SpectrumWars

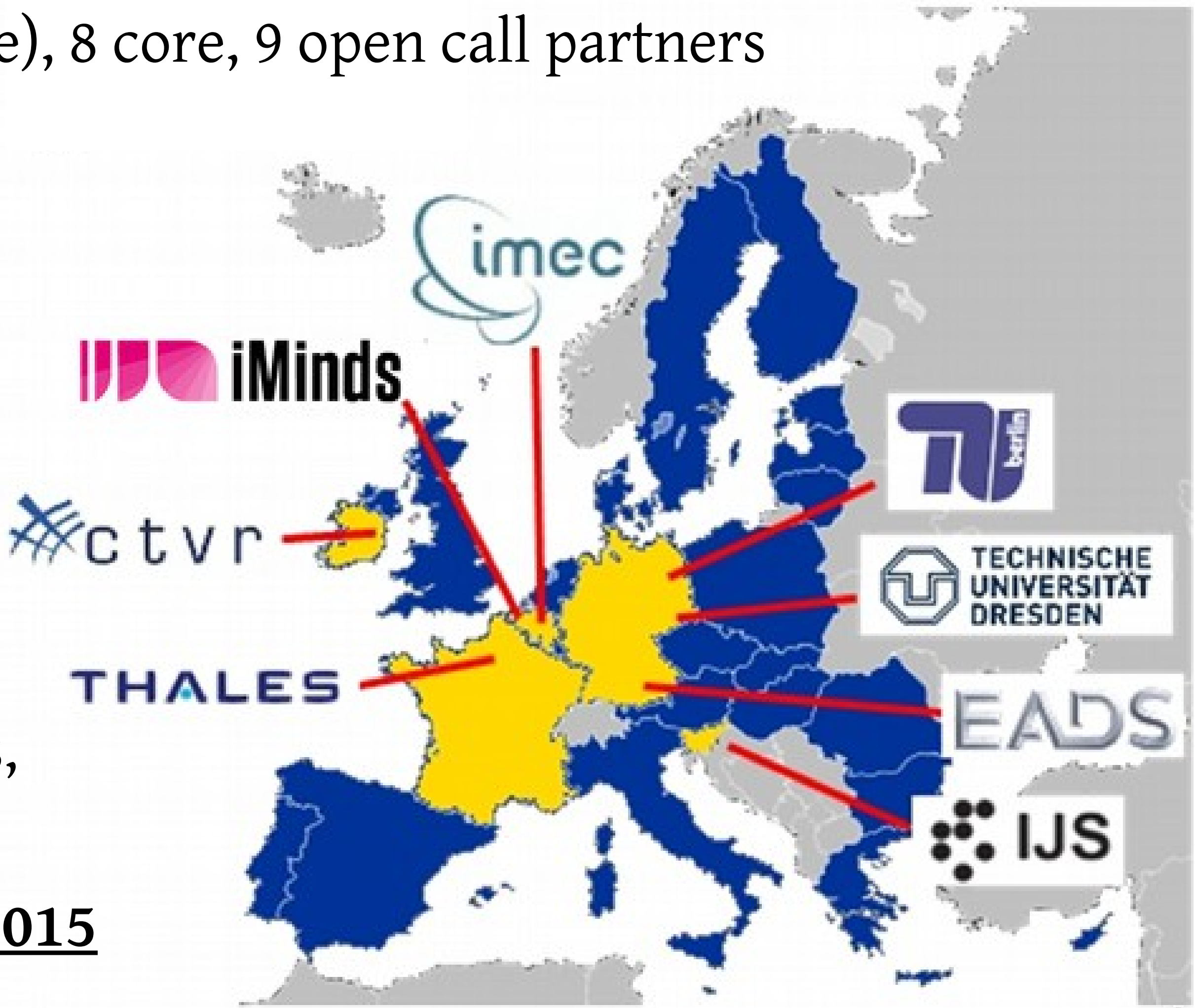
»A programming game where players compete for bandwidth on a limited piece of a radio spectrum«

Tomaž Šolc
tomaz.solc@ijs.si

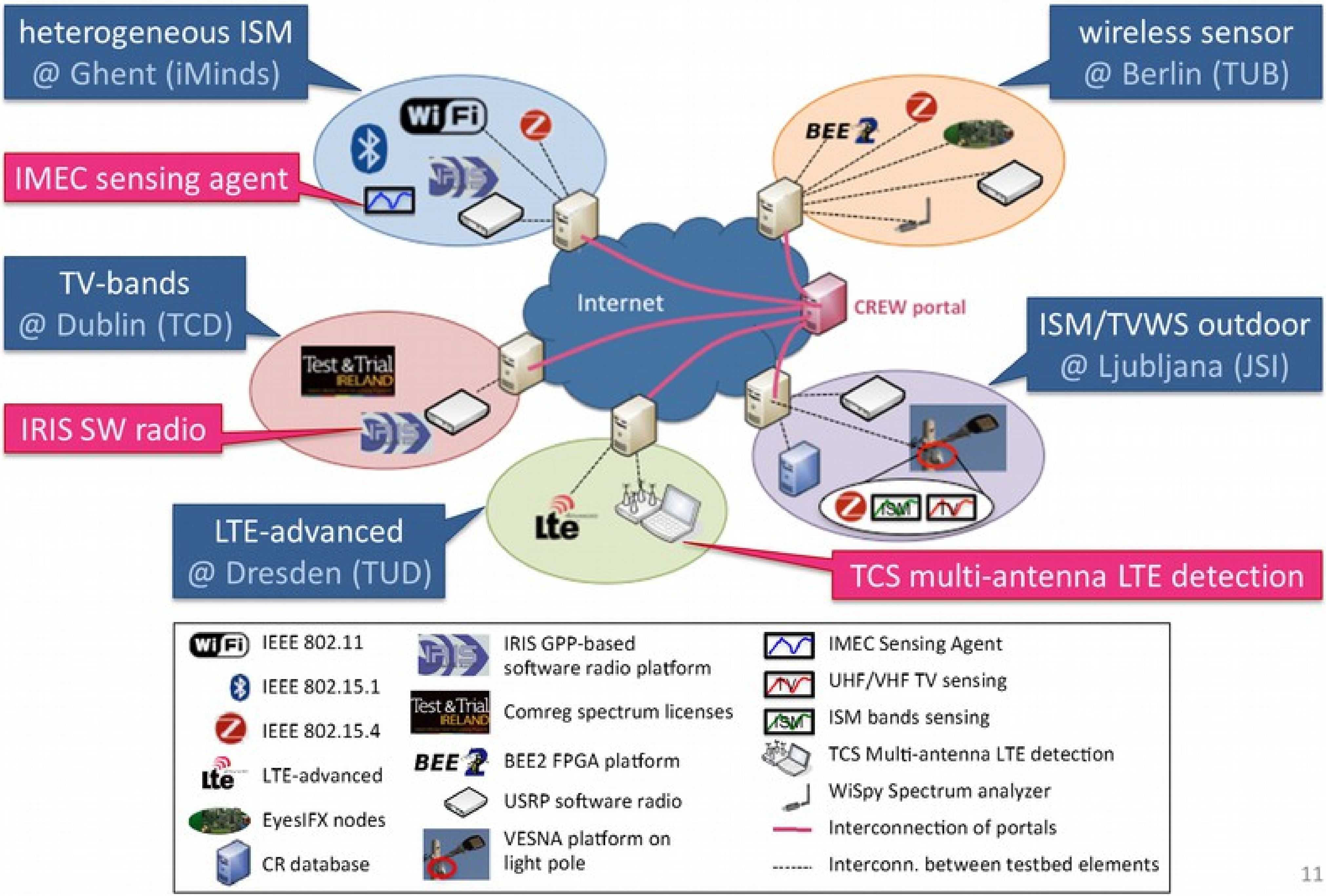
Context

IP CREW

- **Cognitive Radio Experimentation World**
  - Facilitate research into cognitive radio by establishing an open federated test platform.
  - FP7 call 5 (FIRE Initiative), 8 core, 9 open call partners
  - **NOT** doing research nor developing new algorithms
  - Developing facilities for supporting research
  - Augmenting existing facilities
  - Offering better methods, validating solutions
  - <u>**Project ends October 2015**</u>

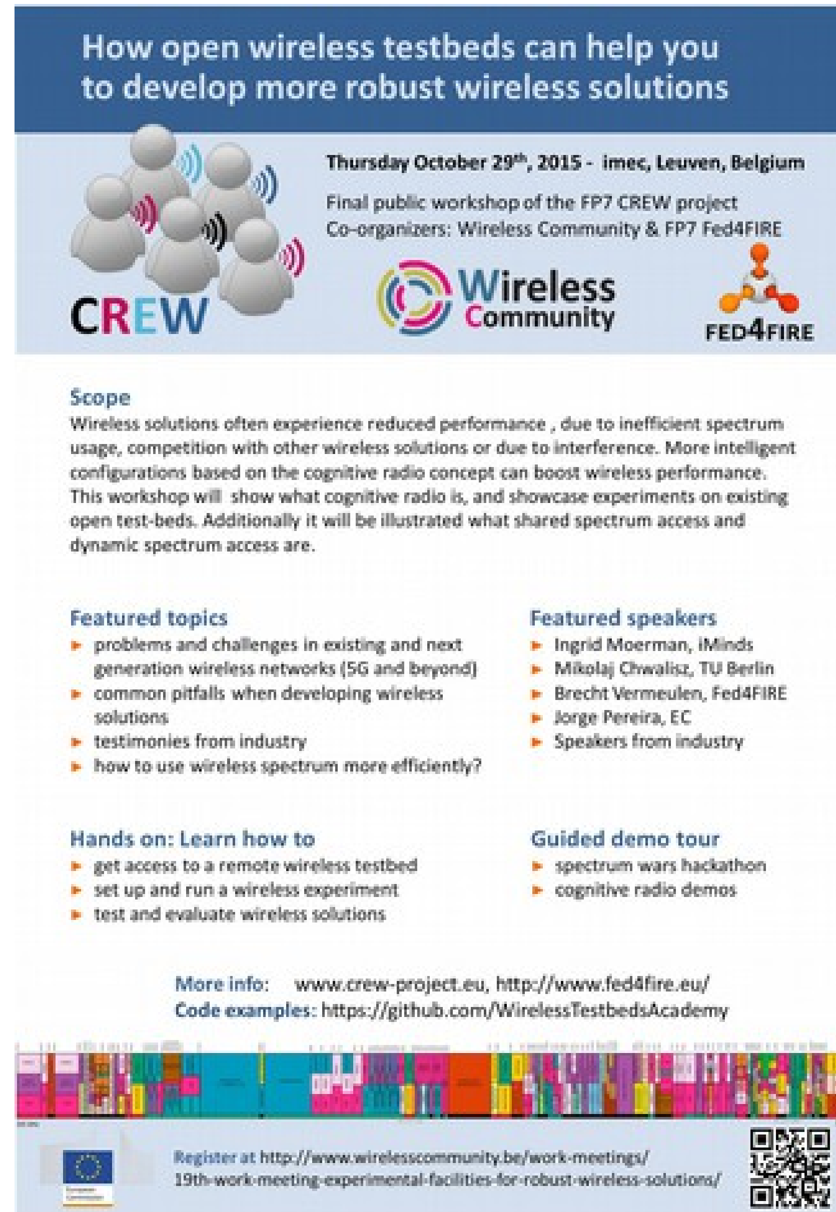


CREW platform



CREW workshop

»The public workshop will be co-organized with the wireless community from Belgium and will showcase CREW demos as well as spectrum competitions and games.«



eWINE

- »Elastic Wireless Networking Experimentation«
- **Project proposal submitted in April 2015**
- »Increase awareness and uptake by running directed open calls in the form of grand challenges or competitions«
- The eWINE Grand Challenge.
  - Targets the research community and highly trained industrial professionals
  - It will be based on existing models such as the DARPA Spectrum Challenge

Idea

DARPA Spectrum Challenge

- *»demonstrate how a radio can use a channel in presence of other signals«*
- Send and receive 15 000 packets on a 5 MHz channel in the UHF band.
- 90 teams, 2 or 3 teams per match.
  - Adversarial: only your score counts, encouraged to interfere with others.
  - Co-existence: score of other players also counts, encourage cooperation.

http://spectrum.ieee.org/telecom/wireless/radio-wrestlers-fight-it-out-at-the-darpa-spectrum-challenge

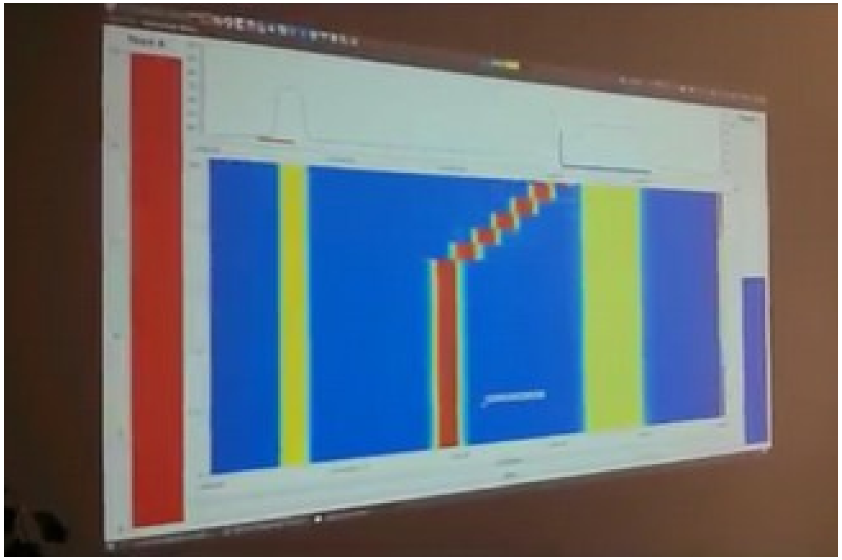
RobotWar

- A 1970s programming game (with many modern imitations)
- *»The task set before you is to program a robot that no other robot can destroy!«*

```
SCAN
  AIM + 5 TO AIM
  AIM TO RADAR
LOOP
  IF RADAR < 0 GOSUB FIRE
  GOTO SCAN
FIRE
  0 - RADAR TO SHOT
  ENDSUB
```

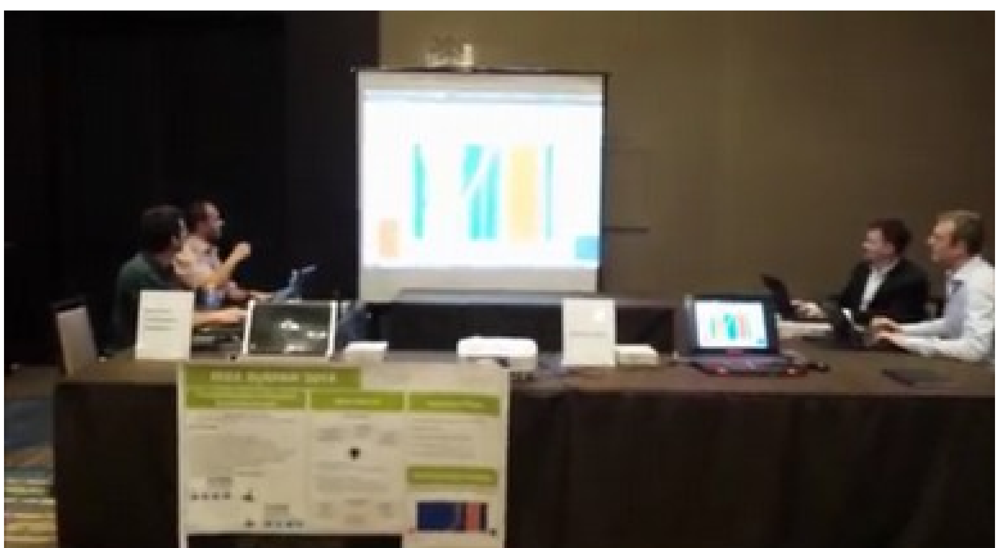
https://en.wikipedia.org/wiki/RobotWar

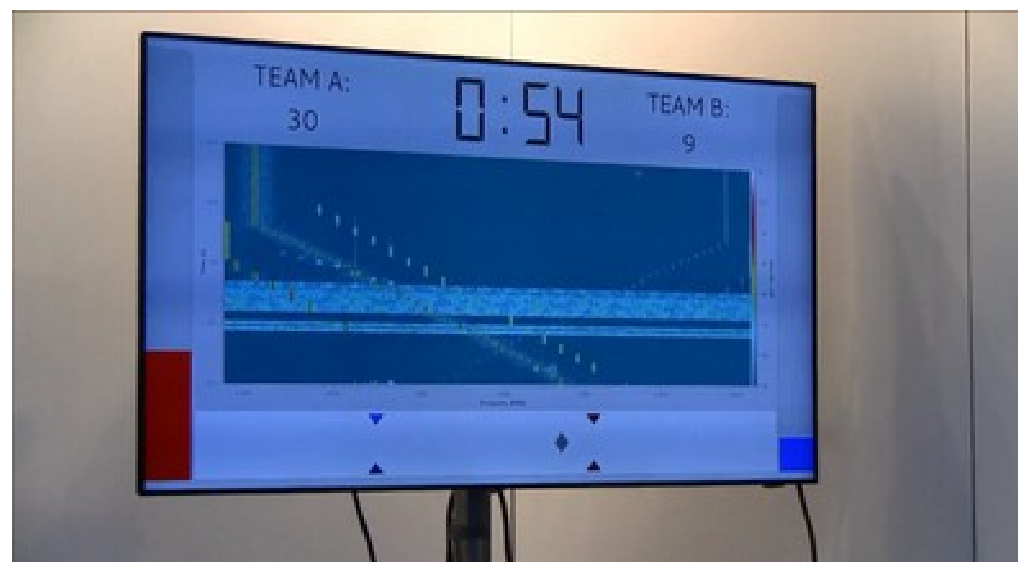
CTVR & iMinds SpectrumWars



IEEE DySpan 2014

Net Futures 2015





https://www.youtube.com/watch?v=pT7sUI8I7gI https://www.youtube.com/watch?v=m-Js5pwuVFA

CTVR & iMinds SpectrumWars

- Two teams of two (human) players
  - each team controls two transceivers:
    one receiver and one transmitter.
  - joystick interface, trigger enables transmission.
- There's a computer controlled interferer.
- A spectrum analyser shows the progress of the game in a waterfall plot which everyone can see.
- Game counts how many bytes were successfully transferred between the transmitter and receiver.
- Team that transmits most data in allotted time wins.

Replace human with a (short) script

- Have each player write two simple programs.
- Each program controls one transceiver.
  - both transceivers are identical, but for the purpose of the game, we call them »transmitter« and »receiver«.
- Player's goal is to efficiently transmit data from transmitter to receiver in the presence of:
  - other players
  - computer controlled interferers
  - simulated primary users

Implementation overview

Player's transceiver API

- Players write code in Python
  - not a toy language, but simple for beginners
- Event-based and procedural interfaces
- Very limited set of instructions
  - send, receive packet (with optional control data).
  - set frequency, transmit power, modulation bandwidth.
  - get power spectral density from the spectrum sensor.
  - hardware query (packet size, no. of channels, etc.)
- Hopefully hardware-independent

Hardware independence

- Parameters are integers without physical units.
  - frequency is defined as channel number (e.g. 0 - 63)
  - power, bandwidth settings, etc. unit-less integers
- Different hardware defines different ranges for frequency, etc.
- Mapping to physical units is in testbed documentation.

Interpretation of bandwidth settings¶

bandwidth	bitrate
0	50 kbps
1	100 kbps
2	200 kbps
3	400 kbps

Interpretation of transmission power settings¶

power	dBm
0	0
1	-2
2	-4
3	-6
4	-8
5	-10
6	-12
7	-14

Channel model

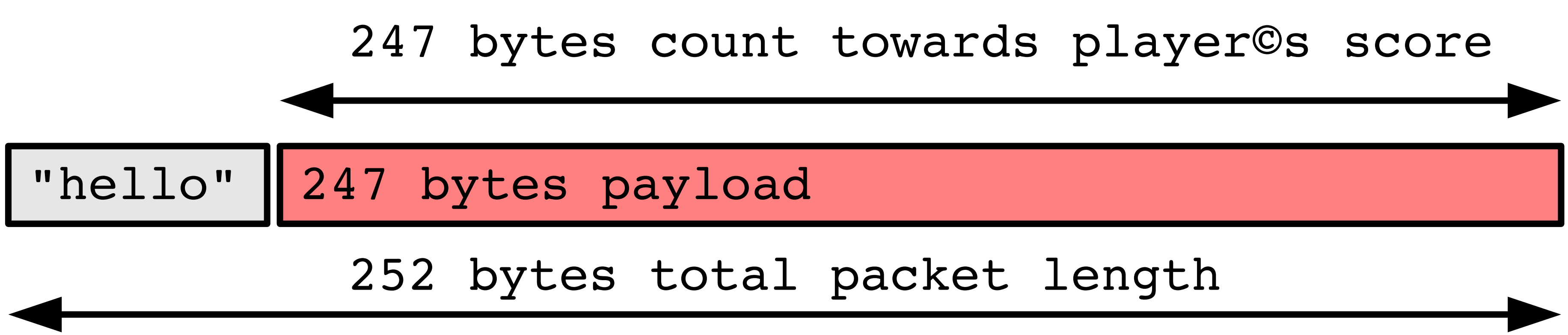
- Player's transmitter and receiver can exchange arbitrary control data over the air.
  - this is the only way the two ends can communicate (there is no reliable backchannel)
  - receiver **can** talk back to transmitter
  - both must be using same frequency, bandwidth
- Example: send next channel to hop to

```
self.send("hello")
```

```
for packet in self.recv_loop():
    # packet.data contains "hello"
    ...
```

Channel model

- However, player's goal is to transfer **payload** data
  - transfer of payload is transparent to the player
  - all bytes left in the packet after control data are automatically used for payload.
  - only TX->RX packets carry payload
- This encourages sparing use of control data.



Procedural TX implementation

```
class Transmitter(Transceiver):
    # This method is called upon the start of the game
    def start(self):

        # Obtain a list of available channels and
        # RSSI on each of them.
        spectrum = np.array(self.get_status().spectrum)

        # Tune to the channel with the minimum RSSI.
        ch = np.argmin(spectrum)
        self.set_configuration(ch, 0, 0)

        # send packets on the selected channel until
        # the end of the game.
        while True:
            self.send()
```

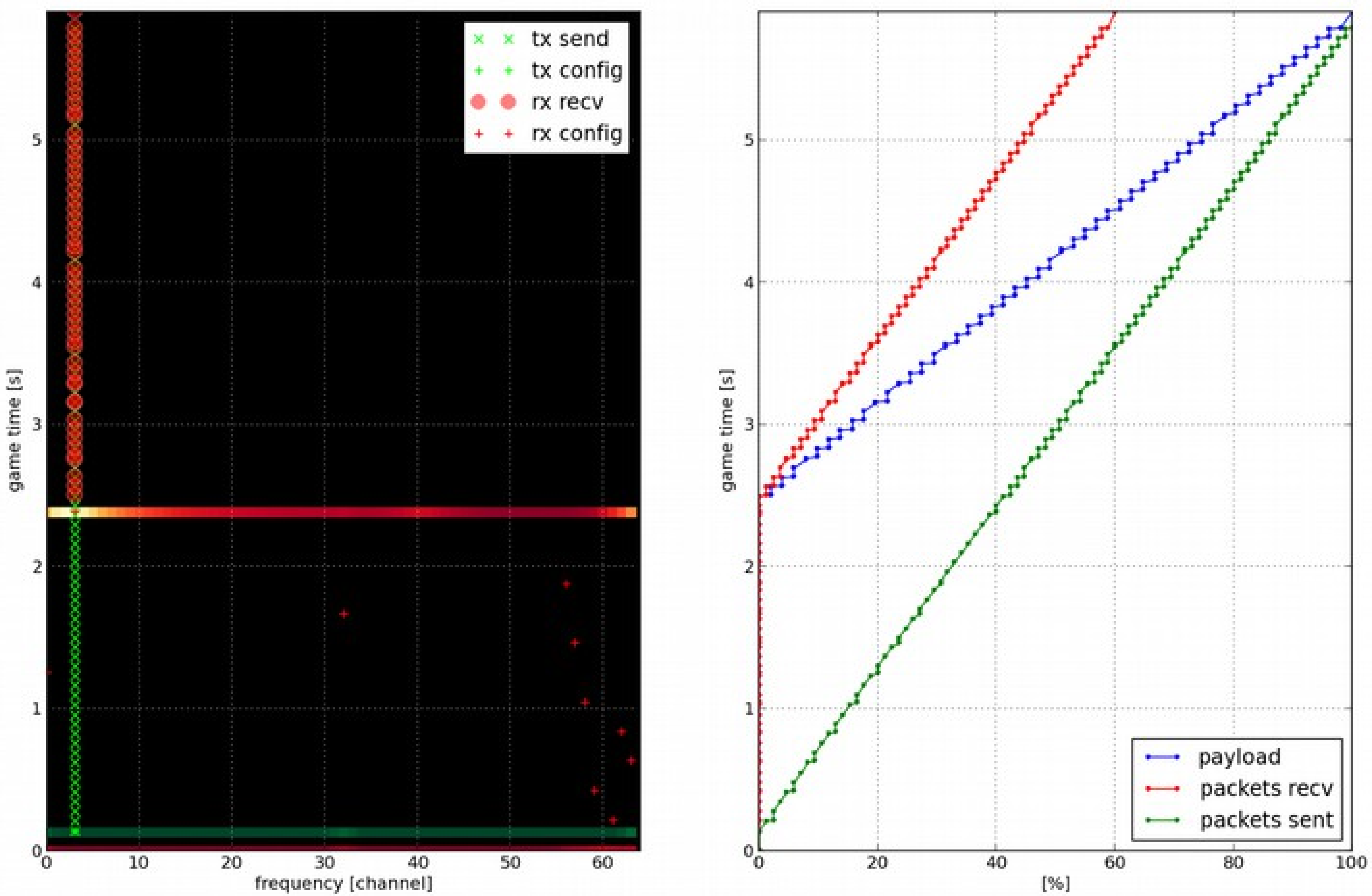
Procedural RX implementation

```
class Receiver(Transceiver):
    def start(self):
        while True:
            # Obtain RSSI for all available channels.
            spectrum = np.array(self.get_status().spectrum)

            # For 10 channels with the highest RSSI...
            chl = np.argsort(spectrum)[::-1]
            for ch in chl[:10]:
                # tune the radio to the channel...
                self.set_configuration(ch, 0, 0)

                # and wait 200ms for any packets.
                for packet in self.recv_loop(timeout=.2):
                    pass
```

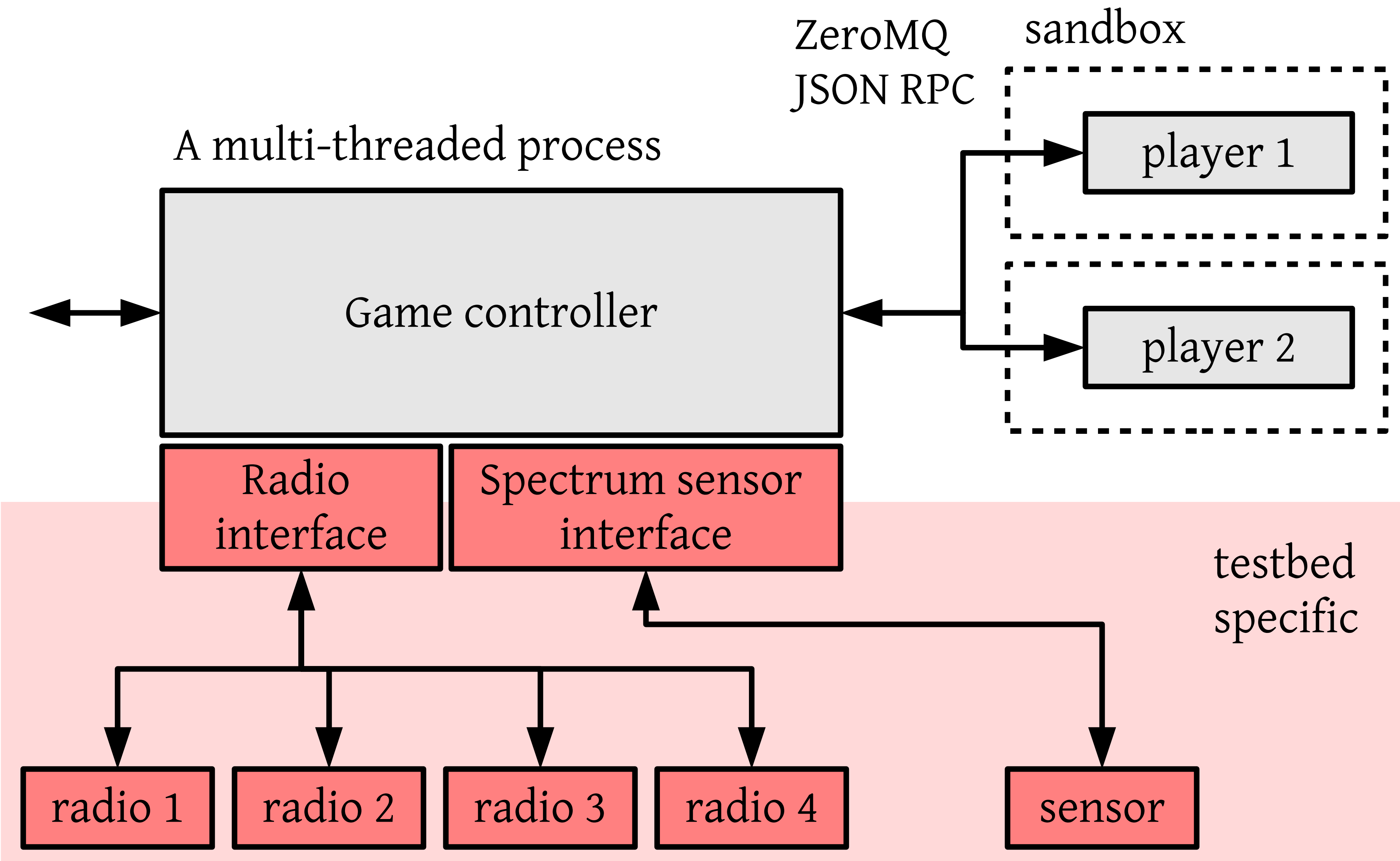
Example game



Scoring

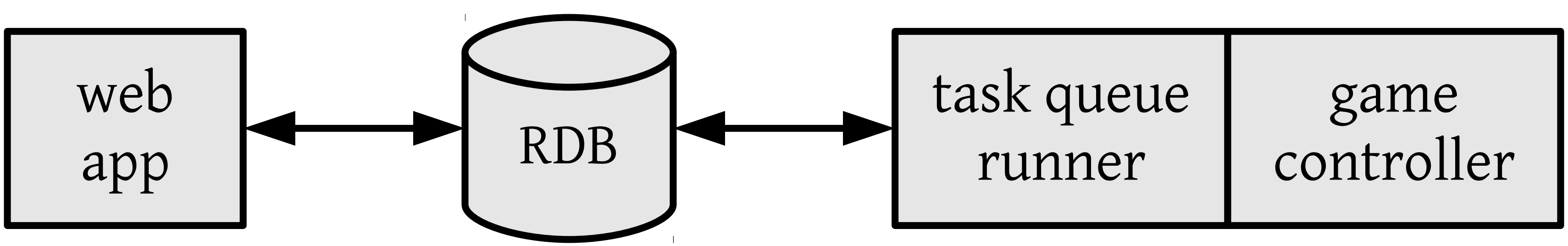
- Not finalized yet. Probably a weighted sum.
- Positive terms (more is better):
  - transferred payload
- Negative terms (less is better):
  - number of transmitted packets
  - time used to transfer payload
  - interferrence with primary user (if there is one)
  - estimated energy used (packets · TX power)
  - unhandled exceptions in the code.

Back-end architecture



Front-end architecture (WIP)

- Web interface
  - player registration
  - submit code
  - debugging info
  - scoreboard
- Web app queues games to play in a RDB
- Queue runner fetches games from the RDB
- Starts game controller and runs the game
- Writes game results back to RDB

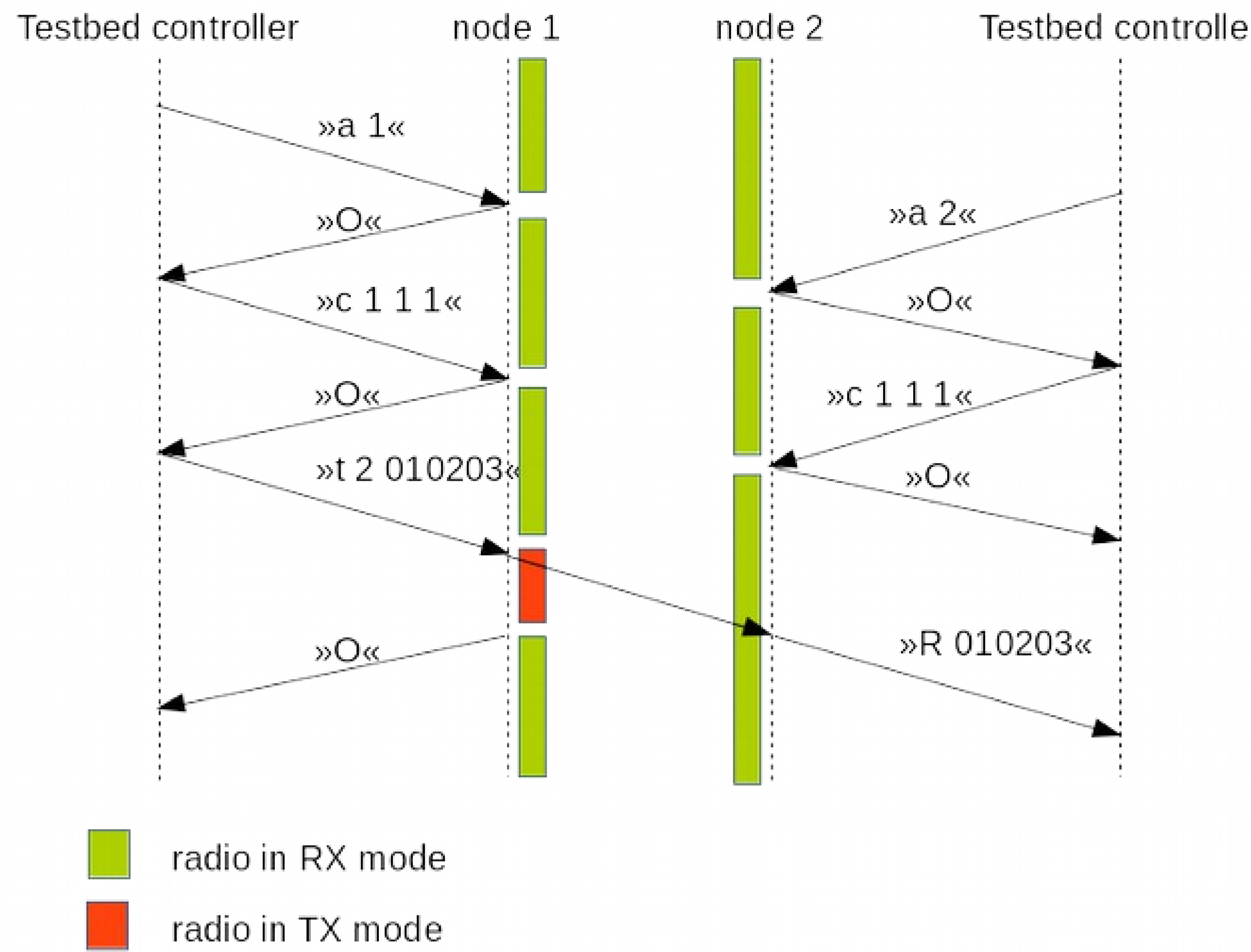


How it works with VESNA

VESNA testbed

- Backend (game controller and sandboxes) are running on one GNU/Linux host.
- One VESNA node with SNE-ISMTV-2400 (CC2500) forms one transceiver under player's control
- Sensor nodes are connected over a powered USB hub to the game controller
  - less wires than with separate RS-232 cables, USB converters and power cables.
  - USB-CDC support on VESNA is buggy.
- Firmware uses simple ASCII-based protocol.

VESNA firmware interface



Spectrum sensing

- USRP N200 with SBX daughterboard
- Duty cycle of CC2500 is <10%
- With any blind time the transmissions are nearly invisible.

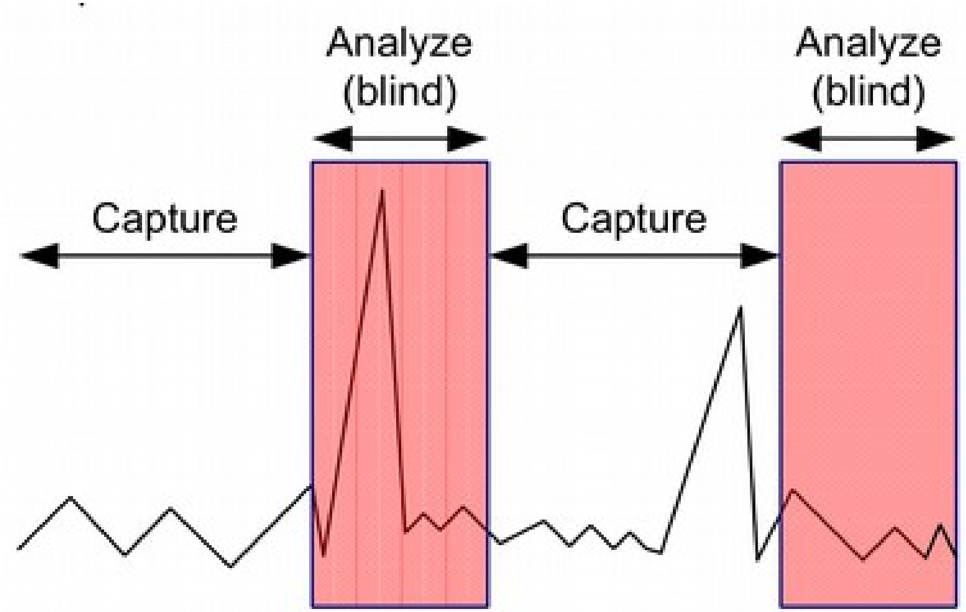
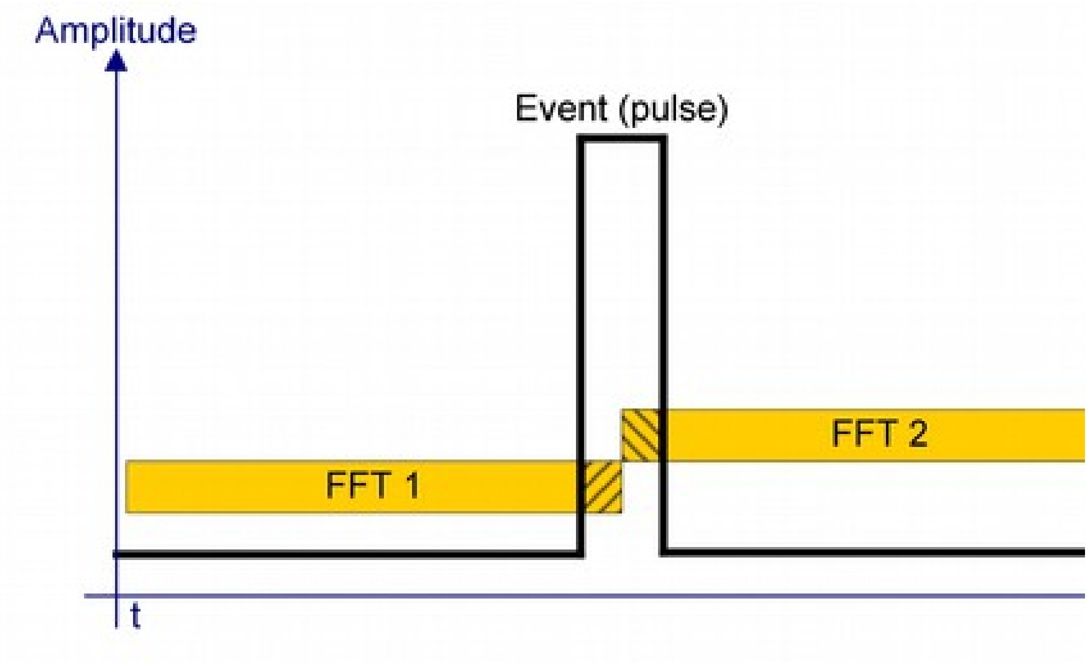


Figure 1: Sequential capture and analysis as used in e.g. FFT analyzers

»Implementation of real-time spectrum analysis«, Rohde&Schwarz

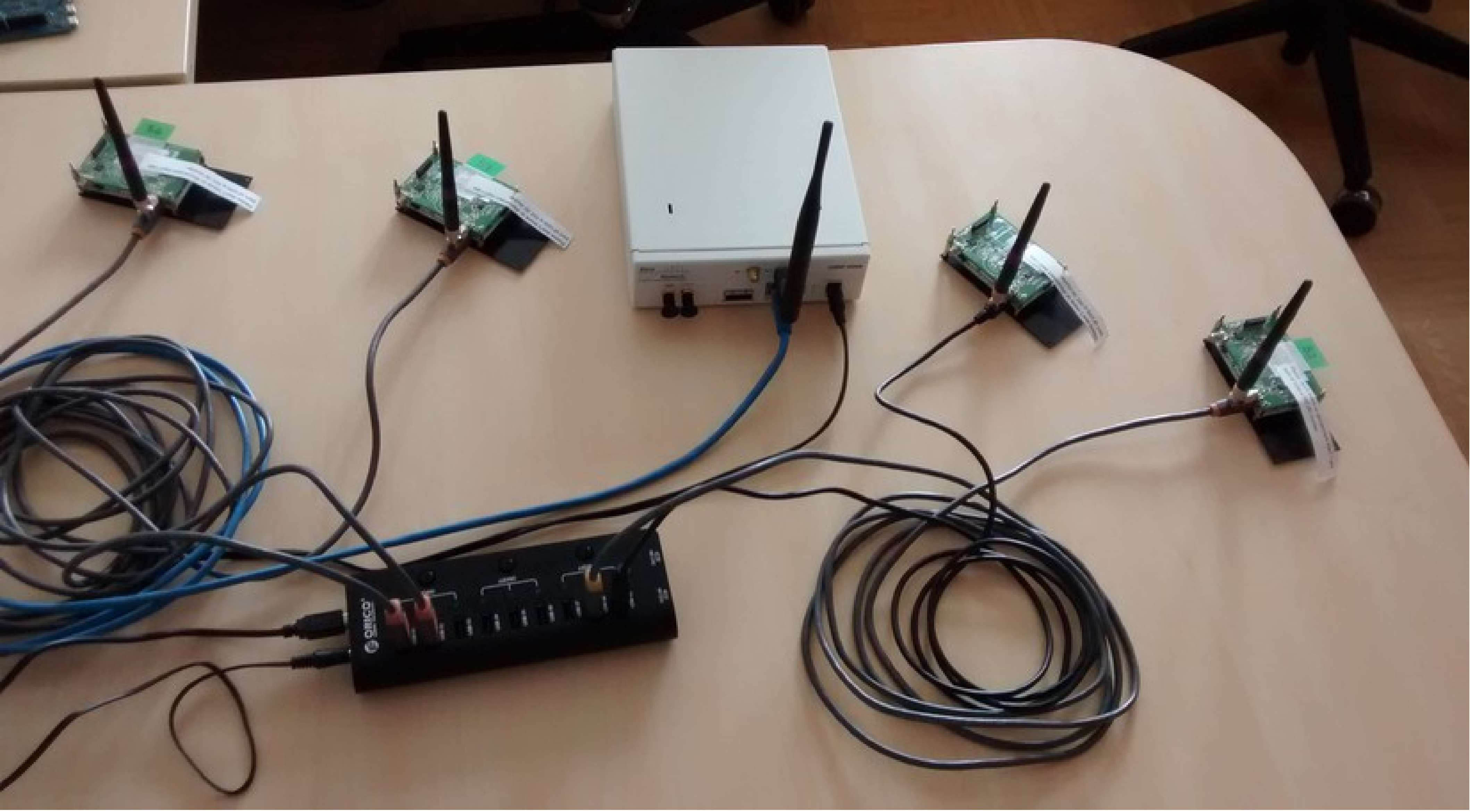
Spectrum sensing

- Energy detection with zero-blind time (i.e. real-time spectrum analysis)
  - USRP continuously senses the spectrum, even when no player is requesting the data.
  - End-to-end FFT with one bin per channel
  - a 200 ms moving-average filter
  - Very CPU intensive max. 64 100 kHz channels



»Implementation of real-time spectrum analysis«, Rohde&Schwarz

Demo

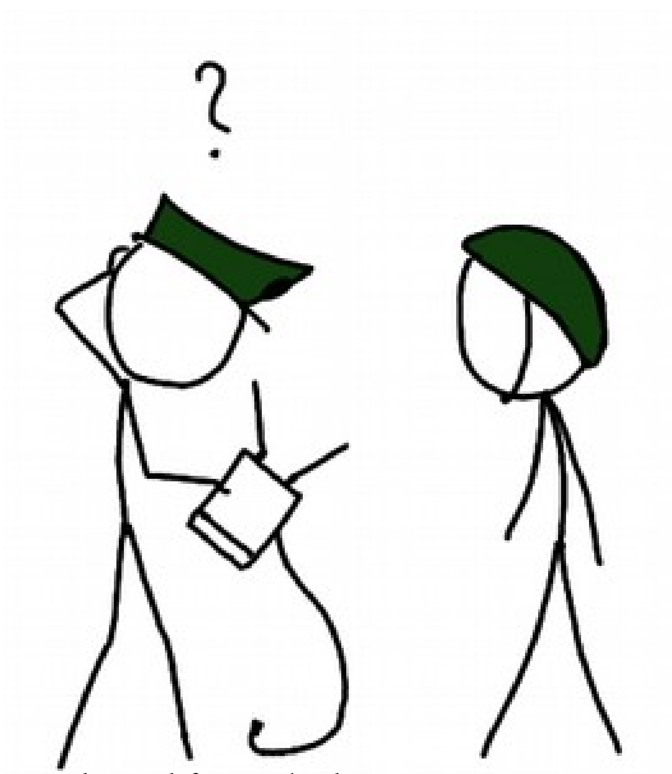

ORICO

Current status

- Game controller works and is well tested
  - only scoring functionality is missing
- Sandbox implemented as a separate Linux process
  - prevents simple ways of cheating
  - not robust against a deliberate attack
  - should probably use VM/SE Linux in the public version
- Only VESNA testbed currently supported
  - CREW partners working on supporting their testbeds
- No work done yet on the web front-end

Questions?

https://github.com/avian2/spectrumwars



adapted from xkcd.com, CC BY-NC 2.5

Tomaž Šolc
tomaz.solc@ijs.si