge is the SDR itself, the conversion from known analogic operations to numeric elaboration.

This work is a building of a robust skeleton, usable, but, mostly, usable as a base for future development.



SUPSI

Framework

- Sampler: together with an embedded on the top of a hill
- Server: a PC in a webfarm nearby
- Clients: a browser anywhere

19/09/2016 Software Defined Radio with remote head and Internet clients 6

A simple scheme:

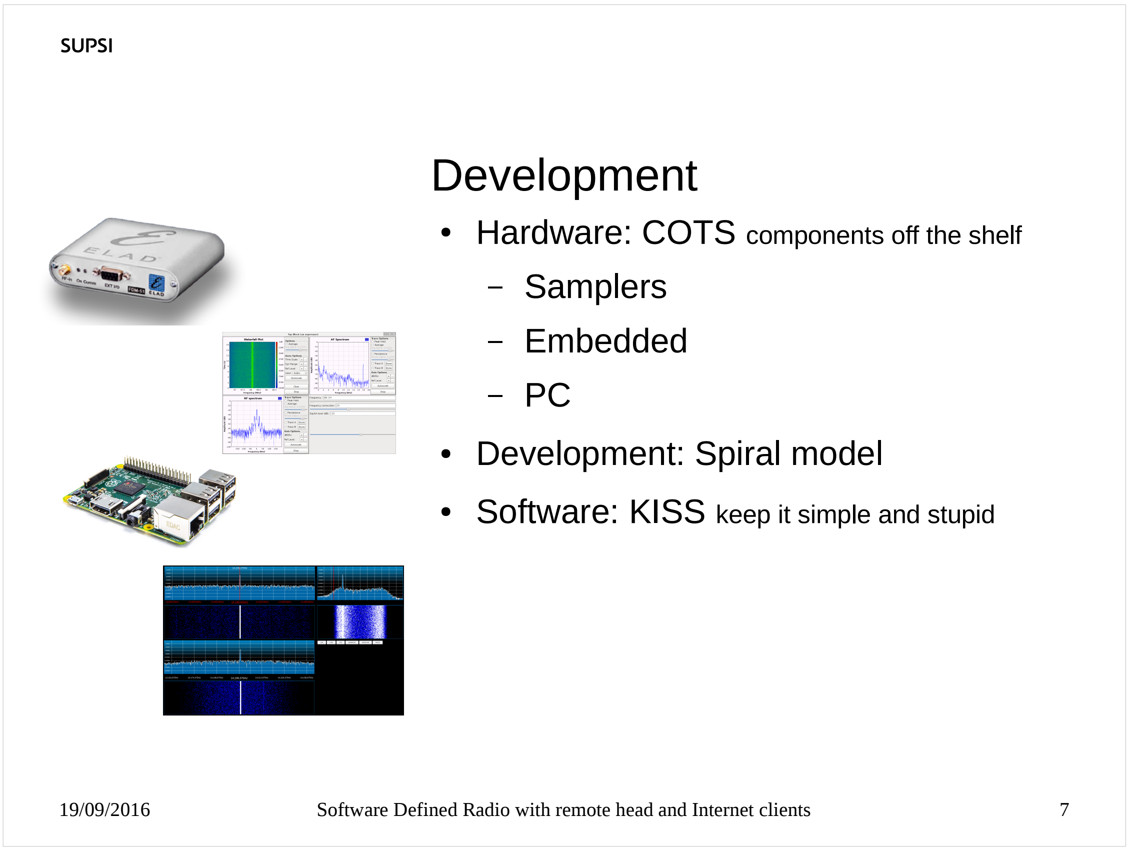
On a top of a hill I put antenna, sampler, an embedded that sends UDP data.

On a webfarm I put a server.

Connection between top-of-the-hill and server can be made with a WiFi bridge in the 5GHz band.

The server is accessible via Internet.

Bridge tested on S.Salvatore-Barbengo for 68MBit/s data rate, flow required is 50Mbit/s, if could be feasible.



SUPSI

Development

- Hardware: COTS components off the shelf
  - Samplers
  - Embedded
  - PC
- Development: Spiral model
- Software: KISS keep it simple and stupid

19/09/2016 Software Defined Radio with remote head and Internet clients 7

I believe in the philosophy of COTS, to reduce costs and to have quick replacements.

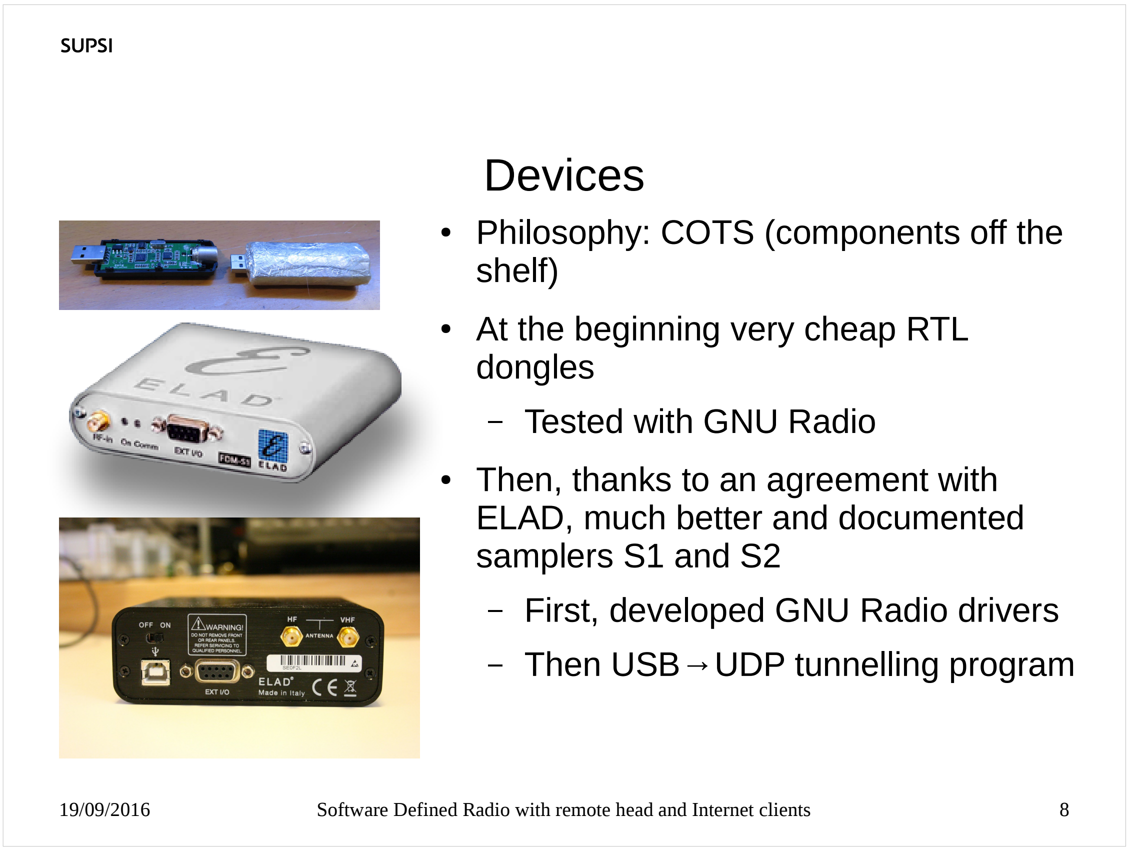
I also try to make the software as simple as possibile to allow further development.

I use a spiral model tinking – coding – testing – changing, one phase at a time.

First phase was acquiring experience on GNURadio.

Then I’ve used GNURadio to test hardware.

Then I wrote software using GNURadio to test output or as a reference.



SUPSI

Devices

- Philosophy: COTS (components off the shelf)
- At the beginning very cheap RTL dongles
  - Tested with GNU Radio
- Then, thanks to an agreement with ELAD, much better and documented samplers S1 and S2
  - First, developed GNU Radio drivers
  - Then USB→UDP tunnelling program

19/09/2016 Software Defined Radio with remote head and Internet clients 8

Following COTS, at the beginning I’ve used very inexpensive RTL-SDR devices (10$/unit). But these devices have poor performances.

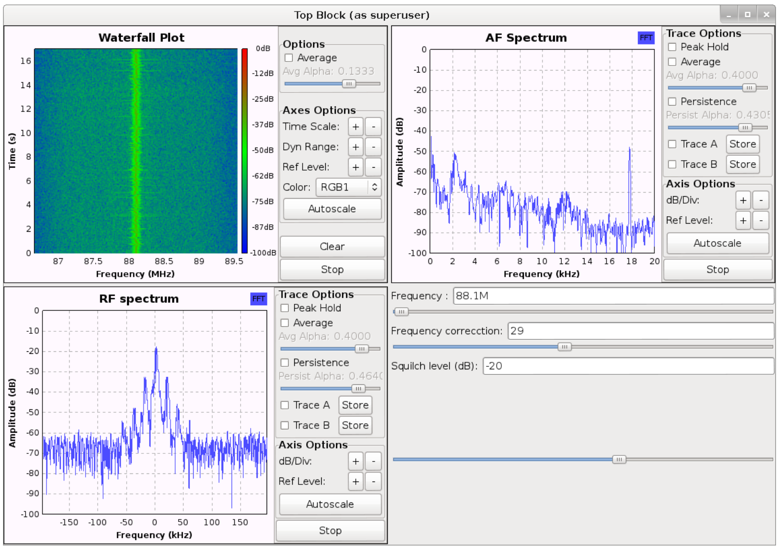
During this phase GNURadio has been carefully studied, and some modules has been developed (see appendix 1)

An agreement with ELAD allowed me to gather much better samplers, fully documented, with access to company engineers at the higher level: I revamped the old GNU Radio module and wrote new modules, so I have gained the knowledge to build the USB→UDP module.

SUPSI

Tools

GNU Radio as a reference and a test bed



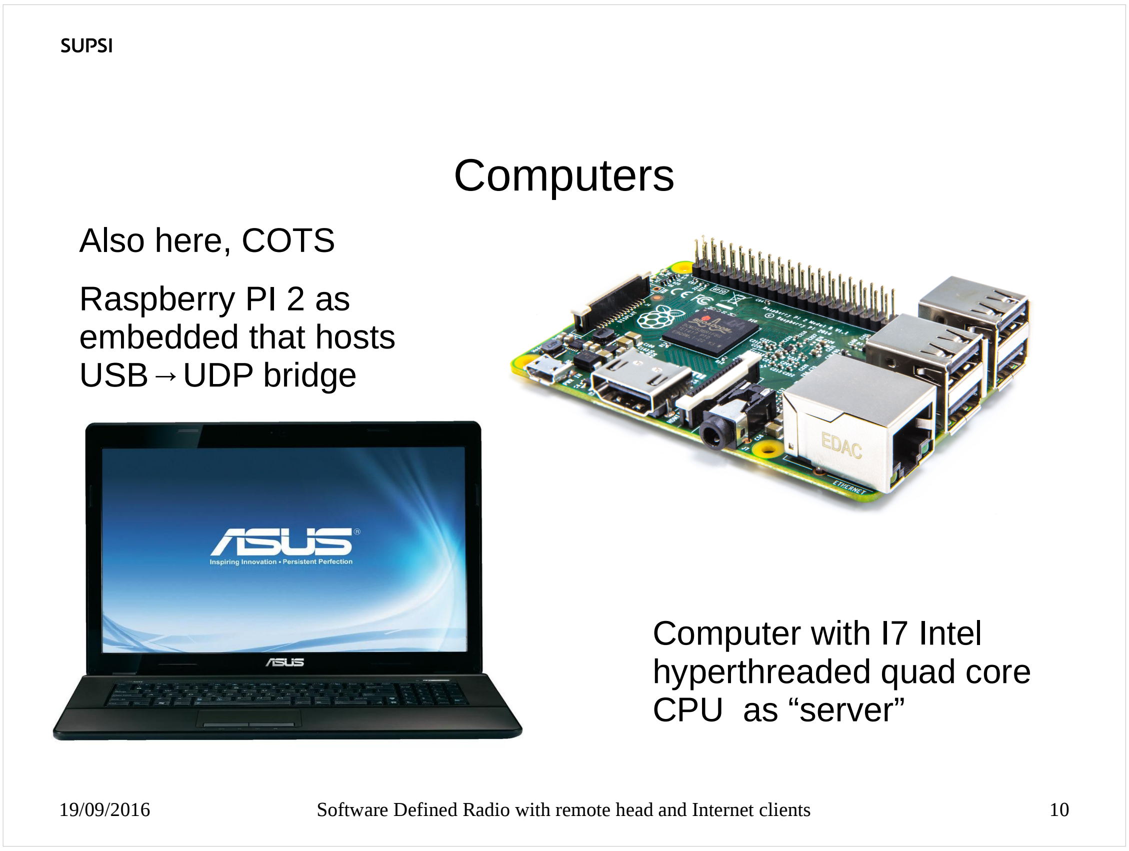
Frequency : 88.1M

Frequency correcction: 29

Squilch level (dB): -20

To verify the feasibility and to have a reference, I’ve used GNU Radio (mostly in the first part of work).

During the test of the first two modules, instead of client not yet developed, I’ve used GQRX that is based on GNU Radio.



SUPSI

Computers

Also here, COTS

Raspberry PI 2 as embedded that hosts USB→UDP bridge

Computer with I7 Intel hyperthreaded quad core CPU as “server”

19/09/2016 Software Defined Radio with remote head and Internet clients 10

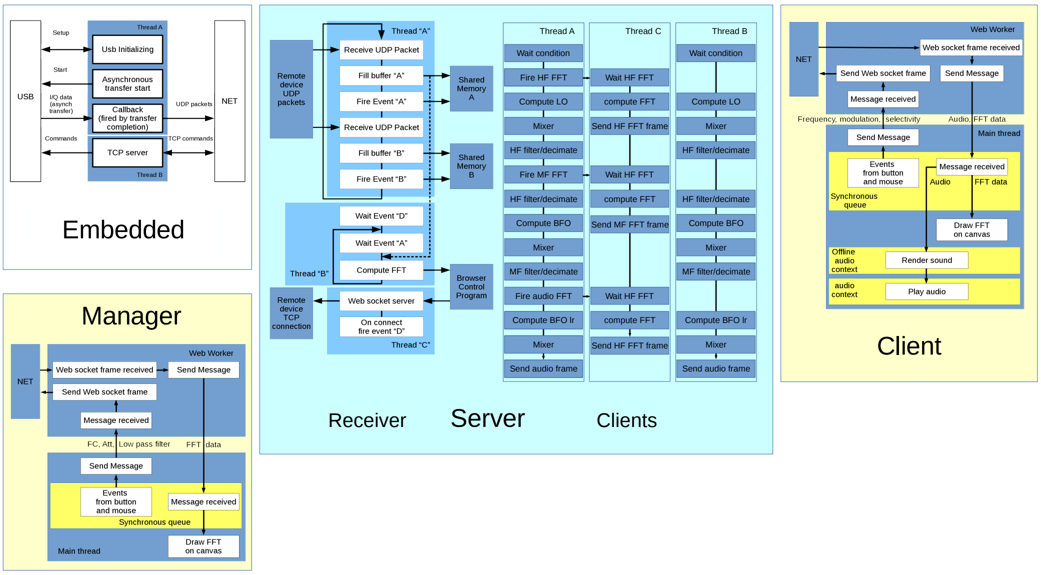
I’ve used Raspberry PI for a lot of projects and I realized that its computation power could be enough.

A series of measurements confirm my assumption, after some measures, I only increase the number of USB reception buffers from 2 to 4 to avoid any risk of data loss.

As server I used my laptop: it is old, but not so outdated, it could be used as reference for a little server, for one OM personal station (I’ve succesfully run 3 clients) and a state-of-the-art server could manage many more clients.

SUPSI

Software



USB

Setup

Start

I/Q data (asynch transfer)

Commands

Thread A

Usb Initializing

Asynchronous transfer start

Callback (fired by transfer completion)

TCP server

Thread B

UDP packets

TCP commands

NET

Embedded

Remote device UDP packets

Thread "A"

Receive UDP Packet

Fill buffer "A"

Fire Event "A"

Receive UDP Packet

Fill buffer "B"

Fire Event "B"

Shared Memory A

Shared Memory B

Wait Event "D"

Wait Event "A"

Compute FFT

Thread "B"

Browser Control Program

Remote device TCP connection

Web socket server

On connect fire event "D"

Thread "C"

Thread A	Thread C	Thread B
Wait condition		Wait condition
Fire HF FFT	Wait HF FFT	
Compute LO	compute FFT	Compute LO
Mixer	Send HF FFT frame	Mixer
HF filter/decimate		HF filter/decimate
Fire MF FFT	Wait HF FFT	
HF filter/decimate	compute FFT	HF filter/decimate
Compute BFO	Send MF FFT frame	Compute BFO
Mixer		Mixer
MF filter/decimate		MF filter/decimate
Fire audio FFT	Wait HF FFT	
Compute BFO lr	compute FFT	Compute BFO lr
Mixer	Send HF FFT frame	Mixer
Send audio frame		Send audio frame

Receiver

Server

Clients

NET

Web Worker

Web socket frame received

Send Web socket frame

Send Message

Message received

Frequency, modulation, selectivity

Audio, FFT data

Main thread

Send Message

Events from button and mouse

Message received

Audio

FFT data

Synchronous queue

Draw FFT on canvas

Offline audio context

Render sound

audio context

Play audio

Client

Manager

NET

Web Worker

Web socket frame received

Send Message

Send Web socket frame

Message received

FC, Att, Low pass filter

FFT data

Send Message

Events from button and mouse

Message received

Synchronous queue

Main thread

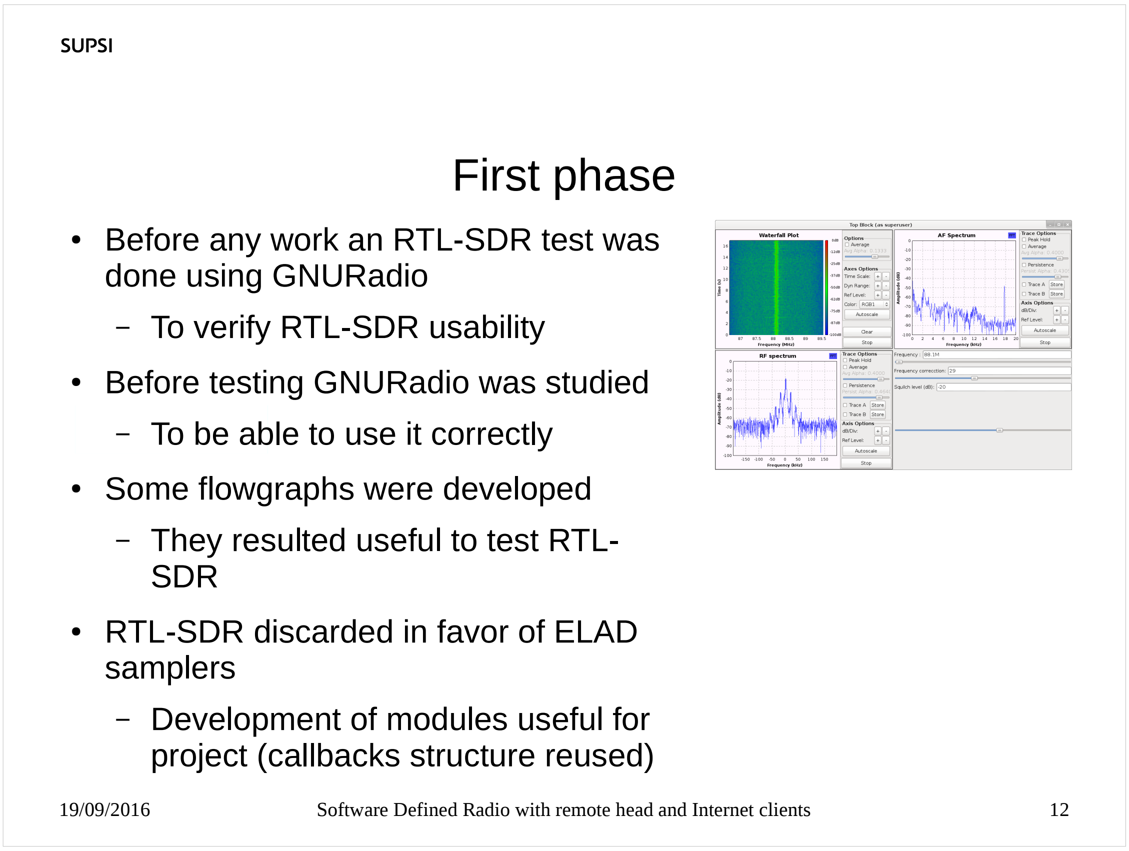
Draw FFT on canvas

This is a panoramic view of the software developed.

The yellow areas are inside a web browser.

Some new development in "web socket", "web worker", "audio context" was necessary to use efficiently the web browser (development done in javascript).

There is a lot of work made to convert analogic heterodyne to DSP algorithms.



SUPSI

First phase

- Before any work an RTL-SDR test was done using GNURadio
  - To verify RTL-SDR usability
- Before testing GNURadio was studied
  - To be able to use it correctly
- Some flowgraphs were developed
  - They resulted useful to test RTL-SDR
- RTL-SDR discarded in favor of ELAD samplers
  - Development of modules useful for project (callbacks structure reused)

19/09/2016 Software Defined Radio with remote head and Internet clients 12

Work with GNURadio has taken more than 4 months.

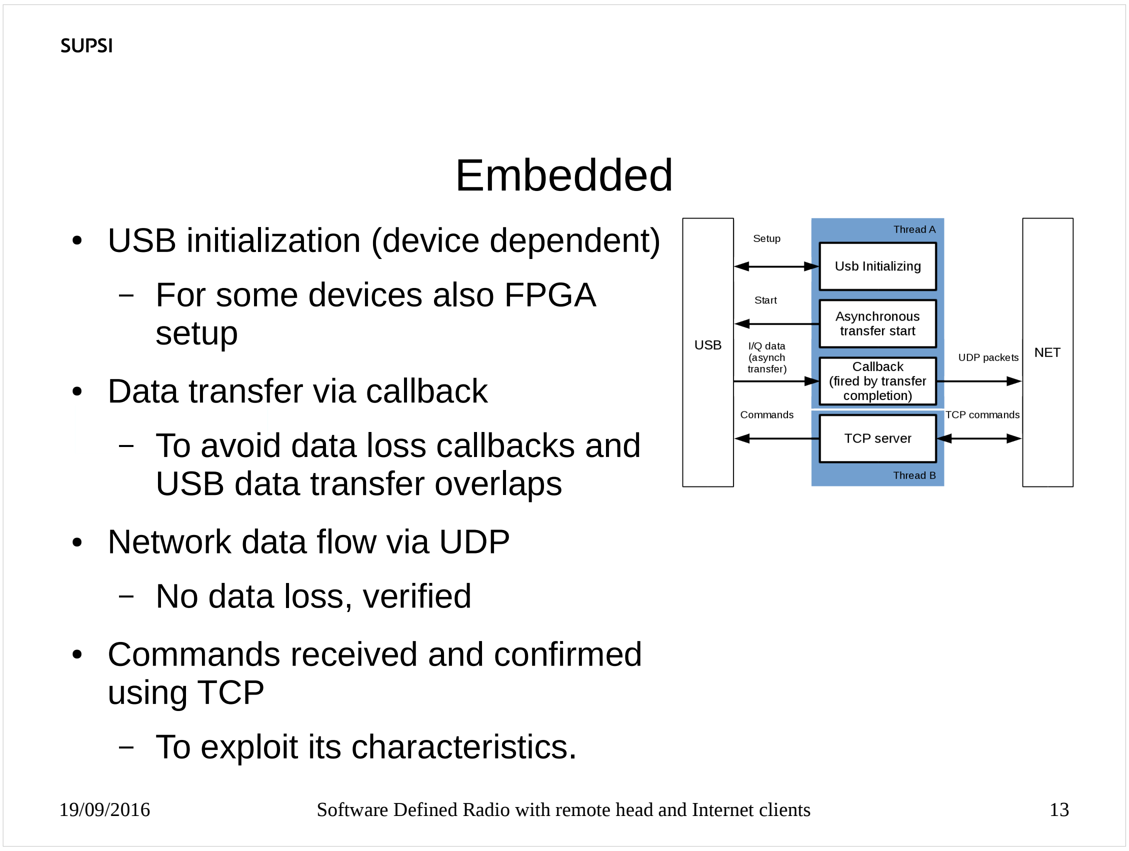
It was fundamental to test RTL-SDR dongles (and to discard them).

Once seen ELAD samplers, a module to link them with GNURadio was developed, with support of ELAD and using as a reference an old module developed by CSI Piemonte.

The new module allows some tests of ELAD samplers usage.

The new module has been used as reference for USB async data transfer transfer.

All this work is listed in appendixes 1 and 2.



SUPSI

Embedded

- USB initialization (device dependent)
  - For some devices also FPGA setup
- Data transfer via callback
  - To avoid data loss callbacks and USB data transfer overlaps
- Network data flow via UDP
  - No data loss, verified
- Commands received and confirmed using TCP
  - To exploit its characteristics.

USB
Setup
Start
I/Q data (asynch transfer)
Commands
Thread A
Usb Initializing
Asynchronous transfer start
Callback (fired by transfer completion)
TCP server
Thread B
UDP packets
TCP commands
NET

The first part developed is Device initialization via USB, then asynch data transfer.

Nothing really new, core tool is libusb-1.0

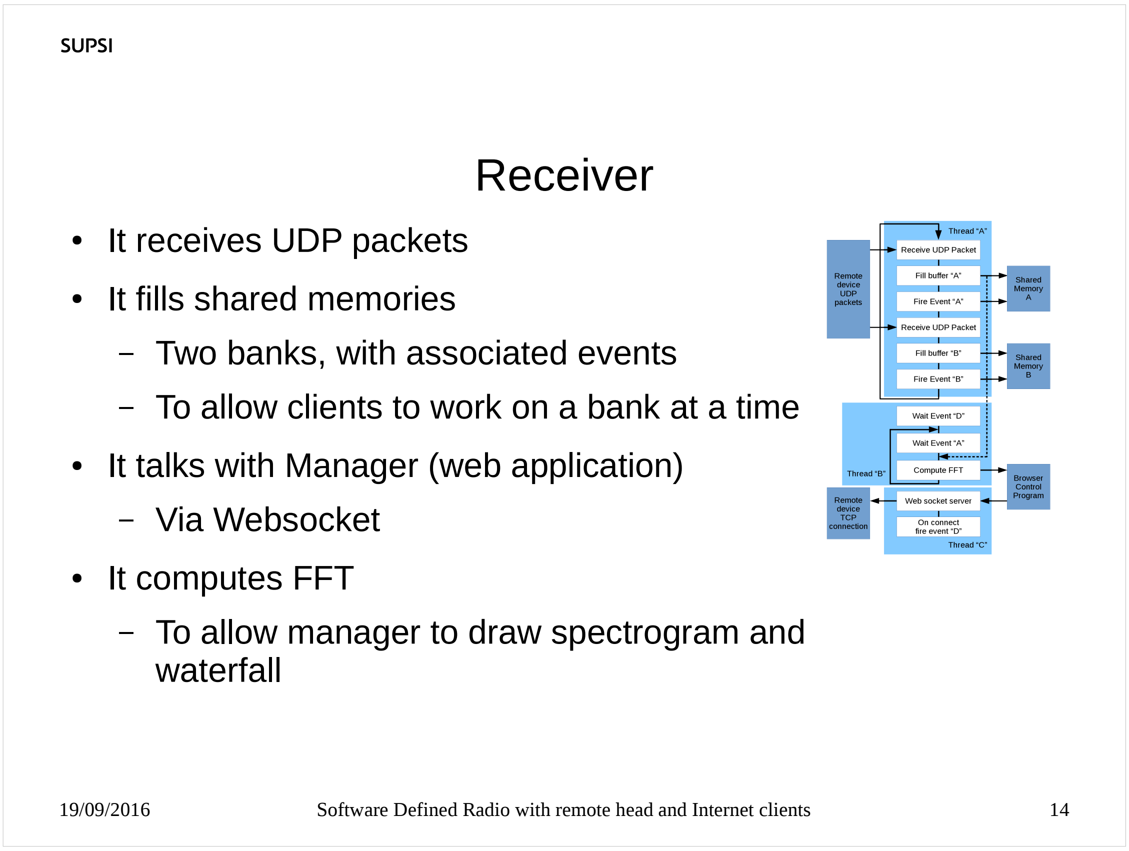
Data flow via UDP, TCP accumulates delays.

OS tailored to run on a “read-only” SD card.

Used threads to implement concurrency between USB and TCP control communications.

All implementations are developed by me, few libraries used (libusb-1.0, libpthread, libm, libc).

All-in-one source code, to enhance readability.



SUPSI

Receiver

- It receives UDP packets
- It fills shared memories
  - Two banks, with associated events
  - To allow clients to work on a bank at a time
- It talks with Manager (web application)
  - Via Websocket
- It computes FFT
  - To allow manager to draw spectrogram and waterfall

19/09/2016 Software Defined Radio with remote head and Internet clients 14

Still nothing really new.

Different threads to shape parallelism, condition variables to synchronize threads.

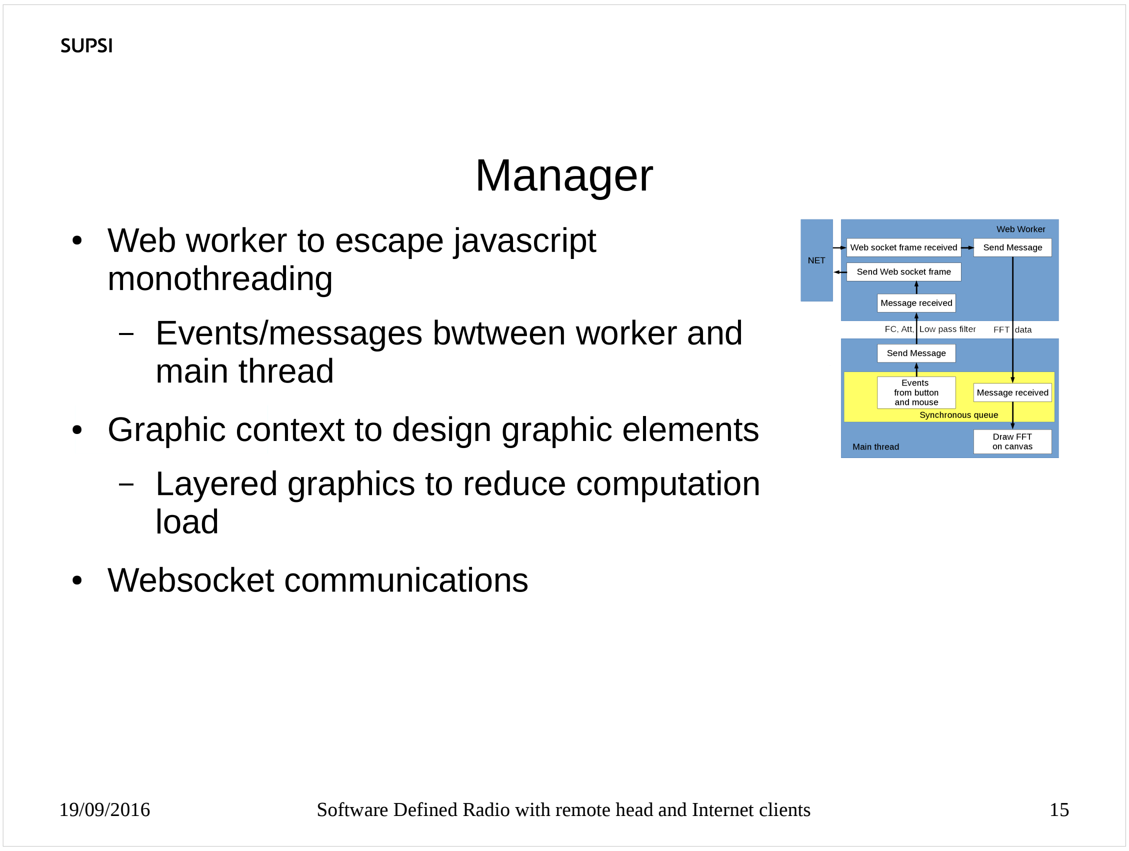
Shared memories filled with data to be processed by clients, condition variables on shared data to synchronize client-threads (via broadcast).

Websocket newly coded by me to allow operations as simple as possible.

FFT computed to send its data to Manager.

As few as possibile libraries (pthread, lssl, crypto).

All packed in a single C source file to enhance readability.



SUPSI

Manager

- Web worker to escape javascript monothreading
  - Events/messages bwtween worker and main thread
- Graphic context to design graphic elements
  - Layered graphics to reduce computation load
- Websocket communications

NET
Web Worker
Web socket frame received
Send Message
Send Web socket frame
Message received
FC, Att, Low pass filter
FFT data
Send Message
Events from button and mouse
Message received
Synchronous queue
Main thread
Draw FFT on canvas

19/09/2016 Software Defined Radio with remote head and Internet clients 15

Written in Javascript, that is interpreted in a single thread.

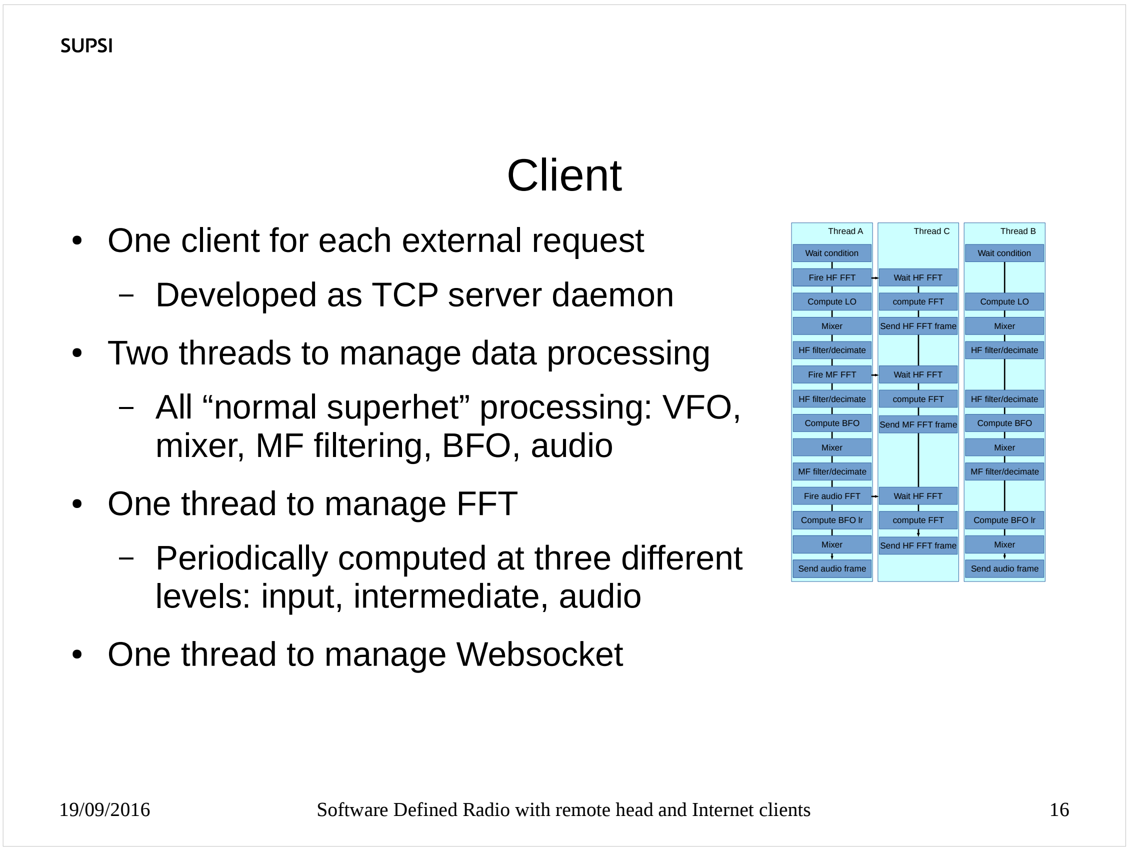
Use of web worker to use an other thread.

Web worker implements Websocket (very simple in Javascript)

Messages to communicate between two threads.

Use of graphic context to draw spectrograms.

Two programs: the main one (index.html) and a separate web worker (web workers request a separate javascript source)



Client

- One client for each external request
  - Developed as TCP server daemon
- Two threads to manage data processing
  - All “normal superhet” processing: VFO, mixer, MF filtering, BFO, audio
- One thread to manage FFT
  - Periodically computed at three different levels: input, intermediate, audio
- One thread to manage Websocket

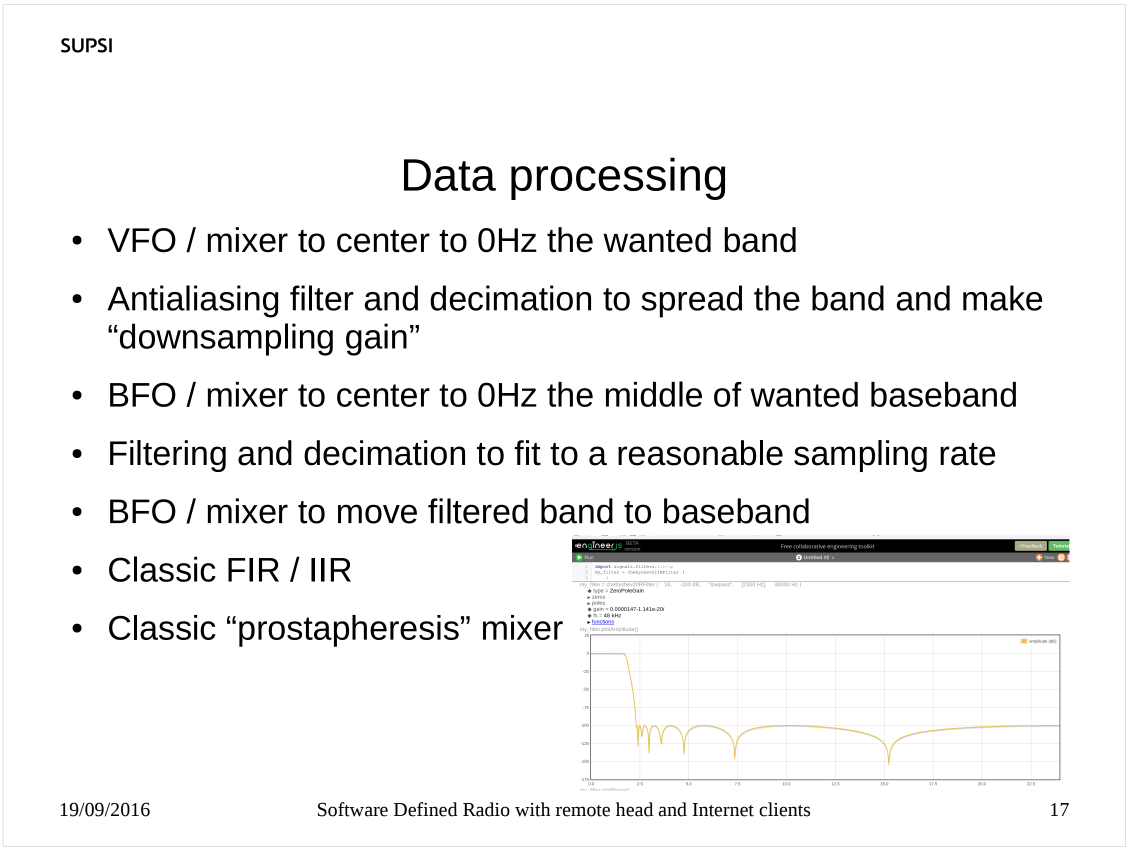
Client is a daemon that implements a TCP server.

Once connected, launch a WebSocket server, two data processing threads, and a thread that computes FFTs.

Websocket and FFT use the same code used in the other programs, no functions, no libraries for sake of semplicity.

The data processing threads mimic a “superhet” receiver (see next slide) and are started by shared memory condition variable.

FFTs are computed and sent to client web browser that draws spectrograms to help operator to tune the frequency.



SUPSI

Data processing

- VFO / mixer to center to 0Hz the wanted band
- Antialiasing filter and decimation to spread the band and make “downsampling gain”
- BFO / mixer to center to 0Hz the middle of wanted baseband
- Filtering and decimation to fit to a reasonable sampling rate
- BFO / mixer to move filtered band to baseband
- Classic FIR / IIR
- Classic “prostapheresis” mixer

```
import signals.filters.iir-1;
my_filter = chebyshev2IIRFilter (
    4
```

```
my_filter = chebyshev2IIRFilter (    16,    -100 dB,    "lowpass",    [2300 Hz],    48000 Hz )
  ◆ type = ZeroPoleGain
  ▸ zeros
  ▸ poles
  ◆ gain = 0.0000147-1.141e-20i
  ◆ fs = 48 kHz
  ▸ functions
my_filter.plotAmplitude()
```

19/09/2016 Software Defined Radio with remote head and Internet clients 17

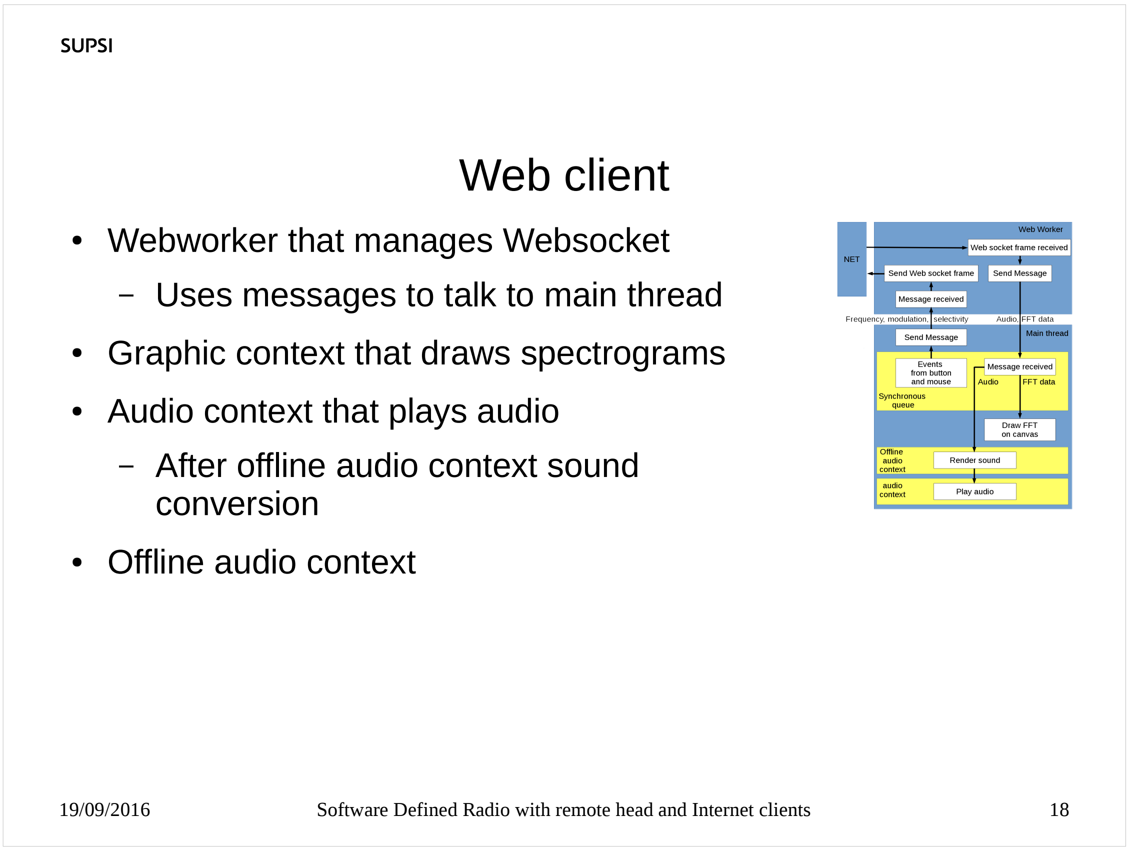
This is not a telecomm thesis so the data processing section is quite “classic”.

The reference is a superhet receiver with the main difference that digital filters are implemented around 0Hz, so a triple conversion is needed.

The oscillators are waveforms generated a bank at a time, using prostapheresys formulas to avoid trigonometric functions usage. An amplitude compensation algorithm is used.

Also mixers uses the prostapheresis formula: having I and Q signals this is straigtforward.

Filters are classical FIR and IIR.



SUPSI

Web client

- Webworker that manages Websocket
  - Uses messages to talk to main thread
- Graphic context that draws spectrograms
- Audio context that plays audio
  - After offline audio context sound conversion
- Offline audio context

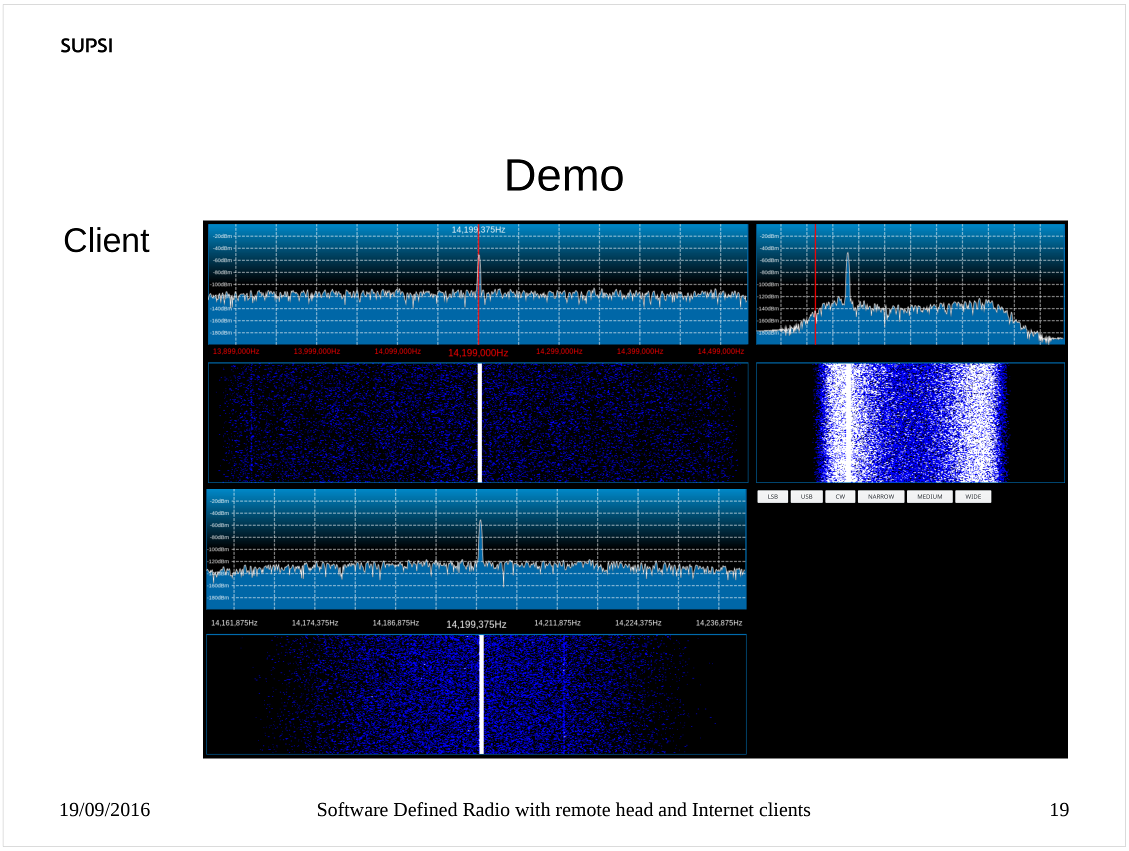
19/09/2016 Software Defined Radio with remote head and Internet clients 18

The web client has many things in common with Manager: web worker, web socket, graphic context

The main difference is the audio management: audio arrives at 24kS/s (audio context manages 20.5kS/s to 48kS/s) and it is converted to the native audio output samplerate (not manageable) by an offline audio context, than it is used by the audio context.

The audio context uses a “processing” block that is fired when the audio card needs data: at that time it reads the samples produced by the offline audio context.

There are callbacks triggered by clicks on spectrograms and by the mouse wheel



SUPSI

Demo

Client

19/09/2016 Software Defined Radio with remote head and Internet clients 19

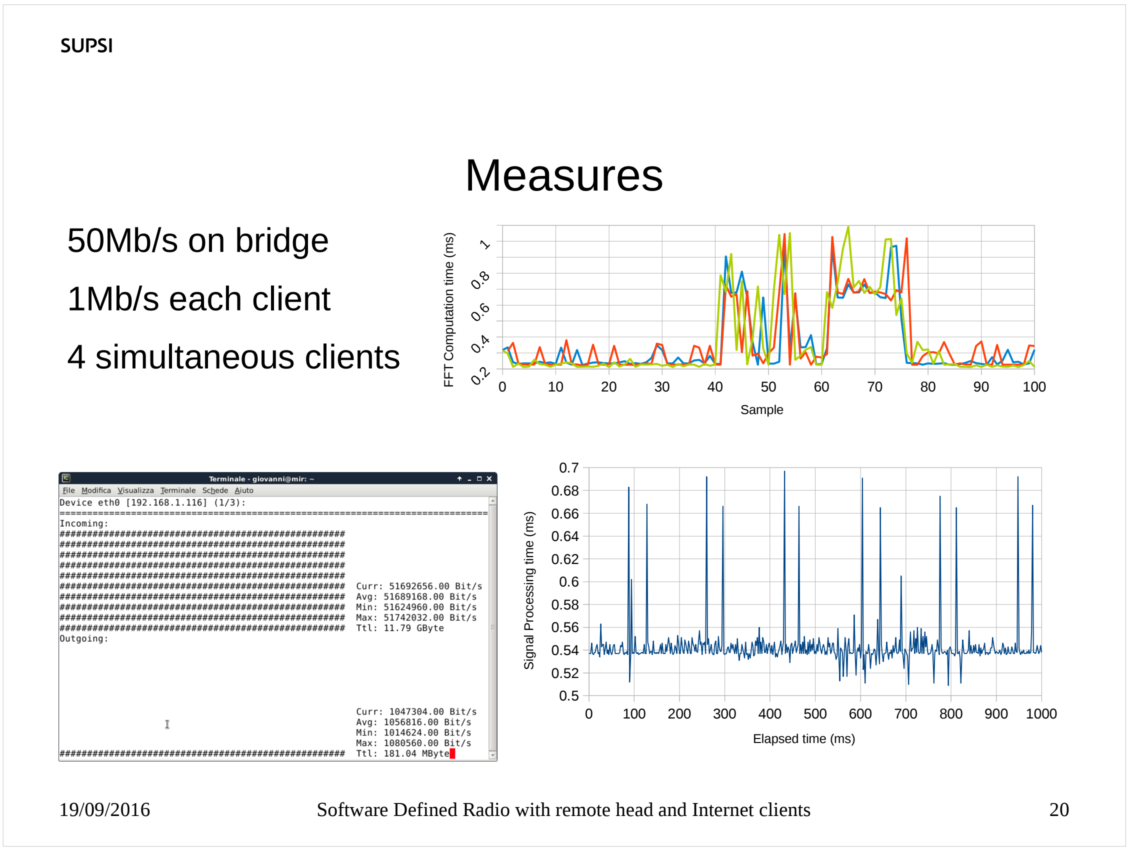
Here we can watch a quick demo of the project.

To make simple the demo I use a small signal generator.

See the click and point freqency set and, also, the mouse weel usage.

There is a lot of space for other buttons and functionality.

The client is far from complete, but it is born as a proof of concept.



SUPSI

Measures

50Mb/s on bridge

1Mb/s each client

4 simultaneous clients

```
Device eth0 [192.168.1.116] (1/3):
============================================================================
Incoming:
###################################################
###################################################
###################################################
###################################################
###################################################
###################################################  Curr: 51692656.00 Bit/s
###################################################  Avg: 51689168.00 Bit/s
###################################################  Min: 51624960.00 Bit/s
###################################################  Max: 51742032.00 Bit/s
###################################################  Ttl: 11.79 GByte
Outgoing:




                                                     Curr: 1047304.00 Bit/s
                                                     Avg: 1056816.00 Bit/s
                                                     Min: 1014624.00 Bit/s
                                                     Max: 1080560.00 Bit/s
###################################################  Ttl: 181.04 MByte
```

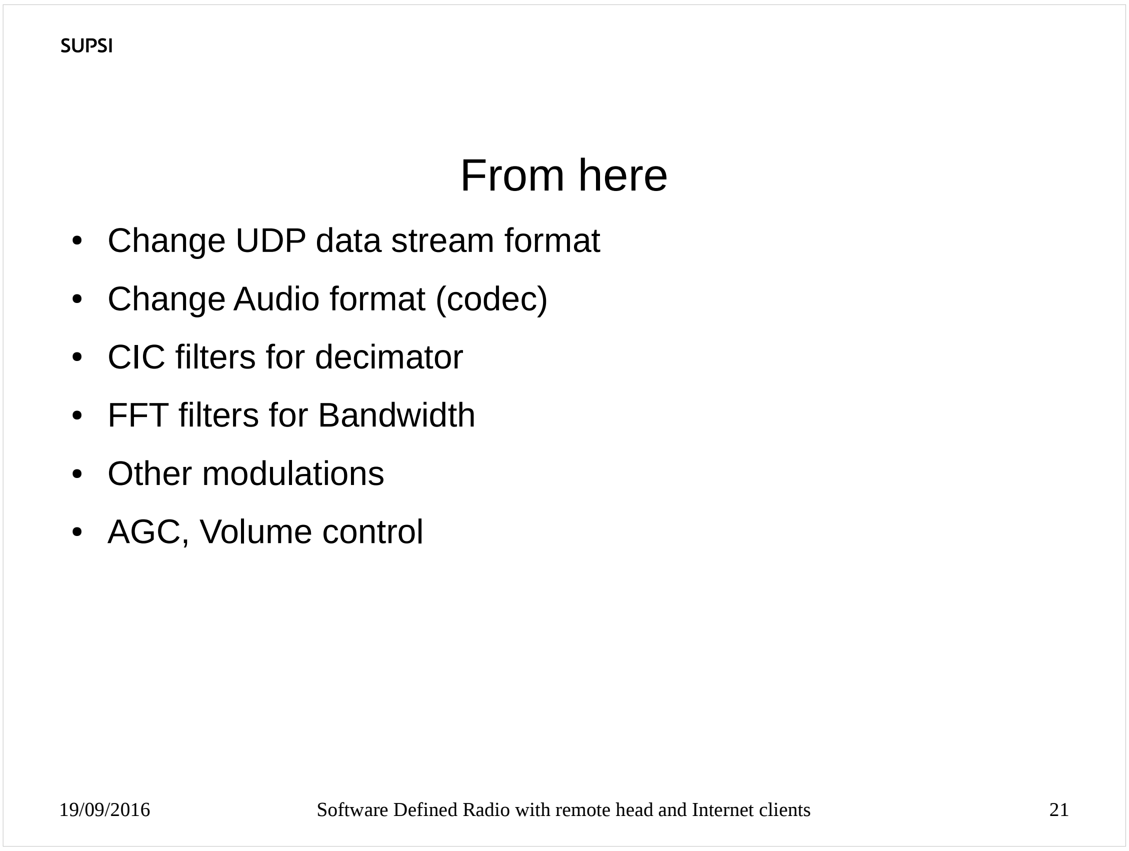
19/09/2016 Software Defined Radio with remote head and Internet clients 20

Network traffic in input (UDP) has an acceptable value (it could also be lowered by changing UDP packet format).

The packet loss monitoring shows no UDP packet loss.

Output network traffic could be seen as a bit higher, but nowadays network connections are far higher (mine is 50/500 allowing tens of output streams).

Elaboration times, even with discontinuity, fall in an acceptable range.



SUPSI

From here

- Change UDP data stream format
- Change Audio format (codec)
- CIC filters for decimator
- FFT filters for Bandwidth
- Other modulations
- AGC, Volume control

19/09/2016 Software Defined Radio with remote head and Internet clients 21

It could be interesting to make a certain amount of changes, suggested by measures and evalutation, to build a real product (idea for a new work?).