RFQuack

The versatile RF-analysis tool that quacks!

With ❤️ From Trend Micro Research

Presented at HITB 2019 Armory by:

Federico Maggi, @phretor

With ❤️ From Trend Micro Research

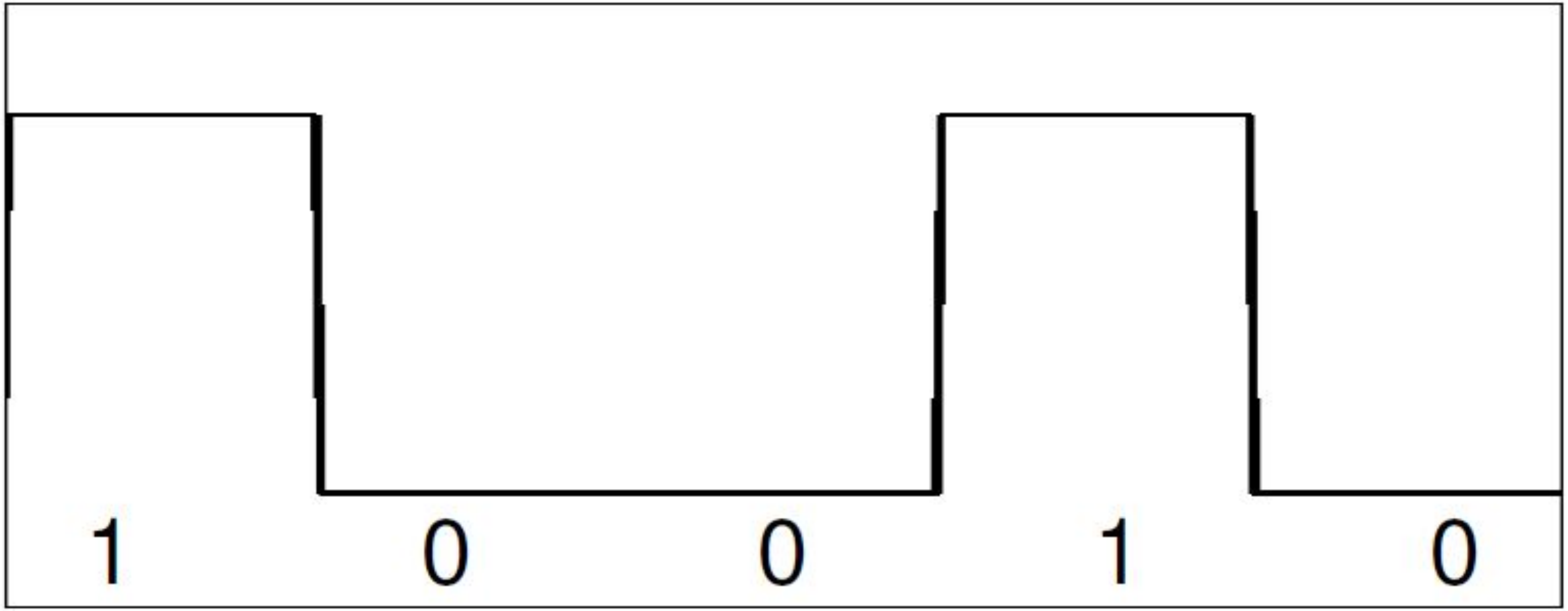
This work wouldn't have been possible without the support of my employer.

In particular, I'd like to thank:

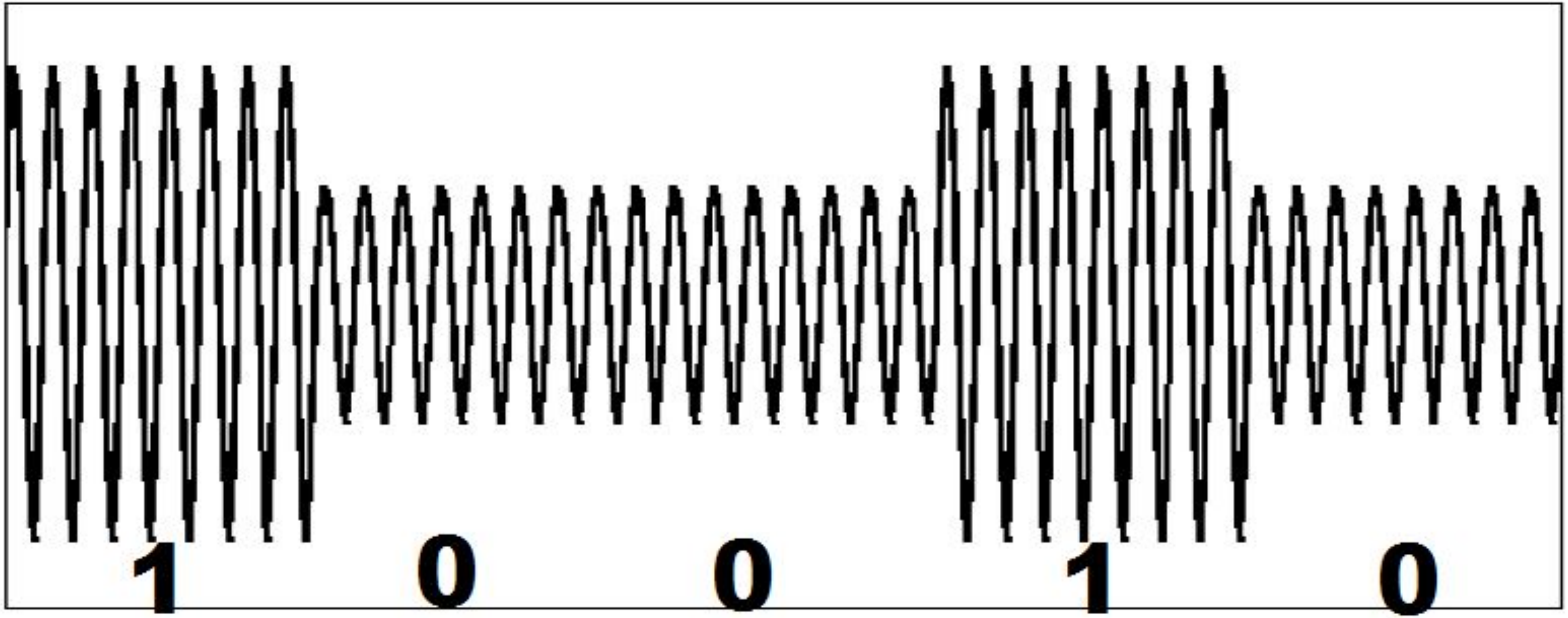
- **Managers** and **execs**, who believed in this project and let me work on it
- **Jonathan Andersson**, who inspired and helped me debugging the quirks of the CC1120
- **Philippe Lin**, an early adopter of (the first-ever prototype of) RFQuack
- **Marco Balduzzi**, who never stopped asking me *"how's RFQuack going?"*
- **Jullienne Yerro**, and the rest of the marketing team for the beautiful logo (kudos to **Jojo Mendoza** for that) and the media support

Signal Analysis 101

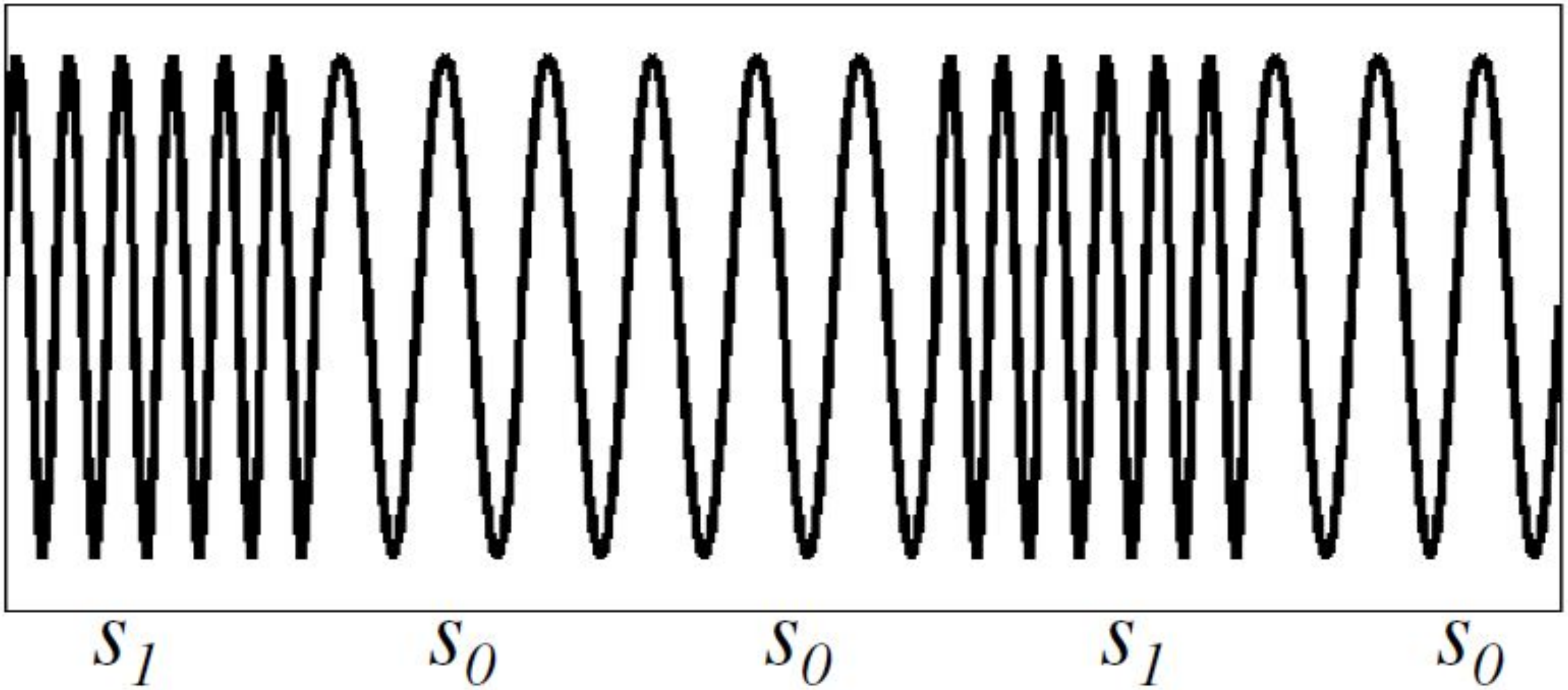
From Symbols to Signal: Baseband Data



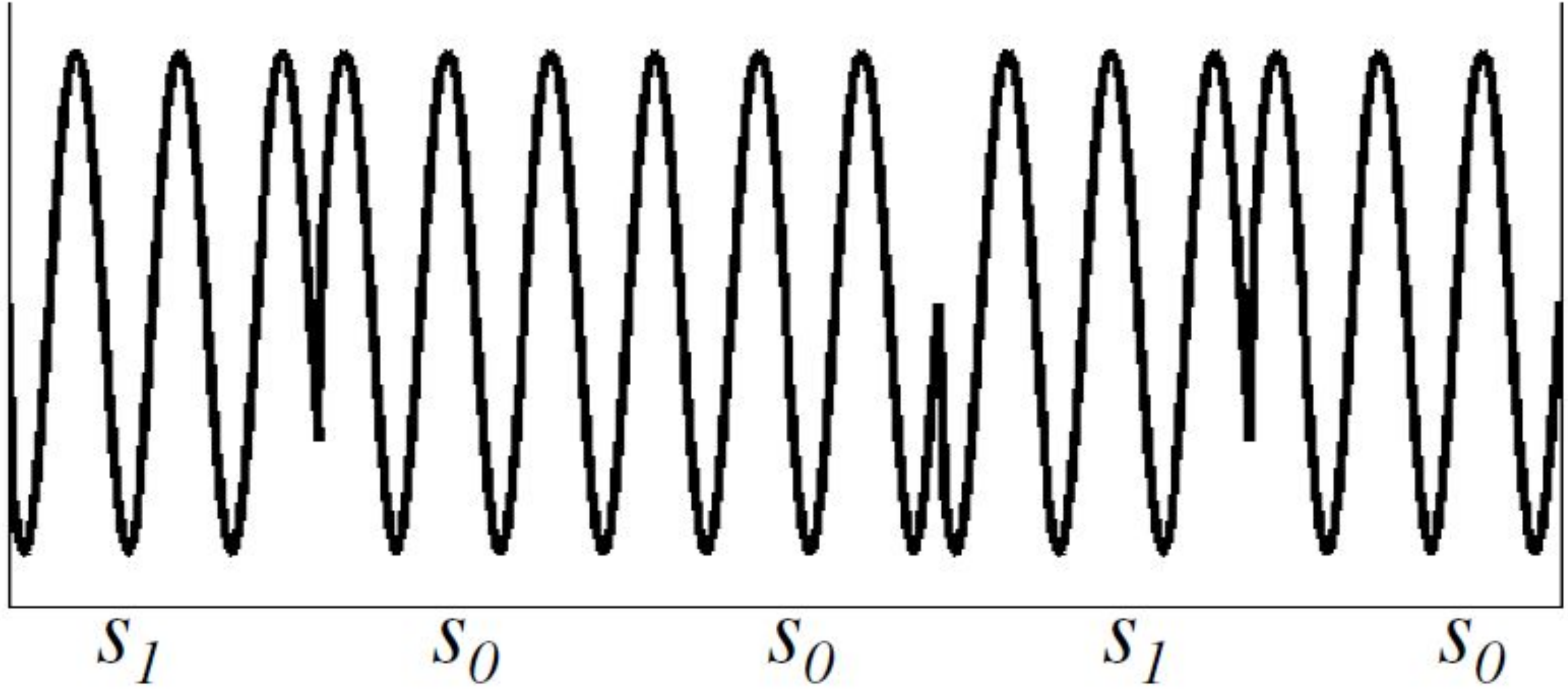
From Symbols to Signal: Amplitude Shift



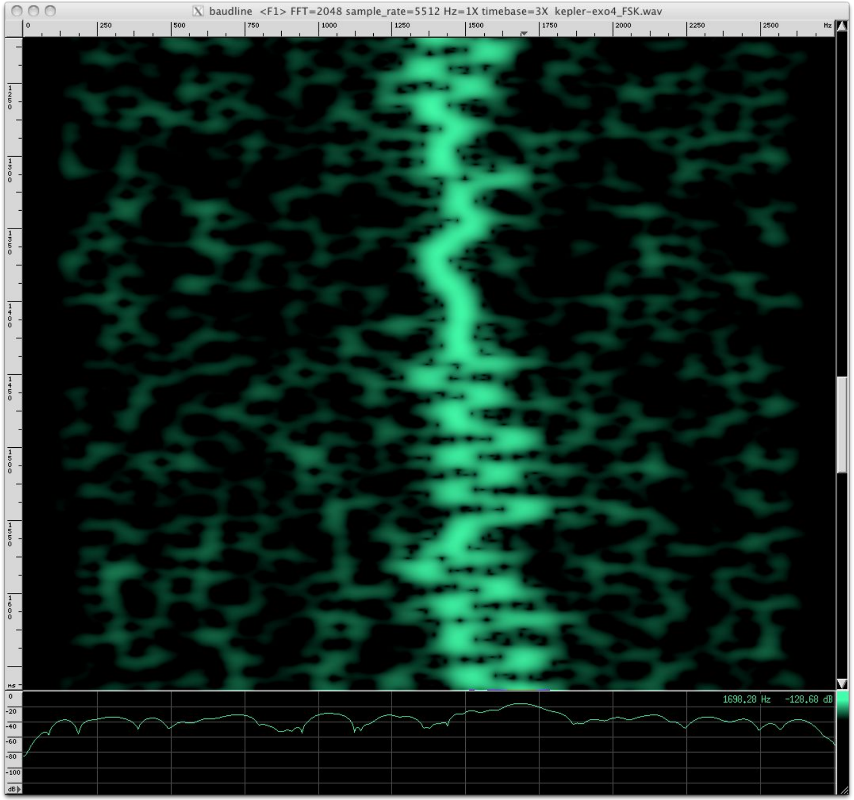
From Symbols to Signal: Frequency Shift

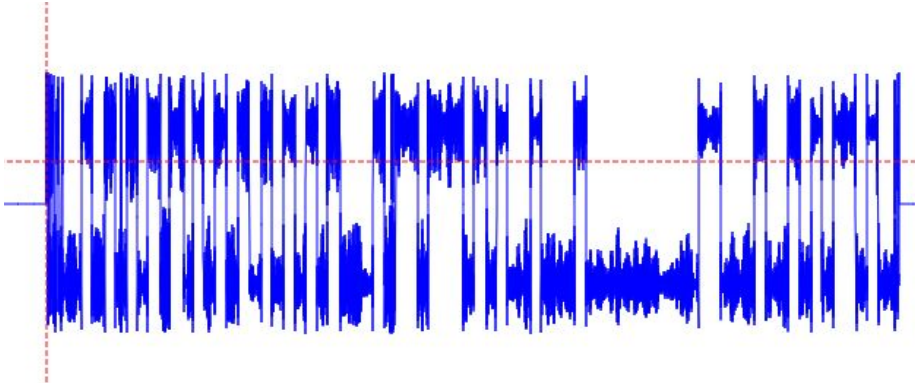


From Symbols to Signal: Phase Shift



From Signal to Symbols





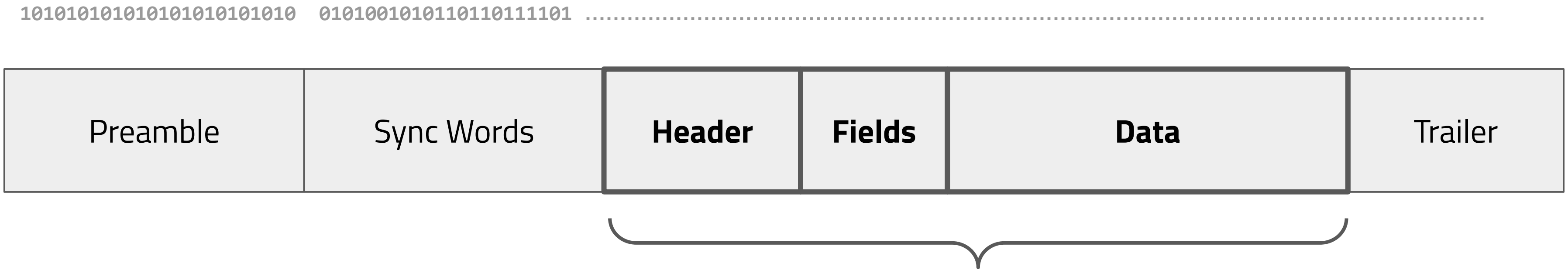
01010110110101010010101101101111010101111111010101

The Hard Part is not Over

From Bits to Packets

Still, we Haven't Reverse Engineered the Protocol

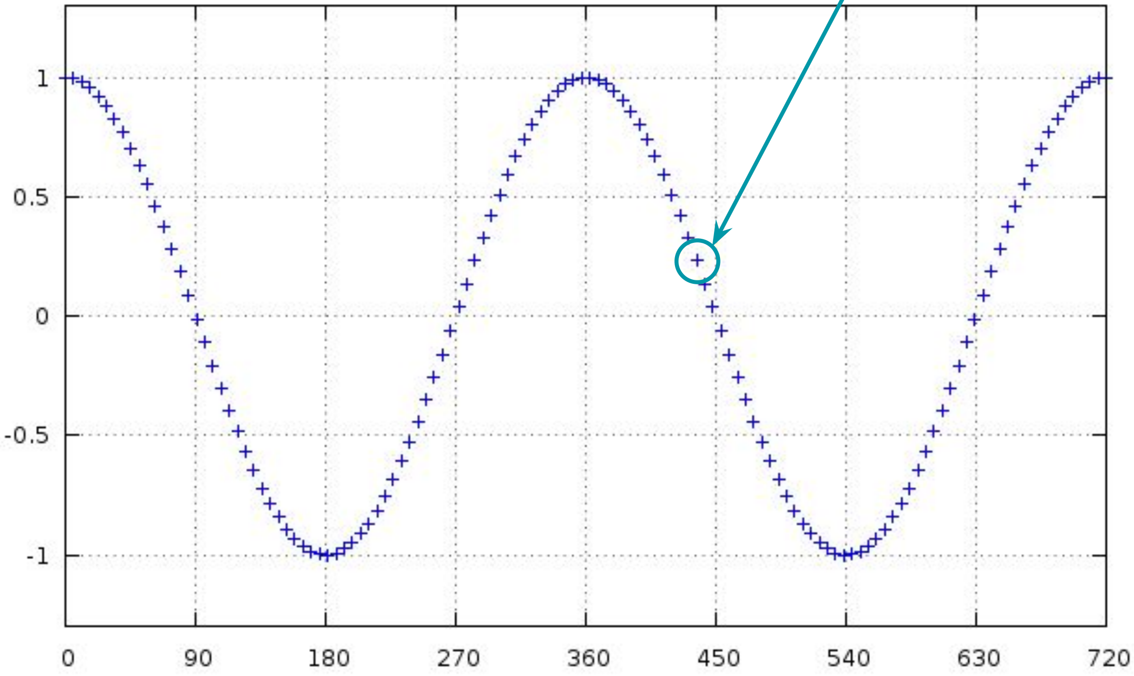
From Packets to Application Payload



Custom application protocol
(with security through obscurity baked in, usually)

Software Defined Radios

SDRs - Main Idea: Take Many RF Signal Samples



SDRs: Pros vs. Cons

- Great for signal **reconnaissance**
- Very flexible: you get straight access to the raw **signal**
- **Software** support to assist in writing radios

- You have to **write your own radio** in software
- Radio **accuracy** is up to you
- Serious ones can be **expensive**

Bottom Line

It's hard to build an accurate and reliable radio

RF Dongles

RF Dongles - Main Idea: Embedded Radio


```
root@edolin ~/rfcat $ ./rfcat -r
'RfCat, the greatest thing since Frequency Hopping!'

Research Mode: enjoy the raw power of rflib

currently your environment has an object called "d" for dongle.  this is how
you interact with the rfcat dongle:
    >>> d.ping()
    >>> d.setFreq(433000000)
    >>> d.setMdmModulation(MOD_ASK_OOK)
    >>> d.makePktFLEN(250)
    >>> d.RFxmit("HALLO")
    >>> d.RFrecv()
    >>> print d.reprRadioConfig()


In [1]:
```

RF Dongles: Pros vs. Cons

- Great to **quickly demodulate** signals
- Very **accurate**: you get reliable access to the demodulated bitstream
- As **fast** as the **hardware** radio

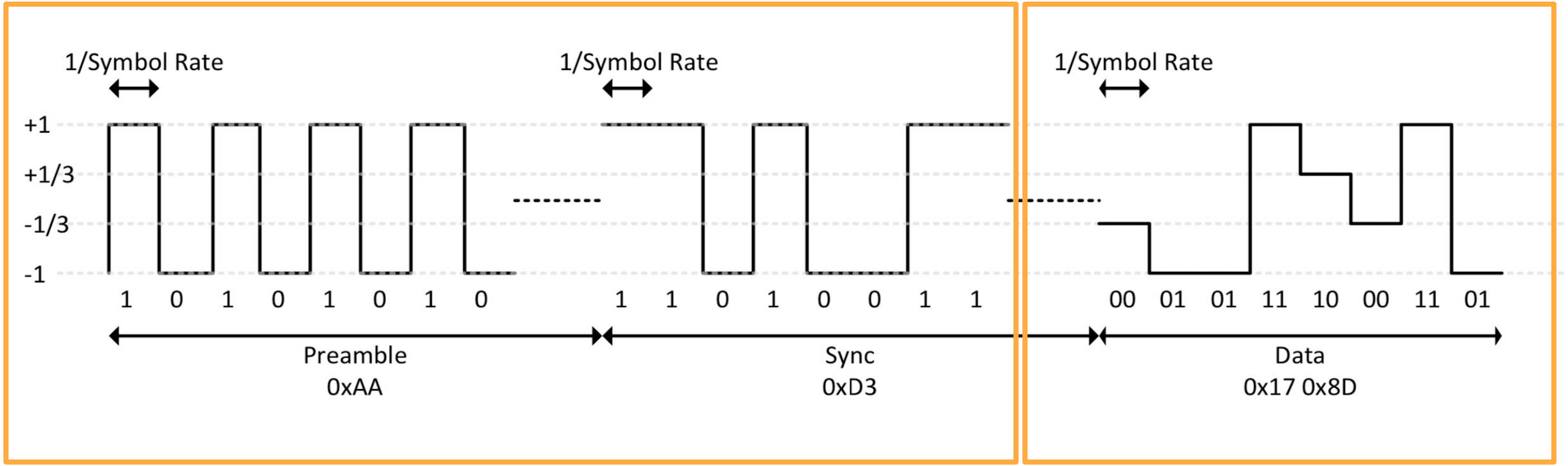
- **Not** as **flexible** as SDRs
- Demodulation support is **limited** to what the **hardware** can do

Bottom Line

There's no such thing like a "generic RF dongle"

The Perfect Corner Case

TI CC11xx's in 4-FSK



It's still 4-FSK, but it uses only 2 symbols for preamble and sync

Then switches to 4 symbols

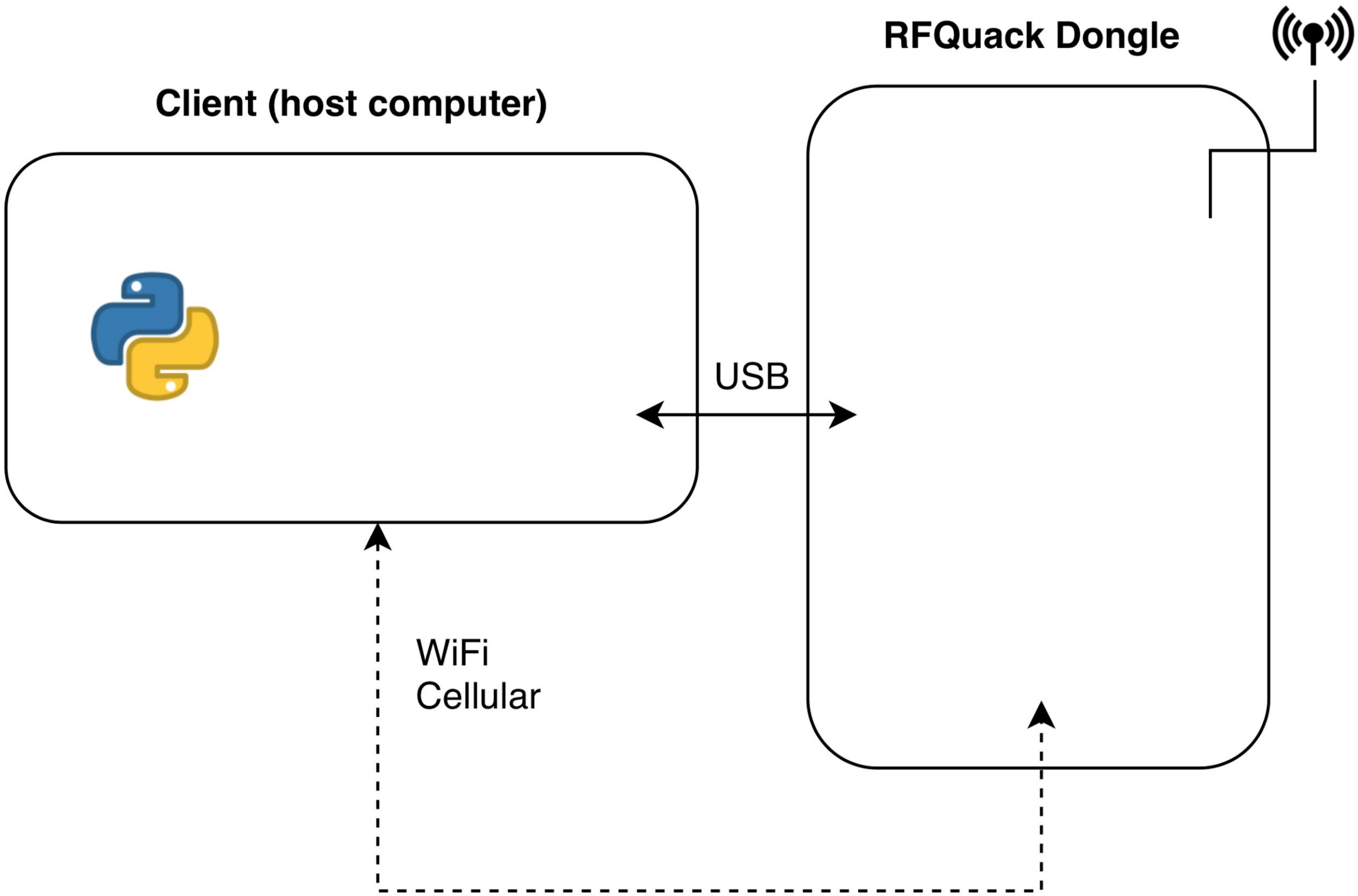
But but...the TI CC1111 can do 4-FSK

6:4	MOD_FORMAT[2:0]	000	R/W	The modulation format of the radio signal	
				000	2-FSK
				001	GFSK
				010	Reserved
				011	ASK/OOK
				100	Reserved 4-FSK
				101	Reserved
				110	Reserved
				111	MSK

followed by the data written to RFD.
The sync. word is a two-byte value set in the SYNC1 and SYNC0 registers. The sync word provides byte synchronization of the incoming packet. A one-byte sync word can be emulated by setting the SYNC1 value to the preamble pattern. It is also possible to emulate a 32 bit sync word by using MDMCFG2.SYNC_MODE set to 3 or 7. The sync word will then be repeated twice.

RFQuack

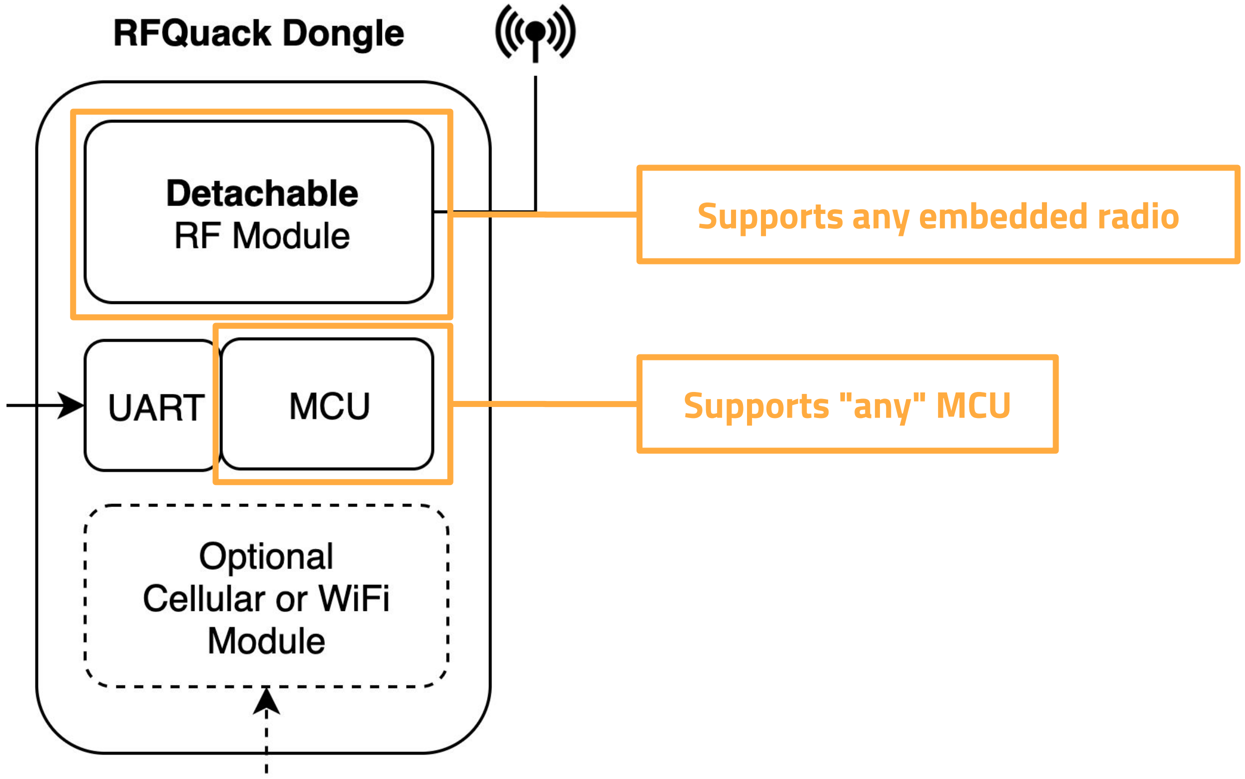
TREND MICRO | research



RFQuack Dongle
Client (host computer)
USB
WiFi
Cellular

Principles

Hardware Modularity



Software Abstraction With Full Low-level Control

- High-level operations
  - Set frequency
  - Switch mode (TX, RX, IDLE)
  - Reset radio

- Low-level operations
  - Set register to value
  - Get register value
  - Upcoming: straight access to make SPI transactions from the Python client

Developer Friendly

- C + Arduino compatible + build system based on PlatformIO
- Simple and clean API: Inspired by, and including MQTT
    - **In**bound `>[command]~Base64([Protobuf-serialized blob])`
    - **Out**bound `<[command]~Base64([Protobuf-serialized blob])`

- Verbose, configurable logging facility

From RFQuack

```
[RFQ]        156 T: RFQuack data structure initialized: WEMOSD1_CC1120
[RFQ]        464 T: Connecting WEMOSD1_CC1120_6c54 to MQTT broker 192.168.42.225:1883
[RFQ]       2117 T: MQTT connected
                    ...
[RFQ]       2130 T: Subscribed to topic: rfquack/in/#
```

From the radio driver

```
[RHAL] SRES
[RHAL] SCAL
[RHAL] SIDLE
[RHAL] START MARCSTATE.MARC_STATE ==============================
[RHAL] Waiting for MARCSTATE.MARC_STATE == 0b1
[RHAL] END MARCSTATE.MARC_STATE ==============================
[RHAL] IRQ bus clear
[RHAL] _variablePayloadLen = 1
```

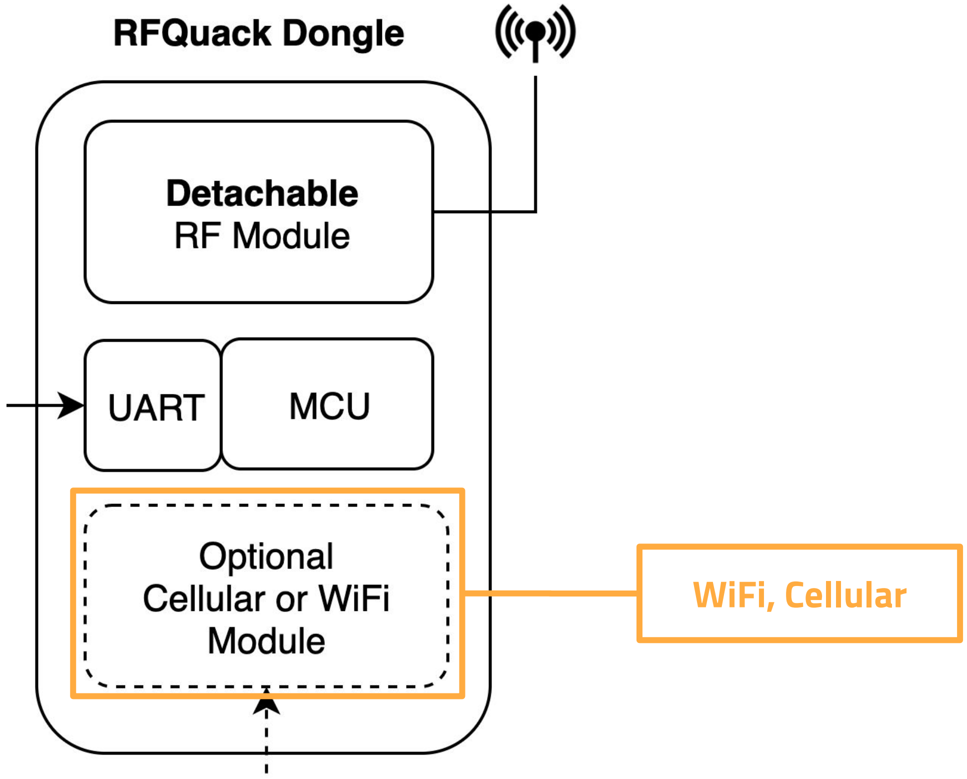
```
▾ config/
    general.h
    logging.h
    network.h
    radio.h
    transport.h
▾ defaults/
    general.h
    logging.h
    network.h
    radio.h
    transport.h
▸ radio/
▸ utils/
  rfquack.h
  rfquack.options
  rfquack.pb.c
  rfquack.pb.h
  rfquack.proto
  rfquack_common.h
  rfquack_config.h
  rfquack_logging.h
  rfquack_network.h
  rfquack_radio.h
  rfquack_transport.h
banner.txt
library.json
library.properties
LICENSE
Makefile
README.md
```

```
#define RFQUACK_TOPIC_REGISTER_DEFAULT "register"

#define RFQUACK_TOPIC_PACKET_MODIFICATION "packet_modification"

#define RFQUACK_TOPIC_PACKET_FILTER_DEFAULT "packet_filter"

#define RFQUACK_TOPIC_RADIO_RESET_DEFAULT "radio_reset"

#define RFQUACK_MAX_TOPIC_LEN_DEFAULT 64

/*****************************************************************************
 * Serial Configuration
 *****************************************************************************/

#define RFQUACK_SERIAL_MAX_PACKET_SIZE_DEFAULT RFQUACK_MAX_PACKET_SIZE_DEFAULT
```

```
NORMAL  +0 ~0 -0  <.h   21:transport.h  22:transport.h   cpp  utf-8[unix]  64%
```

```
#define RFQUACK_TOPIC_MODEM_CONFIG RFQUACK_TOPIC_MODEM_CONFIG_DEFAULT
#endif

#ifndef RFQUACK_TOPIC_PACKET
#define RFQUACK_TOPIC_PACKET RFQUACK_TOPIC_PACKET_DEFAULT
#endif

#ifndef RFQUACK_TOPIC_MODE
#define RFQUACK_TOPIC_MODE RFQUACK_TOPIC_MODE_DEFAULT
#endif

#ifndef RFQUACK_TOPIC_REGISTER
#define RFQUACK_TOPIC_REGISTER RFQUACK_TOPIC_REGISTER_DEFAULT
#endif
```

Cut the Cords

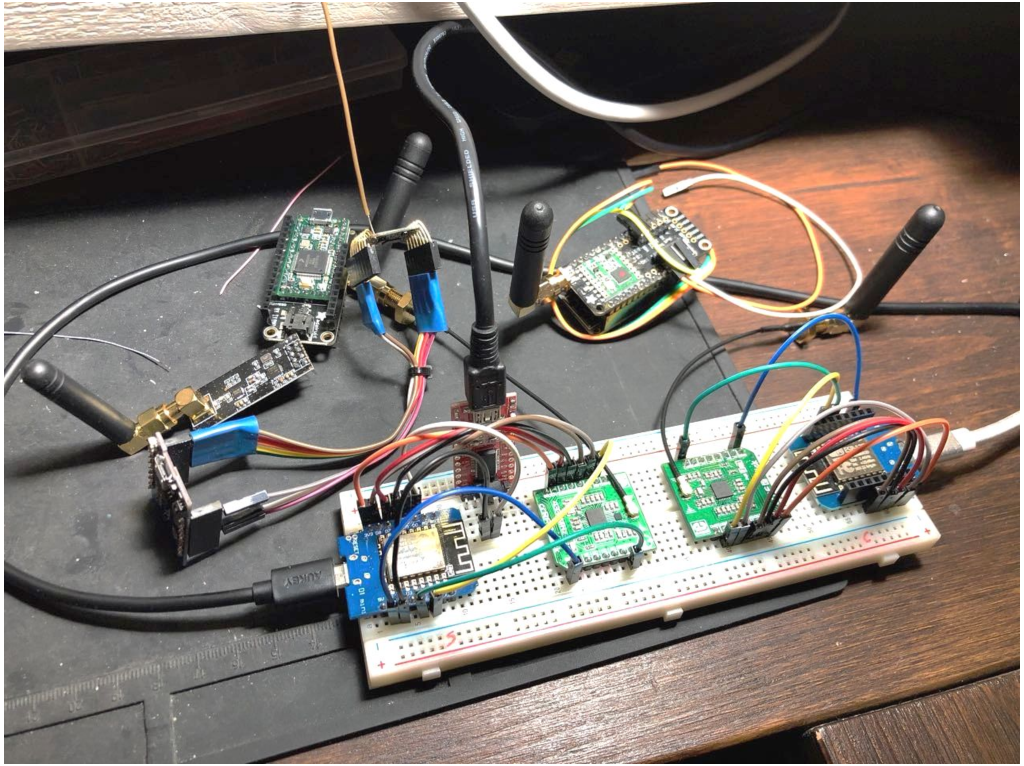


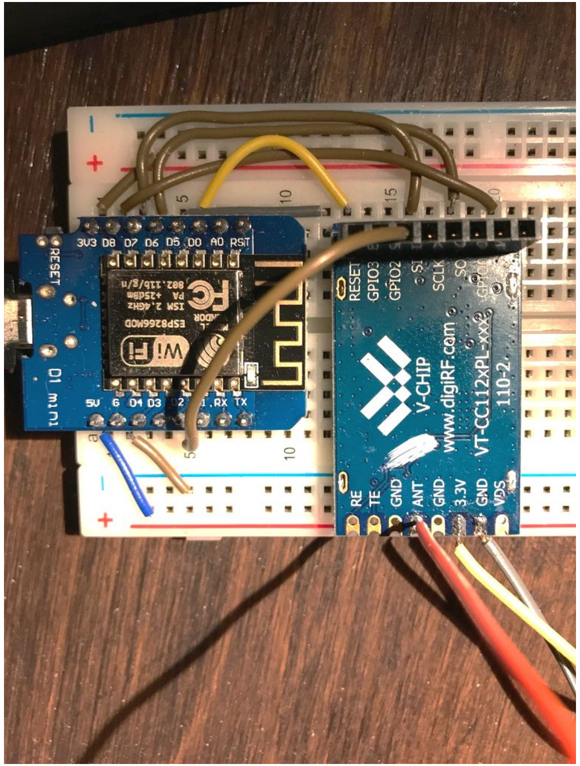
Comparison Matrix

	SDRs	YardStickOne	PandwaRF	RFQuack
Supported Radios	Any (software)	CC1101	CC1101	Any (even multi radio)
Client Support	Lots of options	RFCat firmware and client	RFCat client	Developer-friendly API
Open Software	Depends	Yes	Not the firmware	Yes, Arduino compatible
Open Hardware	Depends	Yes	Yes	Yes
Connectivity	USB, Gigabit	USB	USB, BT	USB, WiFi, Cellular
Price	\$20–2000	>= \$100	>= \$110	>= \$40

Getting Started

Get and Assemble the Hardware



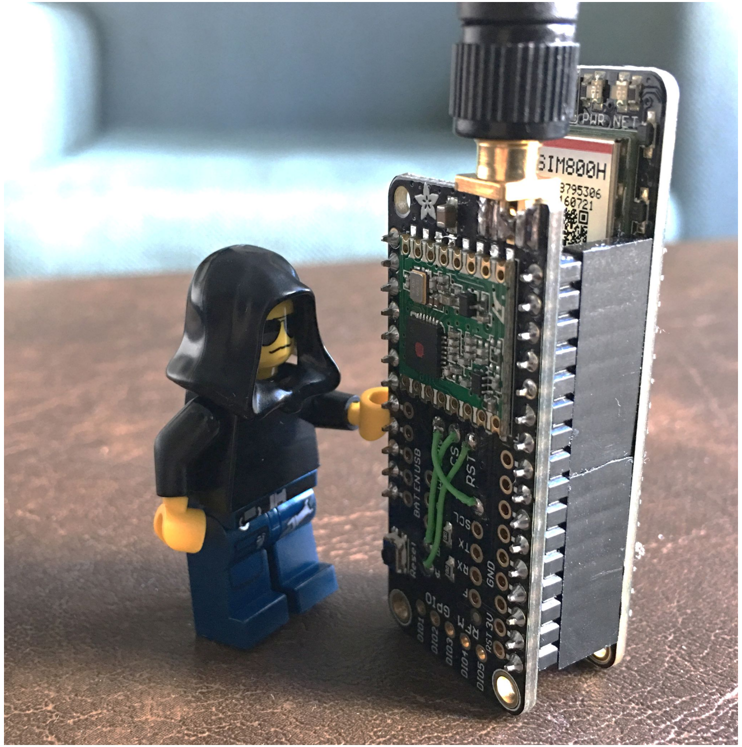


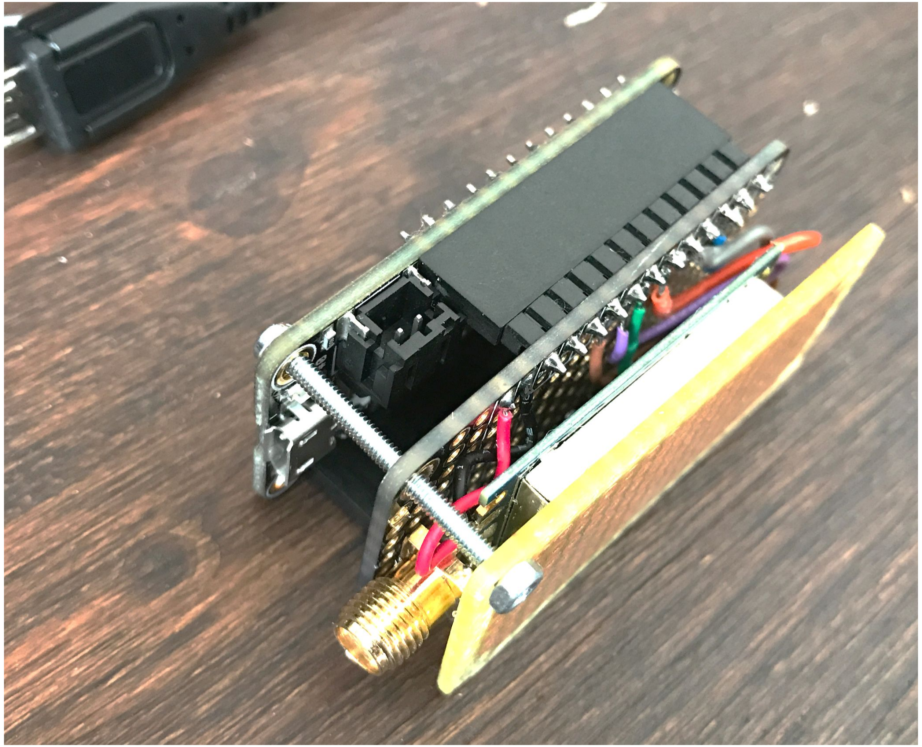
TREND MICRO research

What the Hardware!?

- Pick any **SPI** (Serial Peripheral Interface) embedded **radio** module
  - Available anywhere from Adafruit, Sparkful, eBay, Amazon, AliExpress
  - RFM69, CC1111, CC1120, nRF24, nRF51
- Hint: there are **pre-made shields** for popular radios (e.g., FeatherWing Radio)
- Connect **SPI pins**
  - MOSI
  - MISO
  - SCLK
  - CS
- Plus at least **one interrupt line** to the MCU's **GPIO** pin
- Add an **antenna**

Make it Nicer (and give it a modem)





Check out the Code

```
$ git clone https://github.com/trendmicro/RFQuack

$ cd RFQuack

$ pip install -r src/client/requirements.pip

$ pio install -g <library name>  # from library.json

$ cd examples/
```

Branch: gsm-rfm69hcw

RFQuack / examples /

Federico Maggi Pre HITB release

..

RFQuack-huzzah-rf69hw-serial

RFQuack-huzzah-rf69hw

RFQuack-wemosd1-cc1120-serial

RFQuack-wemosd1-cc1120

RFQuack-wemosd1-rf69hcw-serial

RFQuack-wemosd1-rf69hcw

`RFQuack-<board>-<radio>-<transport>`

`<no transport> = MQTT, by default`

Configure the Firmware "src/main.cpp"

```
#define RFQUACK_UNIQ_ID "WEMOSD1_CC1120"          // <- unique ID
#define RFQUACK_NETWORK_ESP8266
#include "wifi_credentials.h"                     // <- not committed because it contains secrets

#define RFQUACK_TRANSPORT_MQTT
#define RFQUACK_MQTT_BROKER_HOST "192.168.42.225"  // <- MQTT broker IP or hostname (credentials are supported too)

#define RFQUACK_RADIO_CC1120                       // <- Radio chip (CC1120 and RF69 are supported as of now)
#define RFQUACK_RADIO_PIN_CS 15                    // <- SPI Slave select PIN
#define RFQUACK_RADIO_PIN_IRQ 4                    // <- Interrupt PIN
#define RFQUACK_RADIO_PIN_RST 5                    // <- Reset PIN
#define RFQUACK_DEBUG_RADIO true
#define RFQUACK_DEV
#define RFQUACK_LOG_SS_DISABLED                    // <- Disable SoftwareSerial logging (we're using HardwareSerial)

#include "rfquack.h"
void setup() { rfquack_setup(); } void loop() { rfquack_loop(); }
```

TREND MICRO research

Mind the Serial Port in "platformio.ini"

```
[env:d1_mini]
platform = espressif8266
board = d1_mini
framework = arduino
upload_port = /dev/cu.wchusbserial14110
monitor_port = /dev/cu.wchusbserial14110
upload_speed = 115200
monitor_speed = 115200
```

Build the Firmware

```
$ git clone https://github.com/trendmicro/RFQuack
$ cd RFQuack
$ pip install -r src/client/requirements.pip
$ pio install -g <library name>  # from library.json
$ cd examples/
$ make && sleep 1 && make upload && make monitor
$ mosquitto -v # if using MQTT transport
```

Boot and Connect

```
[RFQ]        152 T: Setting sync words length to 4
[RFQ]        153 T: Packet filtering data initialized
[RFQ]        154 T: Packet modification data initialized
[RFQ]        156 T: RFQuack data structure initialized: WEMOSD1_CC1120
[RFQ]        464 T: Connecting WEMOSD1_CC1120_6c54 to MQTT broker 192.168.42.225:1883
[RFQ]       2117 T: MQTT connected
[RFQ]       2130 T: Subscribed to topic: rfquack/in/#
[RFQ]       2231 T: 📡 Setting up radio (CS: 15, RST: 5, IRQ: 4)
[RFQ]       3141 T: 📶 Radio initialized (debugging: true)
[RFQ]       3142 T: CC1120 type 0x4823 ready to party 🎉
[RFQ]       3144 T: Modem config set to 5
[RFQ]       3147 T: Max payload length:  128 bytes
[RFQ]       3151 T: 📶 Radio is fully set up (RFQuack mode: 4, radio mode: 2)
[RFQ]       3258 T: Transport is sending 26 bytes on topic rfquack/out/status
```

```
[RFQ]        155 T: RFQuack data structure initialized: WEMOSD1_CC1120
[RFQ]        464 T: Connecting WEMOSD1_CC1120_28e7 to MQTT broker 192.168.42.225:1883
[RFQ]       1549 T: MQTT connected
[RFQ]       1556 T: Subscribed to topic: rfquack/in/#
[RFQ]       1656 T: 📡 Setting up radio (CS: 15, RST: 5, IRQ: 4)
[RHAL] SRES
[RHAL] SCAL
[RHAL] SIDLE
[RHAL] START MARCSTATE.MARC_STATE ==============================
[RHAL] Waiting for MARCSTATE.MARC_STATE == 0b1
[RHAL] END MARCSTATE.MARC_STATE ==============================
[RHAL] IRQ bus clear
[RHAL] _variablePayloadLen = 1
[RFQ]       2566 T: 📶 Radio initialized (debugging: true)
[RFQ]       2567 T: CC1120 type 0x4823 ready to party 🎉
[RFQ]       2569 T: Modem config set to 5
[RFQ]       2572 T: Max payload length:  128 bytes
[RFQ]       2576 T: 📶 Radio is fully set up (RFQuack mode: 4, radio mode: 2)
[RFQ]       2683 T: Transport is sending 26 bytes on topic rfquack/out/status
```

RFQuack serial console output (optional, but very useful)

```
1557090916: Sending PINGRESP to WEMOSD1_CC1120_28e7
1557090920: Received PINGREQ from WEMOSD1_CC1120_28e7
1557090920: Sending PINGRESP to WEMOSD1_CC1120_28e7
1557090924: Received PINGREQ from WEMOSD1_CC1120_28e7
1557090924: Sending PINGRESP to WEMOSD1_CC1120_28e7
```

MQTT broker output (optional)

```
RFQuack(RFQuackShell, localhost:1883)>
```

RFQuack interactive shell (connected to an RFQuack dongle via MQTT)

```
RFQuack(RFQuackShell, localhost:1883)> q.get_status()
```

Get status of the RFQuack dongle

```
RFQuack(RFQuackShell, localhost:1883)>
stats {
  rx_packets: 0
  tx_packets: 0
  rx_failures: 0
  tx_failures: 0
  tx_queue: 0
  rx_queue: 0
}
mode: IDLE
modemConfig {
  syncWords: "EDCB"
}
tx_repeat_default: 0
```

Packet statistics

Modem status

```
RFQuack(RFQuackShell, localhost:1883)>

RFQuack(RFQuackShell, localhost:1883)> q.set_modem_config(syncWords='\x93\x0B\x51\xDE', txPower=5)

RFQuack(RFQuackShell, localhost:1883)> q.data
Out[3]:
{u'status': [stats {
    rx_packets: 0
    tx_packets: 0
    rx_failures: 0
    tx_failures: 0
    tx_queue: 0
    rx_queue: 0
  }
  mode: IDLE
  modemConfig {
    syncWords: "EDCB"
  }
  tx_repeat_default: 0]}

RFQuack(RFQuackShell, localhost:1883)>
```

Data exchanged between dongle and client is retained in q.data

DEMO

Talking Nodes

Main Functionalities

Modem Configuration: q.set_modem_config()

```
> q.set_modem_config(
    modemConfigChoiceIndex=0,       # canned RadioHead/RadioHAL modem config
    txPower=14,                     # TX output power (sometimes in dB)
    isHighPowerModule=true,         # required by some radio modules
    syncWords=b'\x43\x42',          # sync words
    preambleLength=4,               # number of bytes of preamble
    carrierFreq=433)                # and of course, carrier frequency
```

Canned Modem Configuration

- Each RadioHead/RadioHAL driver has canned modem configurations
- It's an `enum` type, so `modemConfigChoiceIndex` is the index
- Examples:
  - FSK_Rb2Fd5
    - FSK modulation
    - With data whitening
    - Receiver bandwidth: 2kb
    - Frequency deviation: 5kHz
  - GFSK_Rb9_6Fd19_2
  - OOK_Rb1_2Bw75
- More at: https://www.airspayce.com/mikem/arduino/RadioHead
- For RadioHAL: https://github.com/trendmicro/radiohal

Transmit, Receive

```
> q.set_packet('\x0d\xa2', 13)        # TX'0x0d 0xa2' 13 times
```

- Accepts any raw binary data
- Data size limited by the radio driver (i.e., size of the TX FIFO)
- Re-transmission times limited by RFQuack's TX queue length

```
> q.rx()                              # put radio in RX mode
```

- Will save packets into **`q.data['packet']`**
- Receive rate limited by RFQuack's RX queue length
- Maybe obvious: will match data according to modem config.

DEMO
Sniffing a Weird Protocol

Register Access (a.k.a. program the radio chip)

```
q.set_register(
    0x2e,              # register address (8 or 16 bits)
    0b01000000)        # register value (you can write in HEX or DEC too)

time.sleep(0.2)        # especially if you set many registers in a row
```

- You could **bypass** any (modem) configuration
- You should **study the datasheet** of the radio chip
- You could easily "hang" the radio and RFQuack (just push reset)

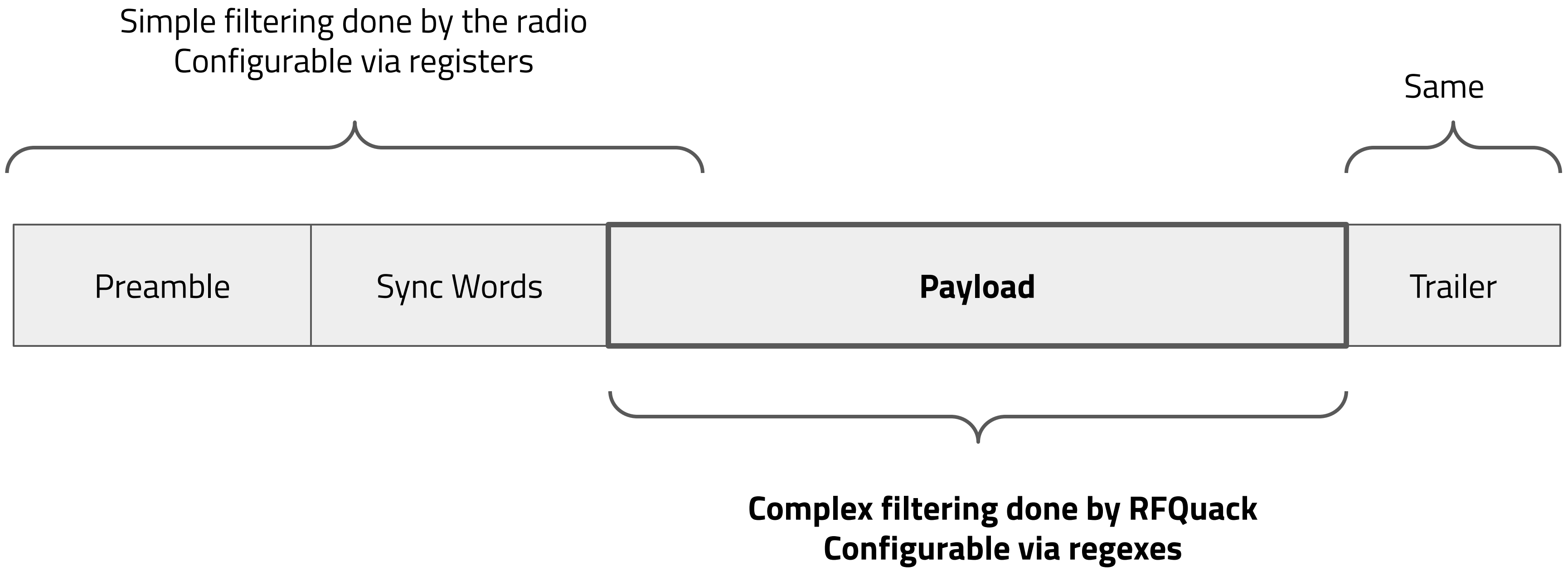
Scripting Up!

```
q.set_modem_config(txPower=14, syncWords=b'\x43\x42', carrierFreq=433)
my_reg_vals = [
    (0x2e, 0x33),
    (0x2f, 0x32),
    (0x01, 0x8D),]

for a,v in my_reg_vals:
    q.set_register(a,v)
    time.sleep(0.2)

q.rx()
```

You can create your own "library" of reusable settings.

Packet Filtering and Manipulation

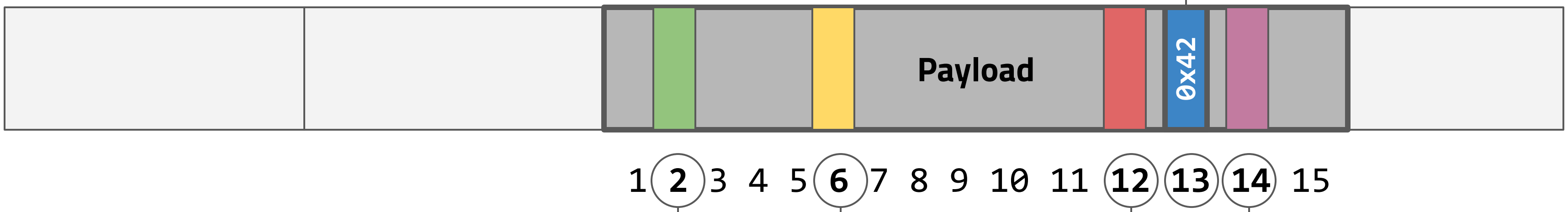
Packet Filtering



Packet Manipulation

`Payload[indexOf(0x42)] = Payload[indexOf(0x42)] OP Value`

Content-based byte-level access



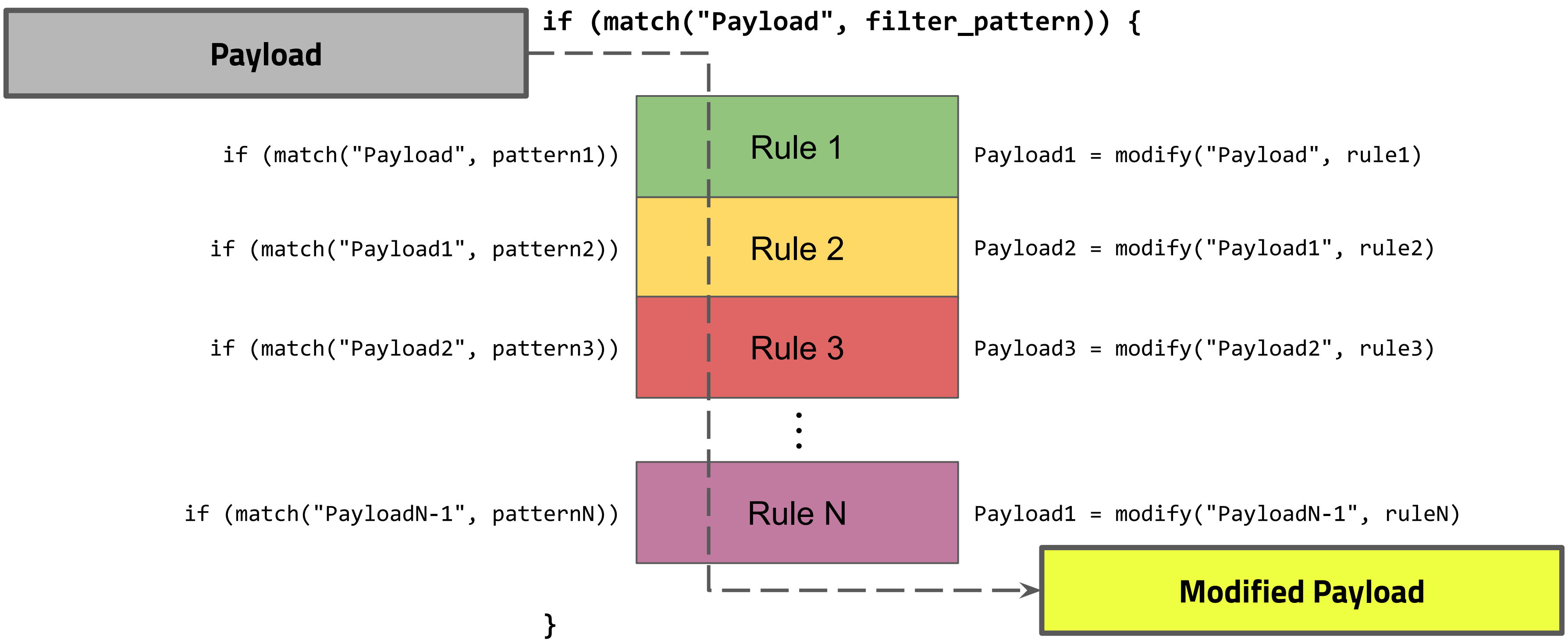
Positional byte-level access

`Payload[position] = Payload[position] OP Value`

Operations:

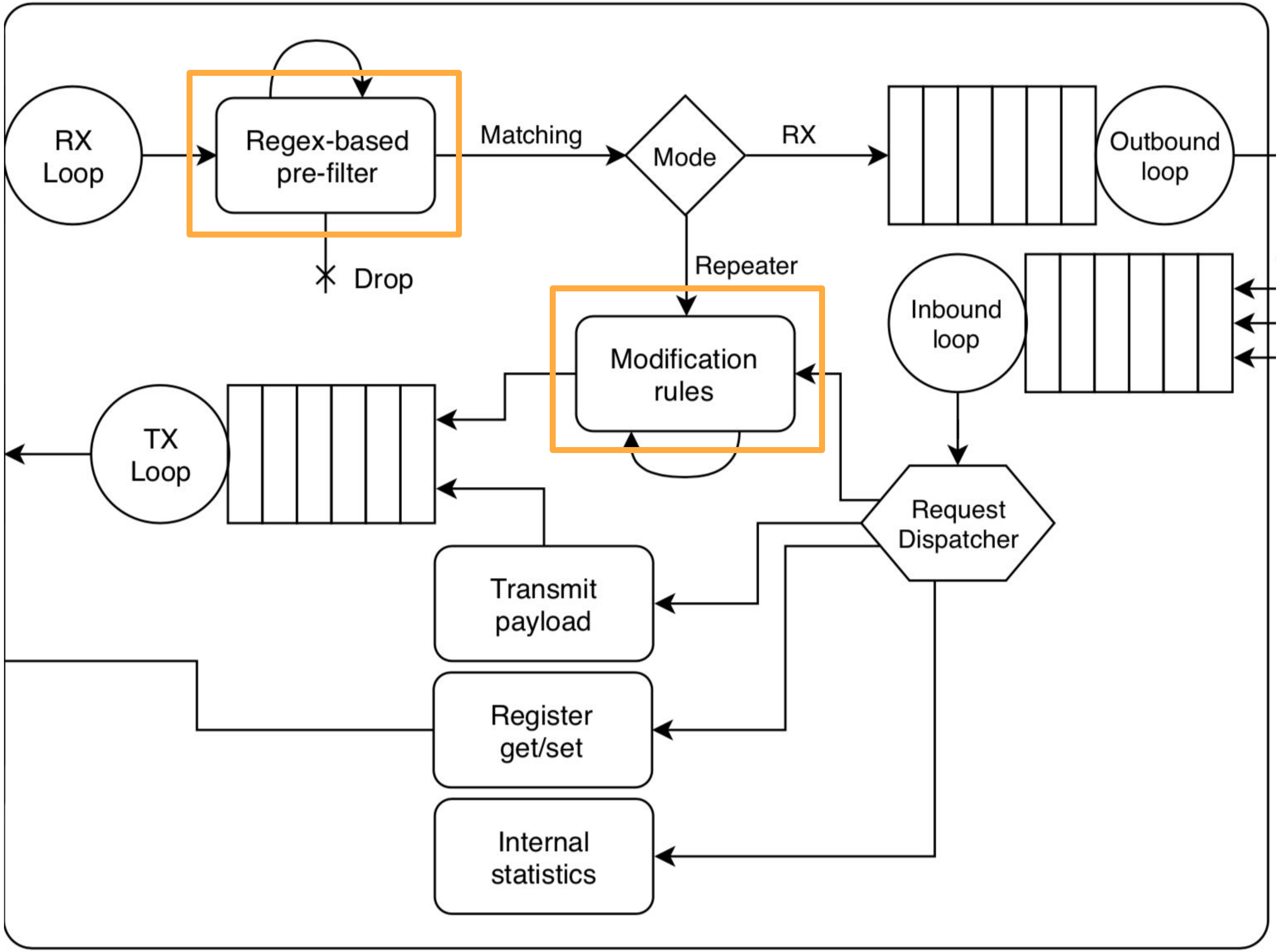
```
XOR
AND
NOT
 >>
 <<
```

Conditional Packet Manipulation



TREND MICRO research

RFQuack Firmware

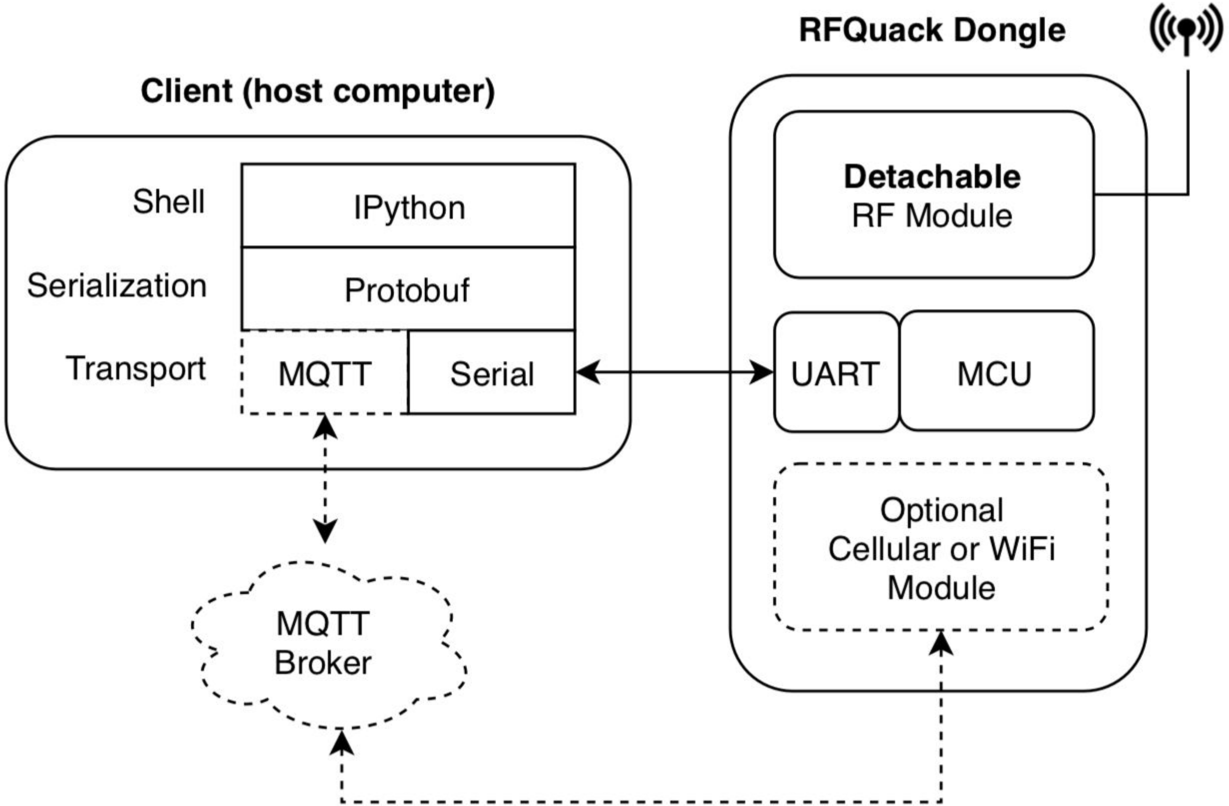


TREND MICRO | research

DEMO
Reverse Engineering a Weird Protocol

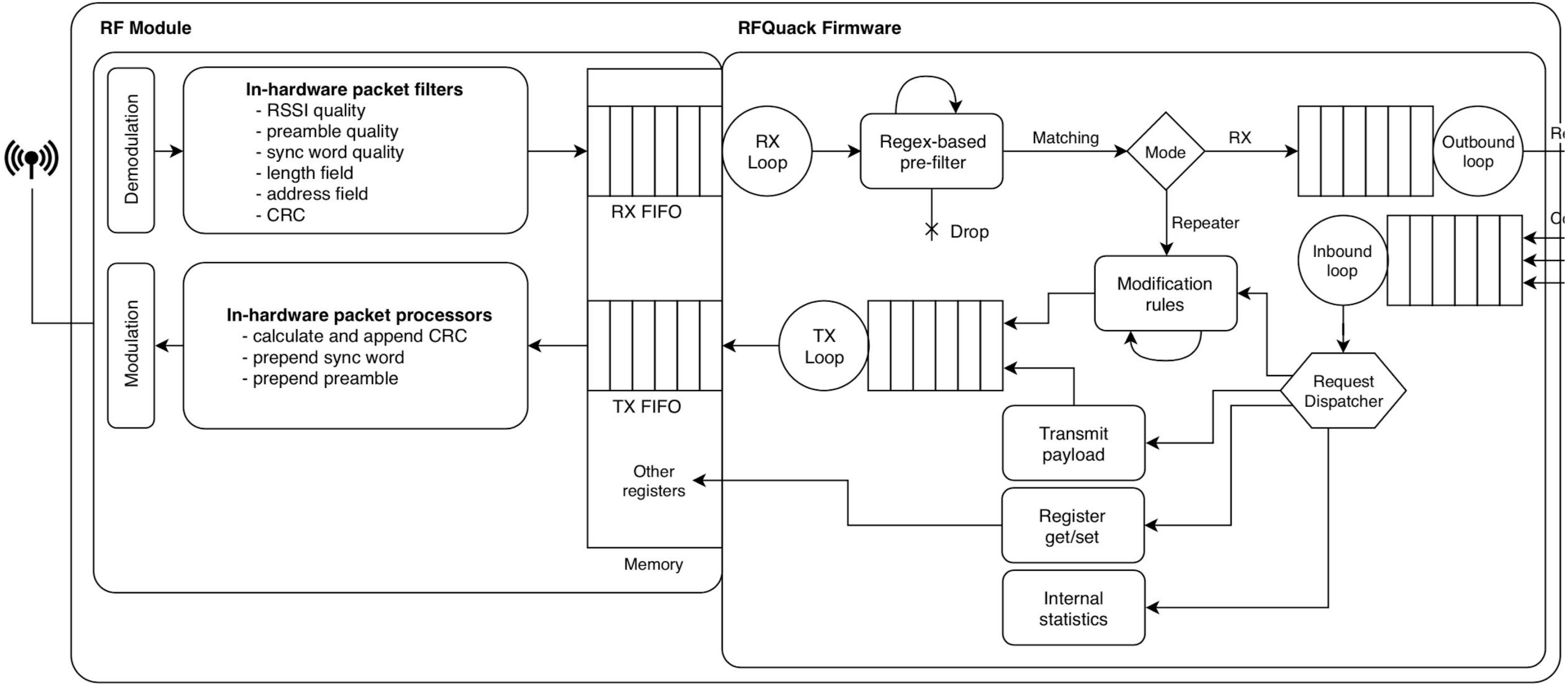
Architecture

High Level



RFQuack Dongle
Client (host computer)
Shell
IPython
Serialization
Protobuf
Transport
MQTT
Serial
Detachable
RF Module
UART
MCU
Optional
Cellular or WiFi
Module
MQTT
Broker

The Radio and Firmware Side

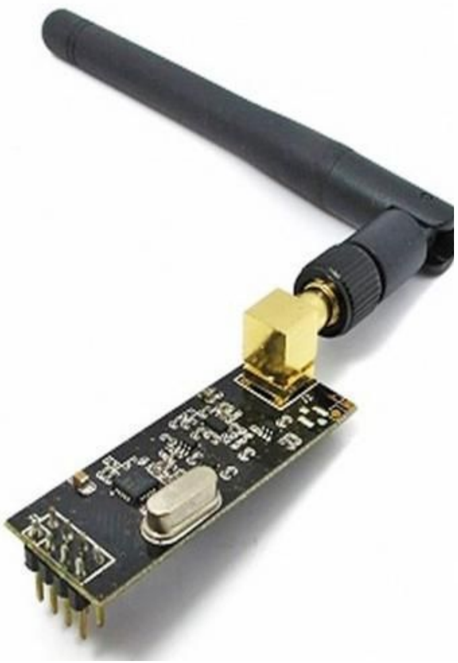


Future

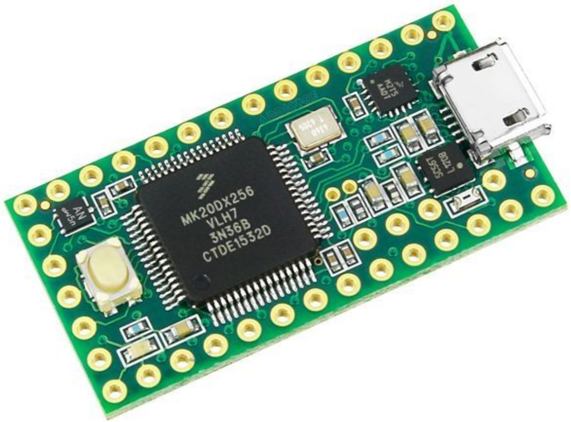
Performance Improvements

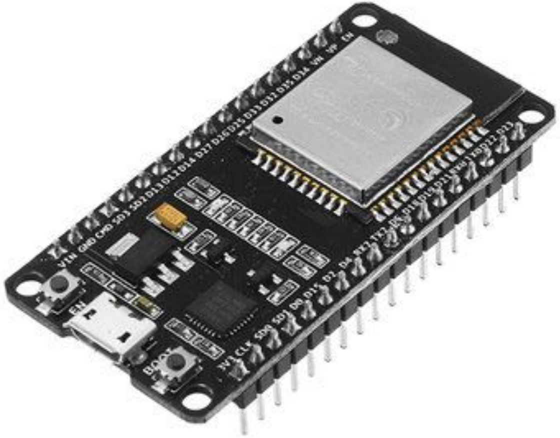
- Interrupt-driven RX function: no polling in the firmware
- Using the radio's TX FIFO buffering when available (reduce SPI traffic for repeated transmissions)
- Make RadioHAL thinner and closer to the radio (less abstract wherever possible)
- Optimize the packet filtering/manipulation engine

Test Other Radios (e.g., 2.4GHz, LoRa)

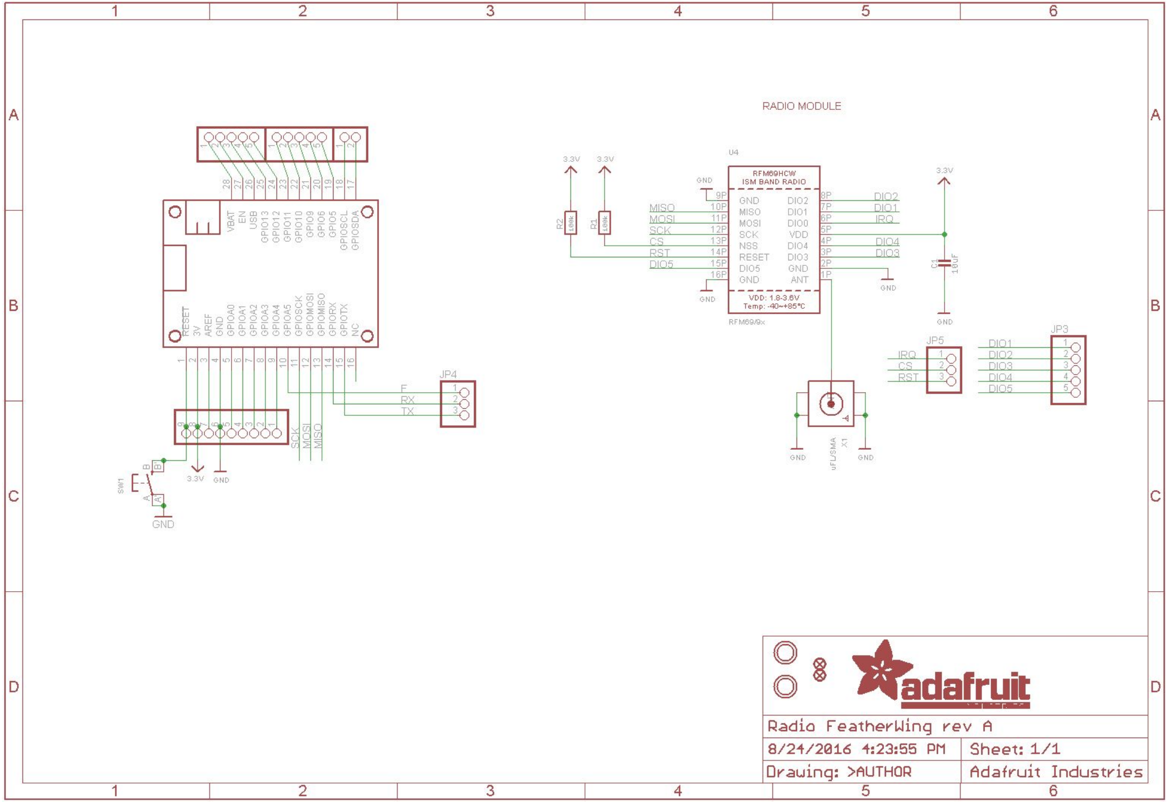


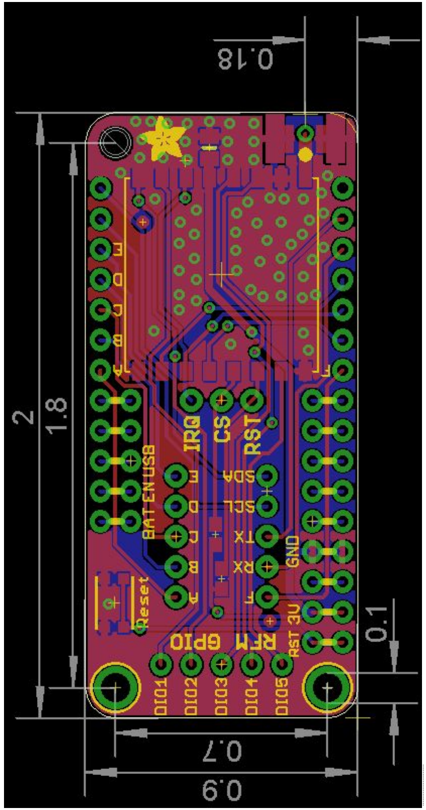
Testing More Platforms





Hardware Shield and Adapters





research

Making a FeatherWing SIM800 (no, not the FONA)

Integrations and Other Enhancements

- GNU Radio and URH
- Web app interface
- Expose a SPI API
- Multiple radio modules (shared SPI bus, 1 IRQ and 1 SS line each)

RFQuack

https://github.com/trendmicro/RFQuack

With ♥ From Trend Micro Research

Presented by:
Federico Maggi, @phretor

TREND MICRO | research