University of Bolzano
Faculty of Computer Science
M.Sc. Software Engineering for Information Systems

unibz

Configurable and resource efficient framework for data and command transmission over LoRaWAN

Dissertation by:
SAVEV DAVID

Supervisor:
JANES ANDREA
Co-Supervisor:
HOFER FLORIAN

Goal

Development of a prototypical framework in the LoRaWAN domain that enables interaction with I/Os and with serial communication protocols.

***Key features*: configurability (functionality)** and **efficiency** in terms of quantity of data transmitted.

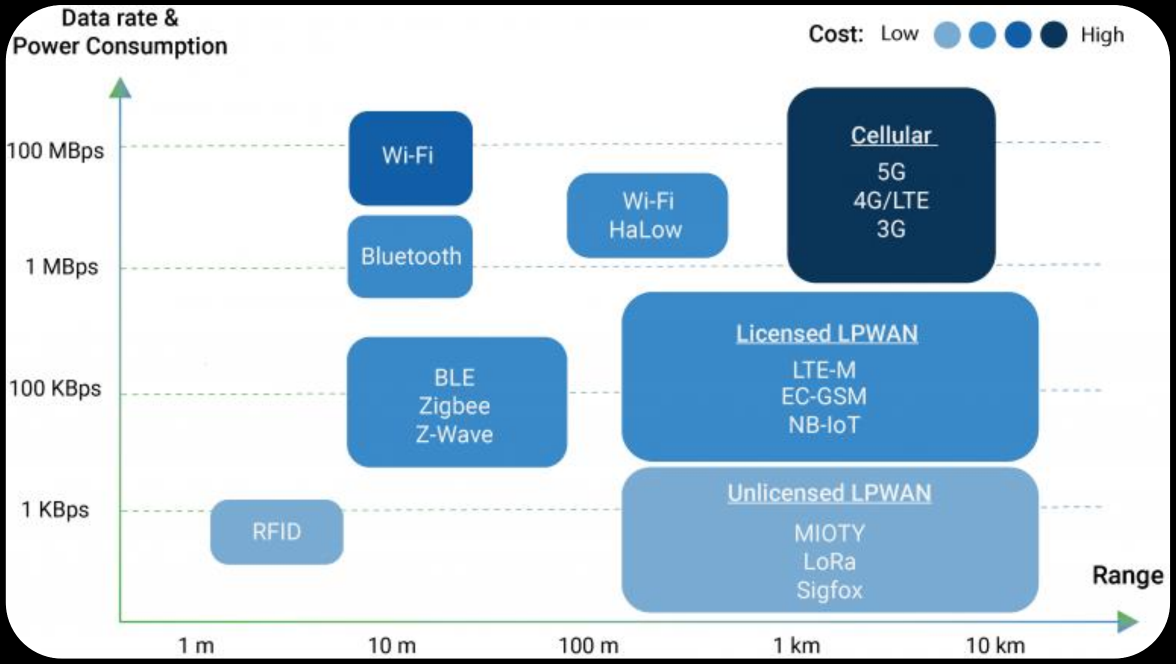
REASON:

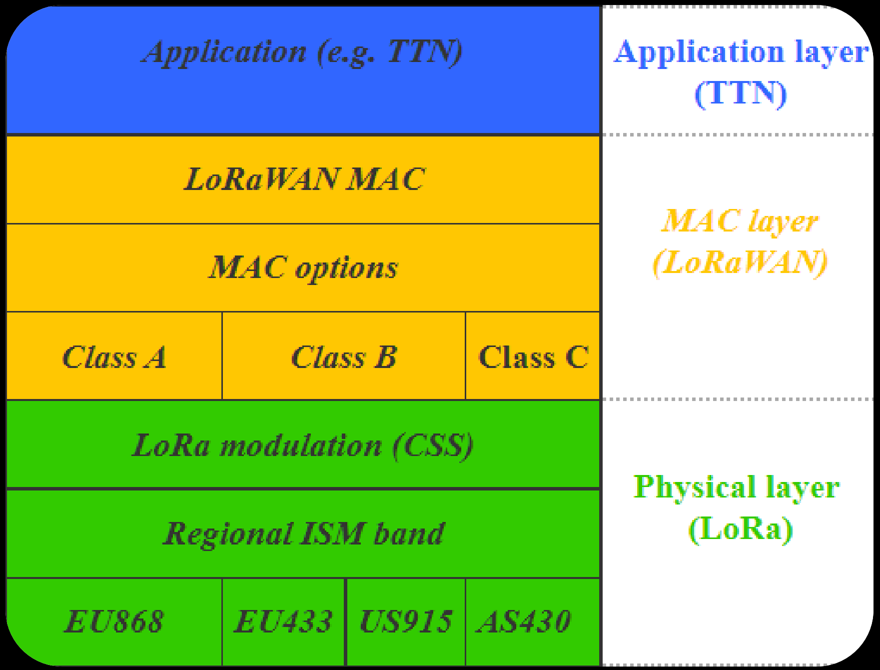
Research gaps

What is LoRa/LoRaWAN?

Over-the-air communication protocol (***LPWAN***):

- **low power consumption;**
- **good coverage** in terms of distance (> 10km);
- **speed** 0.3 - 50 kbps (<u>Uplink</u> & <u>Downlink</u>).

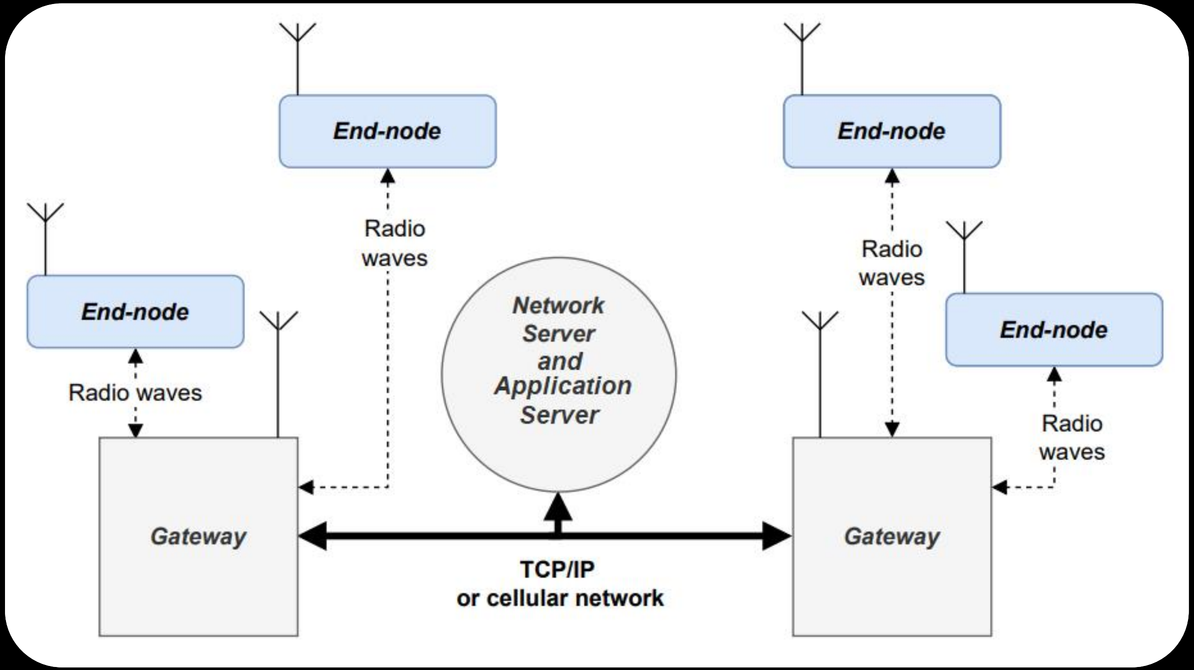




LoRaWAN architecture

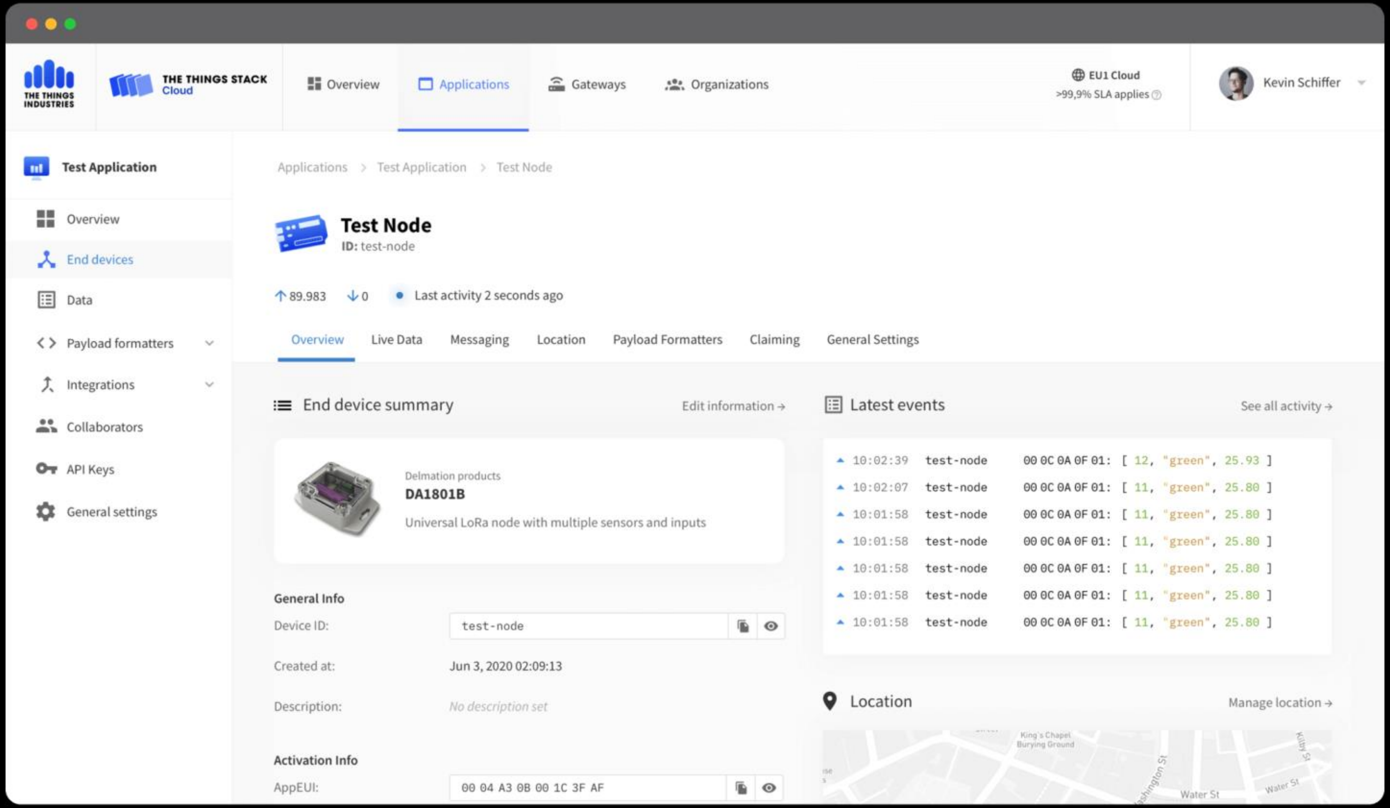
Star network topology, based on ALOHA:

- *End-node*: µP-based system that sends or receives data (Uplink/Downlink);
- *Gateway*: Integrated for merely routing the packets from the end-device to TTN;
- *Network & Application Server*: Servers on which runs a part of Software that guarantees securely processing user data.



The Things Network (TTN)

Freeware and open-source online platform that offers a whole set of functionalities for LoRaWAN => ***APPLICATION LAYER.***
Permits registration and linking of end-nodes and Gateways to different **"Applications".**



Key features:

- <u>Live Data;</u>
- <u>Messaging;</u>
- <u>Payload formatters.</u>

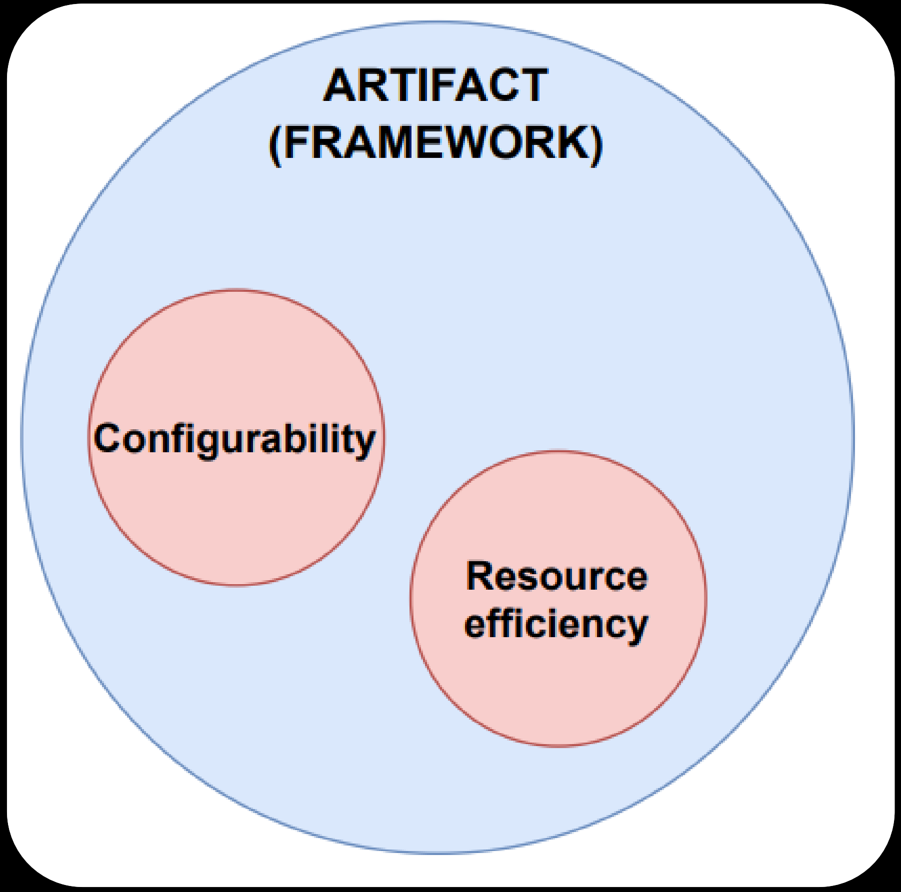
Method

Design science: «Design Science Research is a problem-solving paradigm that seeks to enhance **λ-knowledge** via the creation of innovative artifacts.»

ARTIFACT = Framework.

Core problems => Configurability, data efficiency

12 different iterations to develop and instantiate the final artifact in order to solve the core problems.



Hardware used

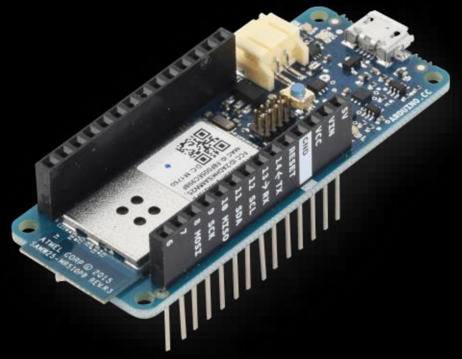
- **Arduino MKR1300 WAN**
  µC board with integrated LoRa module (acting as <u>end-node</u>)

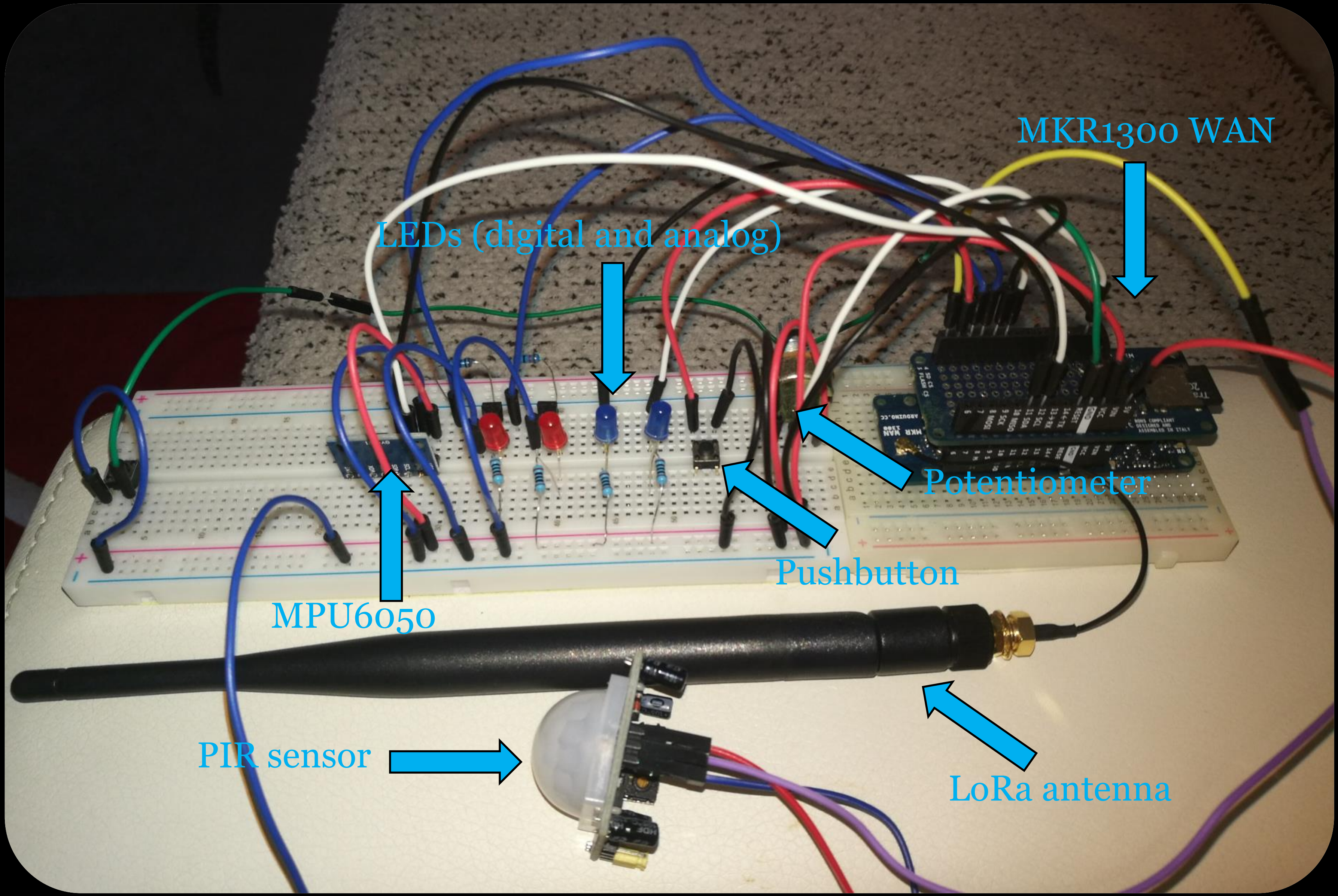
- **Sensors and actuators**
  LEDs, pushbutton, potentiometer, PIR sensor, MPU6050 sensor (temperature)

- **Application layer (TTN)**
  used to visualize incoming data from the end-device and send control and command sequences (Live Data and Messaging tools)

- **LoRa Gateway**
  Used as simple packet forwarder between end-node and application layer

THE THINGS NETWORK

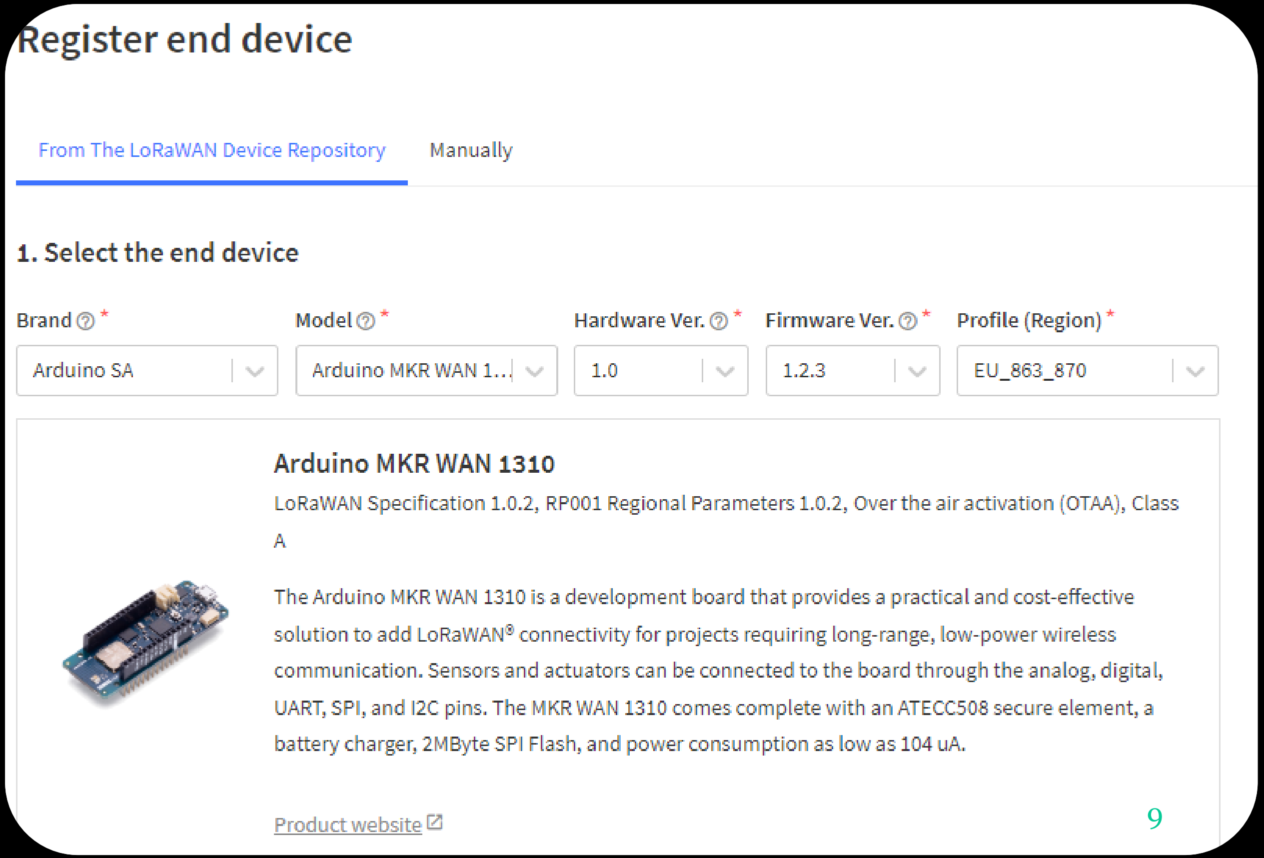



MKR1300 WAN
LEDs (digital and analog)
Potentiometer
Pushbutton
MPU6050
PIR sensor
LoRa antenna

End-node setup and registration

On TTN, ”Add application” option; <u>registration</u> of end-node within application.

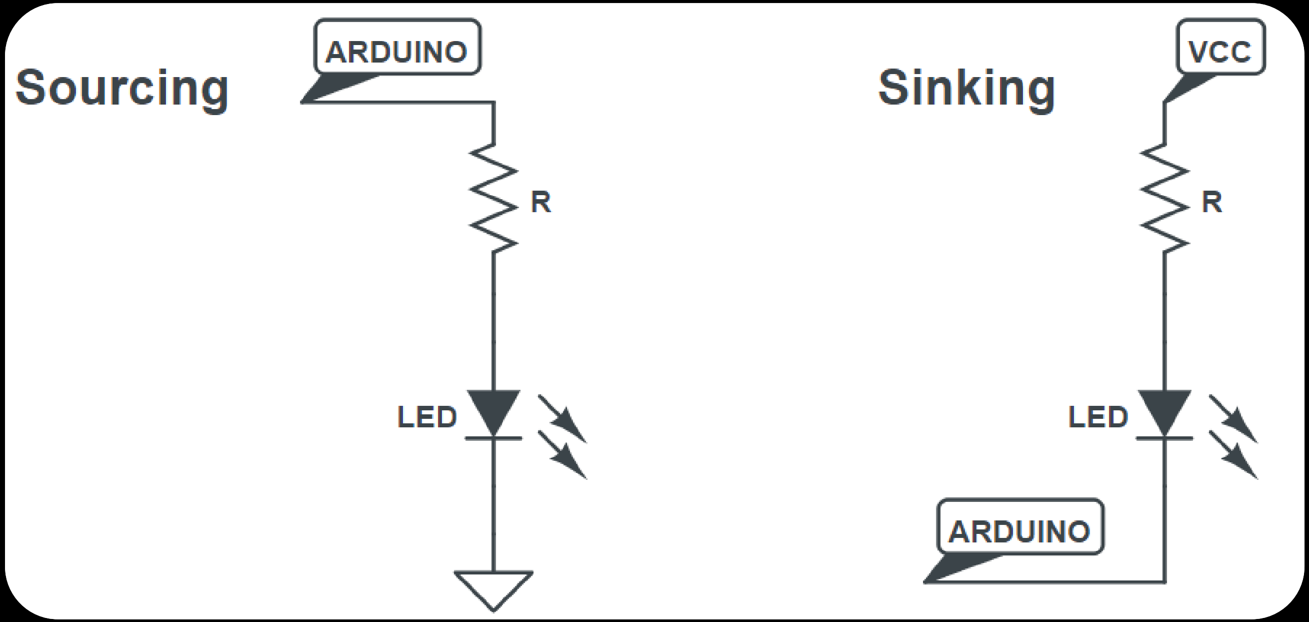
- <u>End-node brand and model</u>;
- <u>Firmware version</u>;
- <u>Frequency plan</u>;
- <u>AppEUI (application identifier)</u>;
- <u>DevEUI (Hard-coded)</u>;

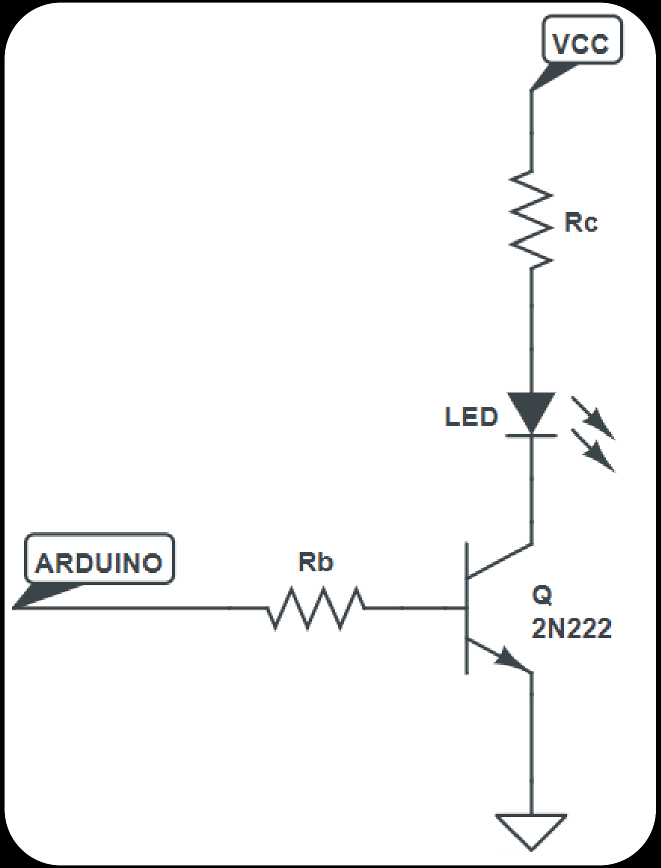


```
LoRaModem modem;

void setup() {
	Serial.begin(115200);
	while (!Serial);
	while (!modem.begin(EU868))
		Serial.println("Failed to start module");

	Serial.print("Your module version is: ");
	Serial.println(modem.version());
	Serial.print("Your device EUI is: ");
	Serial.println(modem.deviceEUI());
}
```

Implementation of digital/analog outputs

- **Digital outputs:** 2 red LEDs in sinking configuration (7mA max. sourcing current).
- **Analog (PWM) outputs:** 2 blue LEDs driven by transistors.

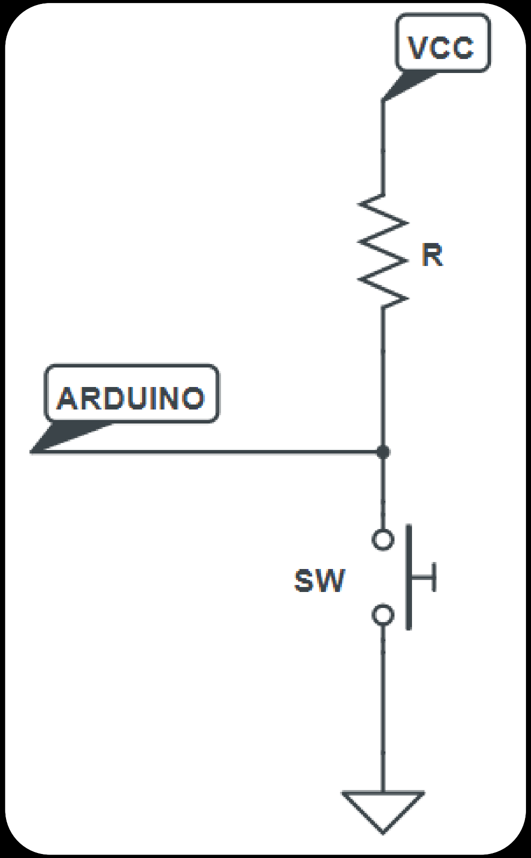




Implementation of digital/analog inputs

- **Digital input:** normally open push-button (with internal pull-up resistor);
- **Analog input:** linear potentiometer; range 0-1023 with *analogRead(PIN)*;

```
#define TOGGLE_LSB(val) (val ^ 0b1)
volatile uint8_t butt = 1;
.....
void buttonChange() {
    butt = TOGGLE_LSB(butt); }
.....
pinMode(PIN_DIGITIN, INPUT_PULLUP);
attachInterrupt((uint8_t)PIN_DIGITIN, buttonChange, CHANGE);
.....
```



Packet structuring

loraPacketBaseBody struct (within loraPacket.h)

- **Opcode:** Used to recognize the type of command being sent (e.g. DIGITOUT, ANALOUT ...);
- **HasNext:** used in order to signal whether there are following packets in the transmission;
- **Pin:** indicating the Arduino pin associated to the packet;
- **Value:** field indicating the value to which the pin has to be set.

Fields name	Size (bits)
Opcode	8
HasNext	8
Pin	8
Value	16

Packet structuring

loraPacketBaseBody struct ⊆ union as array (**several sequential packets**)

STATIC ALLOCATION of packets (*#define MAX PACKET SIZE 51*)

```
typedef union {
    loraPacketBaseBody gpio_body;
    //TO BE DONE (UART, I2C, MODBUS, DALI, ...)
} loraPacketBody;

typedef struct {
    loraPacketBody body[MAX_PACKET_SIZE];
} loraPacket;
```

Example

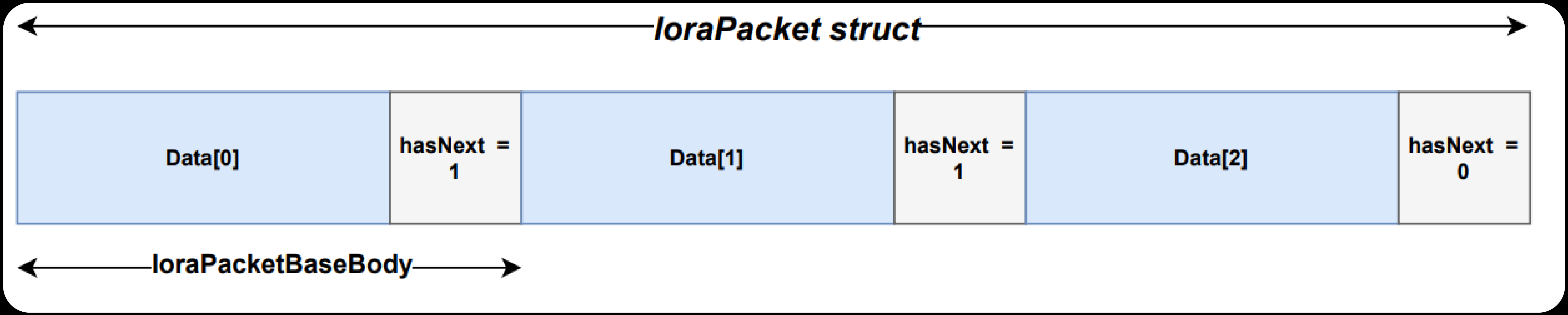
loraPacket my_packet;

initPacket(&my_packet);

packGpioData(&my_packet, DIGITIN, PIN_10, 0);

packGpioData(&my_packet, ANALIN, PIN_13, analogRead(PIN_13));

packGpioData(&my_packet, DIGITIN, PIN_12, 1);



Uplink & Downlink messages

Two buffers for sending and receiving data: **uint8_t rcvBuffer[SIZE], sndBuffer[SIZE]**

```
//TRASMIT
modem.beginPacket();
modem.write(sndBuffer, sndBufferCnt);
modem.endPacket(false);

//RECEIVE
if(modem.available())
        rcvBuffer[rcvBufferCnt++] = modem.read();
```

Downlink scenario

1. Creation and initialization of a loraPacket;
2. In case new data is present (modem.available()), the data is read and concatenated in rcvBuffer;
3. the data present in the rcvBuffer is deserialized and inserted into the loraPacket;
4. rcvBuffer is emptied;
5. the loraPacket is iterated through a while loop within the main function and associated commands are executed (e.g. turn on the LED, output a PWM of 50%) .

Deserialization

```
while(has_next) {
    switch(data[index]) { //Get opcode
        case DIGITOUT:
        case ANALOUT:
            packet.body[body_index].gpio_body.opcode = data[index]; //Opcode
            packet.body[body_index].gpio_body.pin = data[++index]; //Pin
            packet.body[body_index].gpio_body.val = (((uint16_t)(data[++index]) << 8) | ((uint16_t)data[++index]));
            has_next = data[++index];
            packet.body[body_index].gpio_body.hasNext = has_next; //HasNext
        break; //TO BE DONE (OTHER CASE CONDITIONS)
    }
    body_index++; index++;
}
```

Uplink scenario

1. Packing input data (either analog or digital) within an empty loraPacket (e.g. packGpioData(&new packet, DIGITIN, PIN_DIGITIN, butt));
2. Serialization of the data present within the loraPacket on the sndBuffer array;
3. In the case sndBuffer ≠ empty (size > 0), the buffer is provided to the LoRa functions, and the data is transmitted;
4. sndBuffer is finally emptied.

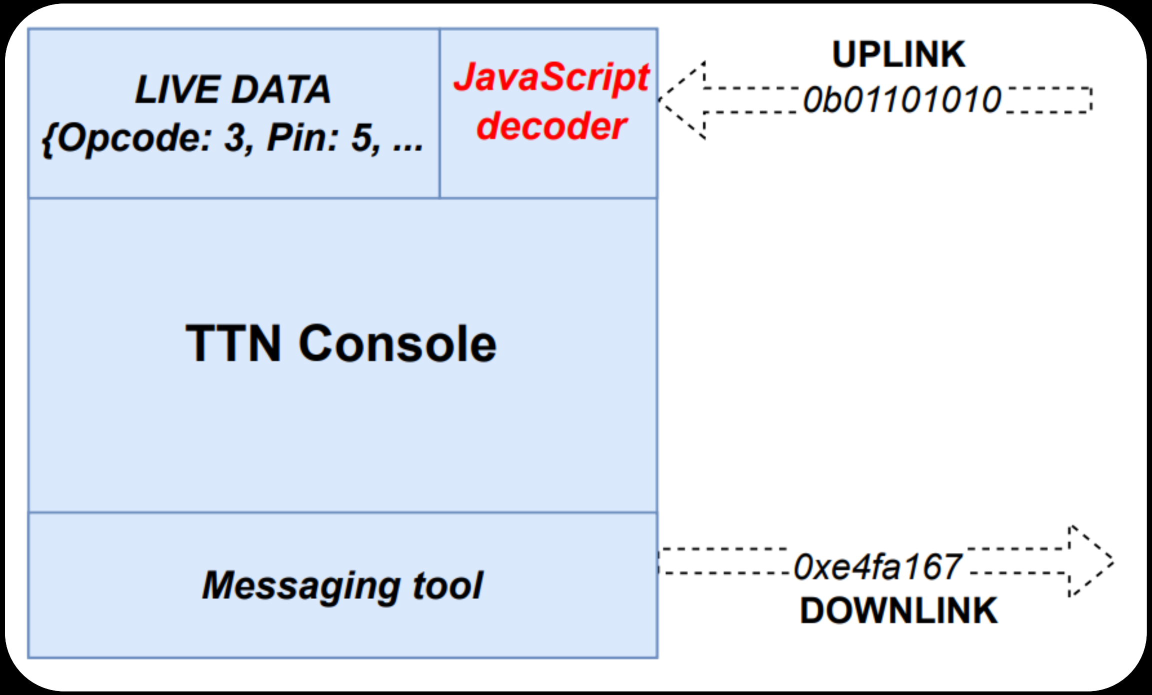
Serialization

```
while(has_next) {
    switch(packet.body[body_index].gpio_body.opcode) {
        case DIGITIN:
        case ANALIN:
            data[index++] = packet.body[body_index].gpio_body.opcode; //Opcode
            data[index++] = packet.body[body_index].gpio_body.pin; //Pin
            data[index++] = (uint8_t)((packet.body[body_index].gpio_body.val & 0xFF00) >> 8);  //Val
            data[index++] = (uint8_t)(packet.body[body_index].gpio_body.val & 0xFF);  //Val
            has_next = packet.body[body_index].gpio_body.hasNext;
        break; //TO BE DONE (OTHER CASE CONDITIONS)
    }
    data[index++] = has_next;  //HasNext
    body_index++;
}
*count = index;
```

Configuration of application layer (TTN)

- **Live data (data visualization) - Uplink:** Bitstream decoded similarly as deserialization function on end-node (built-in JavaScript decoder).

- **Messagging tool - Downlink:** Data sent in HEX format to end-node, e.g. **0x040F008000** = 04: ANALOUT, 0F (15): pin, 0080 (128): value, 00: no hasNext



Implementation of the I2C and UART interfaces

Definition of two new structs:
- **loraPacketUartBody** (for managing ASCII characters over UART);
- **loraPacketTWIBody** (for managing I2C-devices).

Fields name	Size (bits)
Opcode	*8*
HasNext	*8*
Value	*8*

Fields name	Size (bits)
Opcode	*8*
HasNext	*8*
Address	*8*
R/W	1
Register	16
Value (OPTIONAL)	16

```
case TWIRX:
        Wire.beginTransmission(downdata.body[body_index].twi_body.addr);
        Wire.write(downdata.body[body_index].twi_body.reg); //Internal register of I2C sensor

        if (downdata.body[body_index].twi_body.rw) //Write operation
        {
                Wire.write(downdata.body[body_index].twi_body.val);
                Wire.endTransmission();
        }

        else //Read operation; 2 sequential 8 bit registers in MPU6050 for temperature
        {
                Wire.endTransmission();
                Wire.requestFrom(downdata.body[body_index].twi_body.addr, 1);

                if (twiVal == 0) //First 8 bits
                    twiVal = Wire.read();
                else //Next 8 bits
                {
                    twiVal <<= 8;
                    twiVal |= Wire.read();
                }
        }

        hasNext = downdata.body[body_index].twi_body.hasNext;
break;
```

Compression: phase 1

Reduction of HasNext field; from 8 bit to 1 bit flag.
Moreover, concatenation of the field to Opcode:

(Opcode << 1) | HasNext

REASON: HasNext domain is binary {TRUE, FALSE}

PROS: Reduced overhead (-8 bits)
CONS: Opcode reduced from 8 bits to 7 bits.

Compression: phase 2

Introduction of fields **isHomog** (1 bit) and **cntNext** (8 bit); further reduction of Opcode from 7 to 6 bits.

(Opcode << 2) | (HasNext << 1) | isHomog

Avoids <u>repetition</u> in sending or receiving the opcode in case of homogeneous data (e.g. ASCII sequence ”Hello”)

BEFORE:

Opcode	hasNext	Value	Opcode	hasNext	Value	Opcode	hasNext	....
UARTTX	1	'H'	UARTTX	1	'e'	UARTTX	1	....

AFTER:

Opcode	isHomog	CntNext	Value	Value	Value	....
UARTTX	1	4	'H'	'e'	'l'	....

Compression: phase 3

Usage of an existing 8-bit MAC layer field: **fPort; c**ompression of <u>first</u> 8 bits within fPort**.**

```
//TRANSMIT
modem.setPort(sndBuffer[0]);
modem.beginPacket();
modem.write(&(sndBuffer[1]), sndBufferCnt - 1); //Skip first byte, sent through fPort

//RECEIVE
rcvBuffer[rcvBufferCnt++] = modem.getDownlinkPort();
while (modem.available())
        rcvBuffer[rcvBufferCnt++] = modem.read();
```

BEFORE:

fPort	Opcode + isHomog		CntNext	Value	Value	Value	Value	....
NULL	UARTXX	1	4	'H'	'e'	'l'	'l'	....

AFTER:

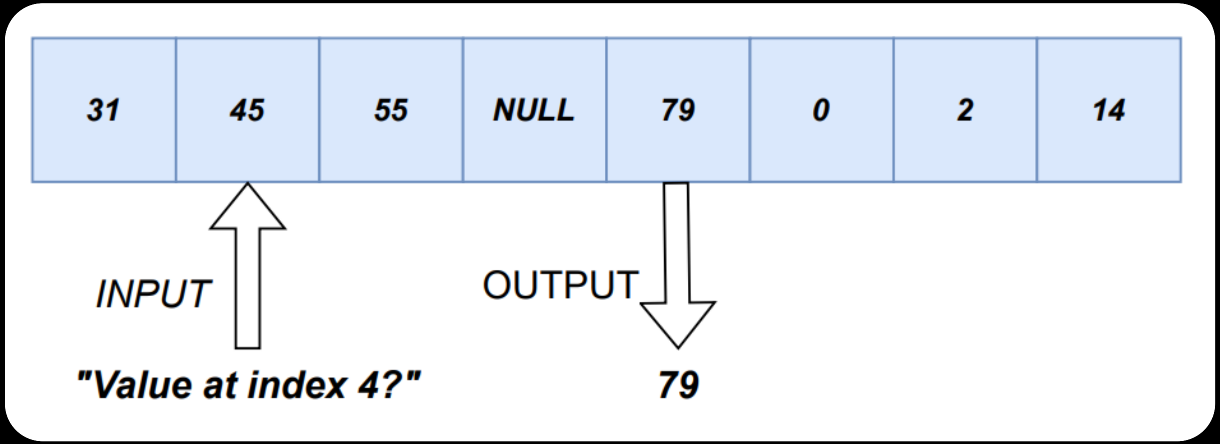
fPort	CntNext	Value	Value	Value	Value	Value
UARTTX + isHomog (1)	4	'H'	'e'	'l'	'l'	'o'

Configurability

Framework's ability to be adaptable to the user's needs: allows sending configuration commands to the end-node so that I/O ports can be dynamically managed.

Achieved through 3 **look-up tables**:

- *uint8_t LUT_PIN_STATUS[255];*
- *uint32_t LUT_UART_SPEED;*
- *uint8_t LUT_TWI.*

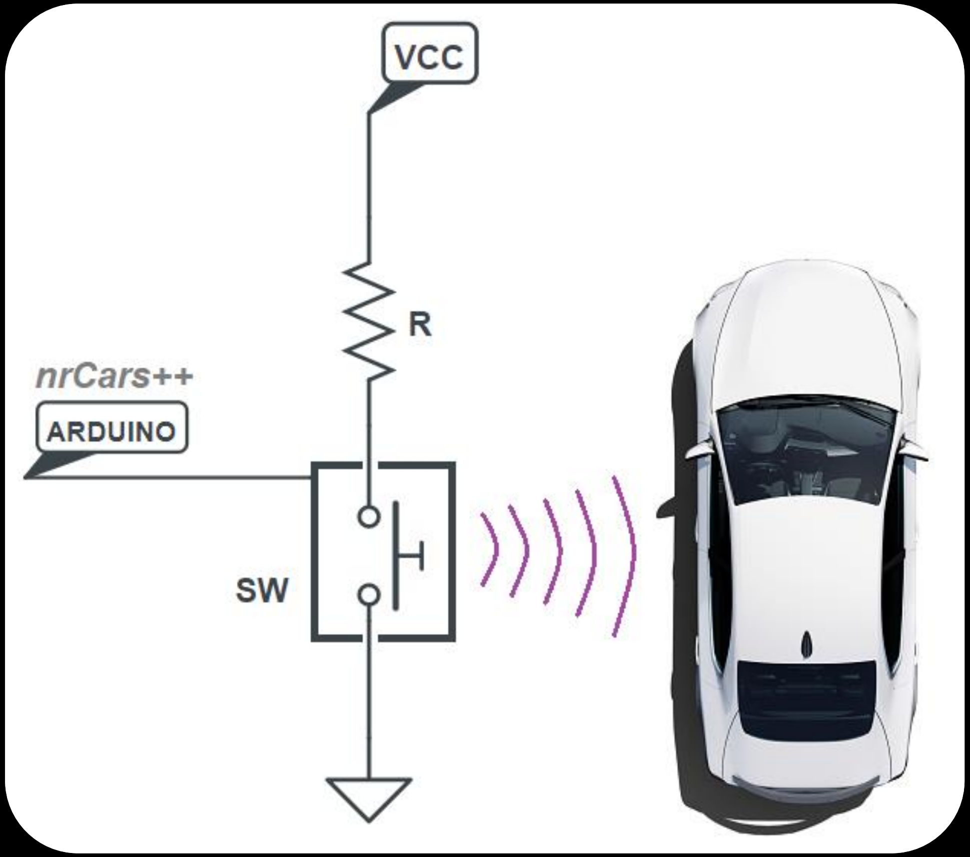


e.g. ***setAsDigitalOutput(5)*** configures pin 5 as DIGITOUT and performs:
pinMode(pin, OUTPUT); LUT_PIN_STATUS[5] = DIGITOUT;
isDigitalOutput(uint8_t pin) returns 0 or 1 depending on the actual status of the pin.

Digital counter feature

It was necessary to transmit data, which in addition to indicating two simple states (ON and OFF), was able to indicate a counter value.

```
uint8_t count = 0;
…
void ISRIncrementCount() {
    if(debounce_timer == 0 || (millis() > debounce_timer + DEBOUNCE_MS)) {
        debounce_timer = millis();
        count = count + 1;
    }
}
```



Compression: phase 4

HasNext flag removed, since was not really necessary.
Opcode restored to the 7-bit size (instead of 6 bit).

isHomog flag put as *MSB* to reduce variability.

(isHomog << 7) | Opcode;

PROS: 7-bit Opcode, less variability

Compression: phase 5

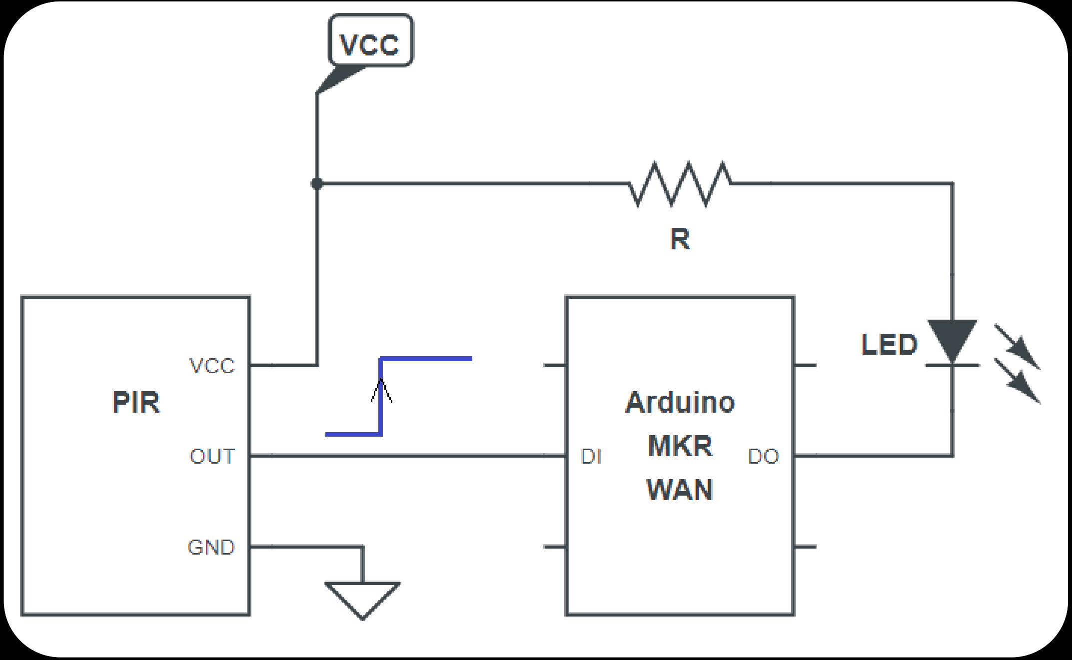
More than compression, for **standardization purposes**
=> **DE-ASSOCIATION OF OPCODE FROM ANY FIELD.**

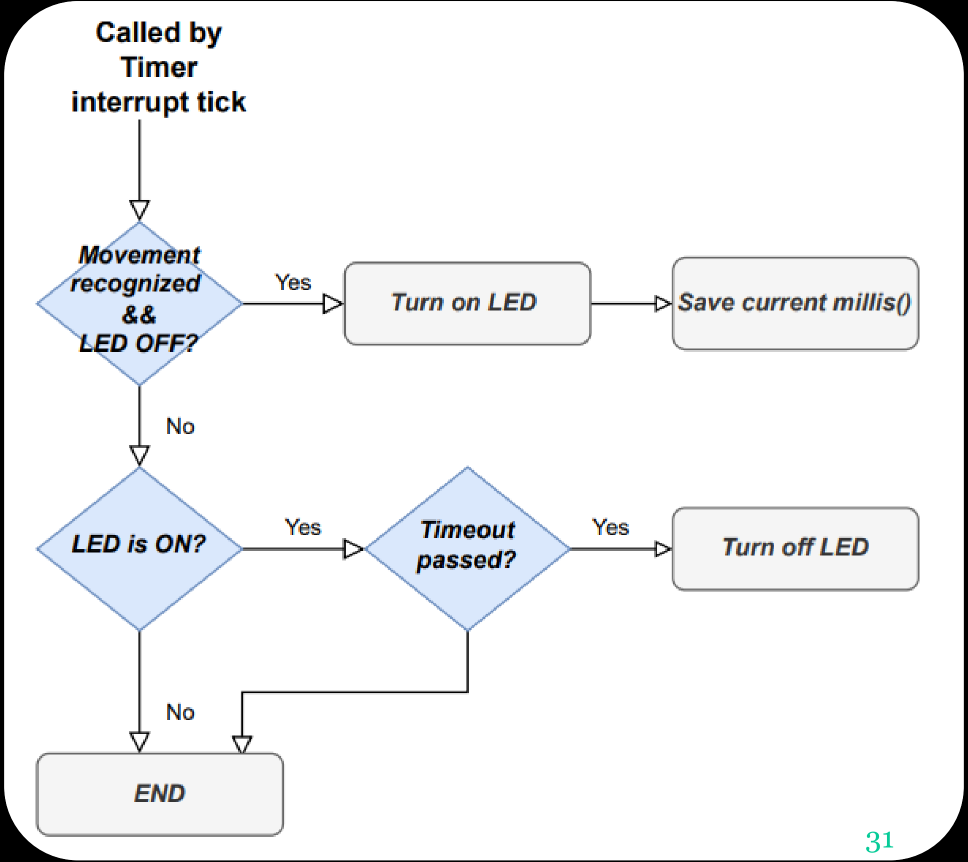
- Homogeneous packets sent in same way, however without isHomog field;
- Non-Homogeneous packets introduce NON_HOMOG MACRO as first field, which is sent through fPort.

PROS: De-association of Opcode (8 bit)
CONS: +8 bit in case of non-homogeneous packets.

Smart movement detector feature

Decentralization in the decisions:
PIR (proximity) sensor attached to digital output and decides <u>whether</u> to activate the output and for <u>how long</u> to activate it.





Conclusions

- The framework (mainly proof of concept) demonstrates an exchange of compressed and complex data structures on a network that structurally was not designed for large amounts of data.
- Moreover, key features such as configurability were provided in order to guarantee customization.
- Flexible nature of the framework permits extensions to handle other I/O peripherals.

"Design is where science and art break even." - **Robin Matthews, professor and activist**

Thank you for the attention …