

# Digital embedded electronics

J.-M Friedt

FEMTO-ST/time & frequency department

`jmfriedt@femto-st.fr`

slides at `jmfriedt.free.fr`

December 18, 2025

## List of presentations:

7 lessons introducing baremetal programming of the STM32:

1. Digital electronics basics  
L3/bachelor: analog aspects, power supply and consumption, datasheet analysis  
Overview of the various peripherals (RS232, SPI, timer, ADC)  
data format (size/encoding), masks, architecture  
Reminders on Atmega32U4 (Makefile, compilation, masks ...)
2. First steps with the STM32, peripheral addresses, internal architecture
3. gcc operation and optimizations:  
preprocess-compiler-assembler-linker, C to assembly language translation, pointers
4. libraries and software architecture (separating algorithm/hardware), simulators & stubs  
libopencm3, newlib & stubs, resources needed to execute libraries
5. Communication bus parallel/serial, synchronous/asynchrone
6. arithmetics on embedded systems  
integers v.s floating point number, converting an algorithm expressed with floats to integers, timers
7. interrupts and analog data acquisition, sampling rate  
interrupt vectors, clock management on STM32, ADC

All supporting material (bachelor and master) on <http://jmfriedt.free.fr>

## Linking with the static library

- ▶ Issue: when linking with the static library `libmylib.a` (archive of objects) with `-lmylib`, all functions in the object are included in the final ELF executable
- ▶ all functions within each archived (`ar rcs`) object will be linked by default to the executable <sup>1</sup>
- ▶ compilation flags `-ffunction-sections` splits the object with one section for each function *when compiling the library...*
- ▶ and the linker option `--gc-sections` (gcc option: `-Wl,--gc-sections`) will only link referenced functions <sup>2</sup>
- ▶ check with `arm-none-eabi-objdump -dSt exec.elf` to see if unused functions have been linked.

---

<sup>1</sup>[https://www.reddit.com/r/C\\_Programming/comments/1cwplcv/are\\_all\\_functions\\_of\\_a\\_linked\\_objectlibrary\\_files/](https://www.reddit.com/r/C_Programming/comments/1cwplcv/are_all_functions_of_a_linked_objectlibrary_files/)

<sup>2</sup><https://stackoverflow.com/questions/31327637/stripping-unused-library-functions-dead-code-from-a-static-executable>

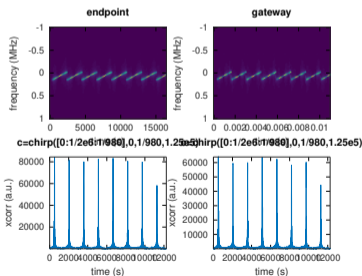
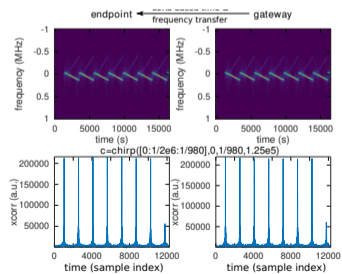
## Wireless sensor networks: outline

- ▶ “Internet of Things” (IoT)
- ▶ long range communication between sensor nodes (endpoints) and gateways (running GNU Linux)
- ▶ routing data over the internet to a network server
- ▶ sharing the data with display/processing software

Reliance on multiple networking protocols, low level layers: radiofrequency communication, UDP (OSI 4 Transport), MQTT (OSI 7 Application layer) over TCP (OSI 4 Transport)/IP (OSI 3 Network)

# LoRa

- ▶ Chirped modulation on carrier frequencies in ISM bands (434 MHz, 868(EU)/915(US) MHz, 2400 MHz)
- ▶ Only same chirp length/bandwidth will correlate
- ▶ Narrowband signal: 125, 250 or 500 kHz
- ▶ Spreading Factor (SF): 7 (fastest) to 12 (slowest)
- ▶ Tradeoff between high data rate (short emission duration) and range
- ▶ Coding Rate (CR), the higher the more repetitions<sup>3</sup>



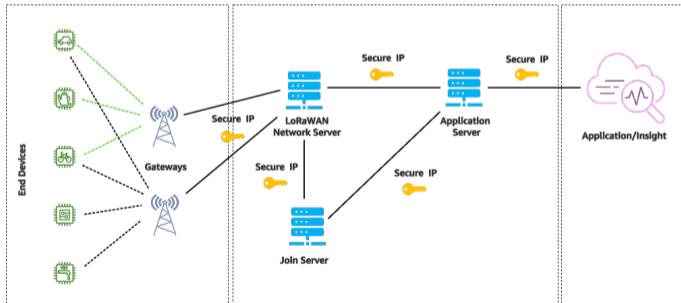
SF	bandwidth	Datarate	Sensitivity
7	125 kHz	5.5 kb/s	-123 dBm
7	125 kHz	10.9 kb/s	-123 dBm
7	125 kHz	21.9 kb/s	-123 dBm
8	125 kHz	3.1 kb/s	-126 dBm
9	125 kHz	1.8 kb/s	-126 dBm
9	250 kHz	3.5 kb/s	-126 dBm
...			
12	125 kHz	0.6 kb/s	-137 dBm

$$R_{b/s} = SF \times \frac{4/(4+CR)}{2^{SF}/BW}$$

<sup>3</sup><https://www.rfwireless-world.com/calculators/lora-data-rate-calculator>

# LoRaWAN

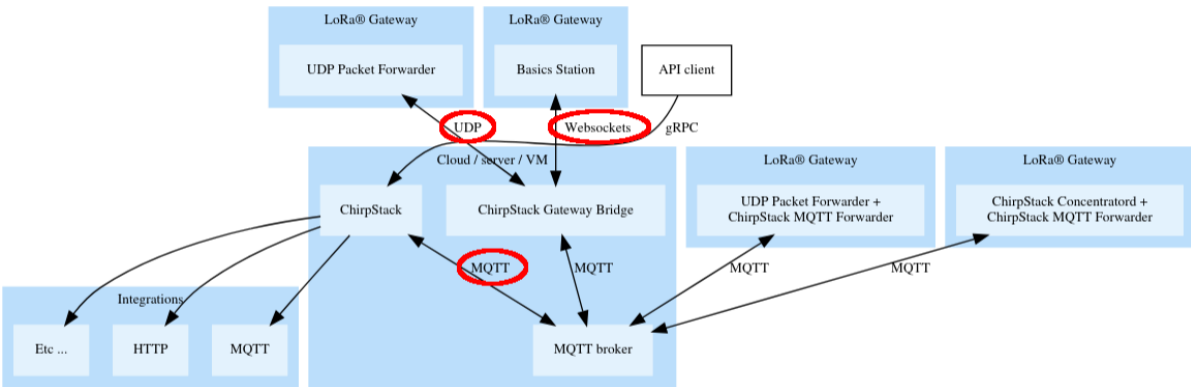
- ▶ Star-shaped network: endpoints initiate communications with the gateway
- ▶ Endpoints are low power devices, yet running the LoRaWAN stack
- ▶ Executive environments: RIOT-OS, ZephyrOS, FreeRTOS<sup>4</sup>
- ▶ Gateways are connected to the internet: running GNU/Linux
- ▶ Gateways communicate over the internet to network servers...
- ▶ ...over an additional layer on top of TCP and UDP: MQTT
- ▶ MQTT broadcasts messages to Application Servers<sup>5</sup>



<sup>4</sup><https://github.com/FreeRTOS/Lab-Project-FreeRTOS-LoRaWAN>

<sup>5</sup>Figure from

# ChirpStack (open-source LoRaWAN Network Server) architecture<sup>6</sup>



<sup>6</sup><https://www.chirpstack.io/docs/architecture.html>

## MQTT demonstration: topics

MQTT server is called “broker”: multiple clients connect to exchange information

```
$ dpkg -l | grep mosquitto
ii  libmosquitto1:amd64  2.0.21-1  amd64  MQTT version 5.0/3.1.1/3.1 client library
ii  mosquitto            2.0.21-1  amd64  MQTT version 5.0/3.1.1/3.1 compatible message broker
ii  mosquitto-clients   2.0.21-1  amd64  Mosquitto command line MQTT clients
$ mosquitto_sub -h 127.0.0.1 -v -t 'test/topic'
$ mosquitto_pub -h 127.0.0.1 -t 'test/topic' -m "hello" # detected
$ mosquitto_pub -h 127.0.0.1 -t 'test/topic2' -m "hello" # not detected
$ mosquitto_sub -h 127.0.0.1 -v -t '#' # detect all topics
```

# MQTT demonstration and interaction with the gateway

```
$ mosquitto_sub -h 192.168.1.170 -v -t '#' # detect all topics

gateway/0016c001ff10d6dc/state/conn {"gatewayId":"0016c001ff10d6dc","state":"ONLINE"}
gateway/0016c001ff10d6dc/event/up {"phyPayload":"APh/AP//VAAAYy2U/v/OgAC2T5pnmGQ=","txInfo":{"frequency":868100000,"modulation":{"lora":{"bandwidth":125000,"spreadingFactor":12,"codeRate":"CR_4_5"}}},"rxInfo":{"gatewayId":"0016c001ff10d6dc","uplinkId":52242,"gwTime":"2025-12-02T07:33:43.209867592Z","rssi":-127,"snr":-22.25,"rfChain":1,"context":"6gLuiw==","crcStatus":"CRC_OK"}}

gateway/0016c001ff10d6dc/event/up {"phyPayload":"QL+UiwCAcAh99ZHneN8M7n1kdlJLyg==","txInfo":{"frequency":868500000,"modulation":{"lora":{"bandwidth":125000,"spreadingFactor":12,"codeRate":"CR_4_5"}}},"rxInfo":{"gatewayId":"0016c001ff10d6dc","uplinkId":64813,"gwTime":"2025-12-02T07:35:06.669407210Z","rssi":-114,"snr":-10.25,"channel":2,"rfChain":1,"context":"7vxYzQ==","crcStatus":"CRC_OK"}}

gateway/0016c001ff10d6dc/event/up {"phyPayload":"gEkCoQEaiAABX0ki0kNq2zxdgEfXDoJV","txInfo":{"frequency":867300000,"modulation":{"lora":{"bandwidth":125000,"spreadingFactor":7,"codeRate":"CR_4_5"}}},"rxInfo":{"gatewayId":"0016c001ff10d6dc","uplinkId":53991,"gwTime":"2025-12-02T07:35:12.996665641Z","rssi":-133,"snr":-1,"channel":4,"context":"710DNg==","crcStatus":"CRC_OK"}}
```

How to decode the payload of these JSON messages?

# Base64 encoding/decoding

How to send binary data?

- ▶ Example of ACARS: ASCII<sup>7</sup> sentences (7 bit characters) framed by STX and ETX (ASCII 02 and 03)
- ▶ Some 8-bit binary payload might be mistaken for frame definition
- ▶ Historically: binary attachments in email handled by uuencode and uudecode, file oriented

```
$ echo hello | uuencode -  
begin 664 -  
&:&5L;&\*  
'  
end
```

- ▶ Base64: encode 8-bit bytes as slightly more compact<sup>8</sup> strings using 64 printable characters, stream oriented

```
$ echo "hello" | base64  
aGVsbG8K  
$ echo "aGVsbG8K" | base64 -d  
hello
```

Input stream (bytes) → split at 6-bit packets (3 bytes become 4 6-bit packets) → associate 1 of 64 printable characters<sup>9</sup> ; pad missing bits with "=" sign.

---

<sup>7</sup>man ascii

<sup>8</sup>[https://www.gnu.org/software/sharutils/manual/html\\_node/uuencode-base64.html](https://www.gnu.org/software/sharutils/manual/html_node/uuencode-base64.html)

<sup>9</sup>RFC4648, *The Base16, Base32, and Base64 Data Encodings* (2006) © <https://datatracker.ietf.org/doc/html/rfc4648>

# Base64 encoding/decoding: gateway payload analysis

Demonstration:

```
$ echo "hello" | basenc --base2msb | sed 's/.\{6\}/& /g'
011010 000110 010101 101100 011011 000110 111100 001010
 26      06      21      44      27      06      60      10
  a      G      V      s      b      G      8      K      no "="
```

so we can decode:

```
$ echo "APh/AP//VAAAYy2U/v/0gAC2T5pnmGQ=" | base64 -d | xxd
00000000: 00f8 7f00 ffff 5400 0063 2d94 feff f480  ....T..c-.....
00000010: 00b6 4f9a 669c 64          ..0.f.d
```

```
$ echo "QL+UiwCAcAh99ZHneN8M7n1kd1JLyg==" | base64 -d | xxd
00000000: 40bf 948b 0080 7008 7df5 91e7 78df 0cee  @.....p.}...x...
00000010: 7d64 7652 4bca          }dvRK.
```

```
$ echo "gEkCoQEAIaABX0ki0kNq2zxdgEfXDoJV" | base64 -d | xxd
00000000: 8049 02a1 0100 8800 015c e922 3a43 6adb  .I.....\." :Cj.
00000010: 3c5d 8047 d70e 8255          <].G...U
```

MAC PHYPayload starting with MHDR<sup>10</sup>:

0x8 = Confirmed Data Up ; 0x4=Unconfirmed Data Up ; 0x0 = Join Request

<sup>10</sup>[https://lora-alliance.org/wp-content/uploads/2020/11/2015\\_-\\_lorawan\\_specification\\_1r0\\_611\\_1.pdf](https://lora-alliance.org/wp-content/uploads/2020/11/2015_-_lorawan_specification_1r0_611_1.pdf)

## Configuration file: `chirpstack-mqtt-forwarder/20-1ns.toml`

```
# ChirpStack MQTT Forwarder publishes to the following topics:
# * [Prefix/]gateway/[Gateway ID]/event/[Event]
# * [Prefix/]gateway/[Gateway ID]/state/[State]
# And subscribes to the following topic:
# * [Prefix/]gateway/[Gateway ID]/command/[Command]
#
# The topic prefix can be used to define the region of the gateway.
# Note, there is no need to add a trailing '/' to the prefix. The trailing
# '/' is automatically added to the prefix if it is configured.
topic_prefix="eu868"

# Use JSON encoding instead of Protobuf (binary).
#
# Note, only use this for debugging purposes.
json=false

# MQTT server (e.g. scheme://host:port where scheme is tcp, ssl or ws)
server="tcp://127.0.0.1:1883"
```

## Endpoint: Seeedstudio Wio-E5 Mini<sup>13</sup>

- ▶ STM32WLE5JC programmed with `arm-none-eabi-*`
- ▶ Meshtastic firmware relying on platformio:
  - ▶ `python3 ./get-platformio.py`<sup>11</sup>
  - ▶ `platformio.ini` configuration file: `default_envs = wio-e5`
  - ▶ `$HOME/.platformio/penv/bin/platformio run`
  - ▶ `$HOME/.platformio/penv/bin/platformio run -t upload`
- ▶ RIOT-OS<sup>12</sup> and examples at [https://gricad-gitlab.univ-grenoble-alpes.fr/thingsat/seed/-/tree/main/firmware/seed\\_lora-e5-mini/riot/apps/field\\_test\\_device](https://gricad-gitlab.univ-grenoble-alpes.fr/thingsat/seed/-/tree/main/firmware/seed_lora-e5-mini/riot/apps/field_test_device)  
Flashing the STM32:

```
openocd -d2 -s $OPENOCD/scripts -f interface/stlink.cfg
-c "transport select hla_swd" -f target/stm32wlx.cfg
-c "program {./meshtastic_firmware.bin} 0x08000000 verify reset; shutdown;"
```

---

<sup>11</sup>from

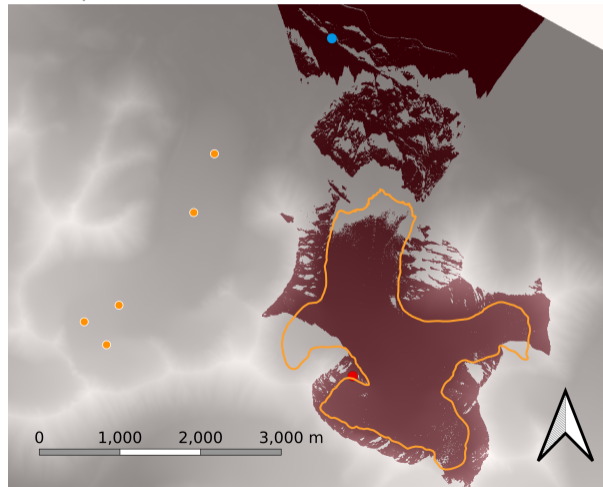
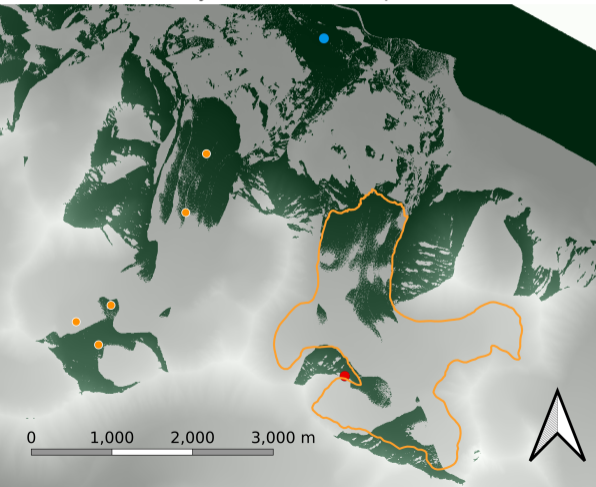
<https://raw.githubusercontent.com/platformio/platformio-core-installer/master/get-platformio.py>

<sup>12</sup><https://github.com/RIOT-OS/RIOT>

<sup>13</sup>[https://wiki.seeedstudio.com/LoRa\\_E5\\_mini/](https://wiki.seeedstudio.com/LoRa_E5_mini/)

## From star to mesh network

- ▶ LoRaWAN: endpoints initiate communication with a gateway connected to the internet: star shape
- ▶ What if the gateway is not in line of sight of a gateway?
- ▶ Need for a relay  $\Rightarrow$  Meshtastic protocol, however not compatible with LoRaWAN out of the box



# Wireless meshed network over LoRa: Meshtastic

- ▶ firmware at <https://github.com/meshtastic/firmware>
- ▶ host tools at <https://github.com/meshtastic/meshtastic> or `pip install meshtastic`
- ▶ Meshtastic uses different frequencies and synchronization word than LoRaWAN: update the Meshtastic firmware to match LoRaWAN<sup>14</sup>
- ▶ Broadcast Meshtastic message (`meshtastic --sendtext "hello"`) and record the encrypted/encoded message received by the MQTT broker (`b64="////////wocFVBa0KYmYjEAKsVlwbSva6DiGOVM"`)
- ▶ convert the received payload from base64 to binary: `raw=base64.b64decode(b64)`
- ▶ extract the source MAC and unique ID to form the cryptographic nonce (little endian)  

```
_from=raw[4:8];_from=int.from_bytes(_from,byteorder='little')  
_id=raw[8:12] ;_id=int.from_bytes(_id,byteorder='little')  
nonce=struct.pack("<q",_id)+struct.pack("<q",_from) # <: little endian
```
- ▶ extract payload: `_payload=raw[16:]`

---

<sup>14</sup>[https://github.com/jmfriedt/RIOT\\_NyAlesund/blob/main/2512\\_meshtastic/meshtastic\\_diff.patch](https://github.com/jmfriedt/RIOT_NyAlesund/blob/main/2512_meshtastic/meshtastic_diff.patch)

# Wireless meshed network over LoRa: Meshtastic

- ▶ use the Meshtastic defined<sup>15</sup> key for decryption (AES-128):

```
key = "1PG70iApB1nvwP+rz05pAQ==" # {0xd4, 0xf1, 0xbb, 0x3a, ...}
key_bytes = base64.b64decode(key.encode('ascii'))
cipher = Cipher(algorithms.AES(key_bytes), modes.CTR(nonce), backend=default_backend())
decryptor = cipher.decryptor()
decrypted_bytes = decryptor.update(_payload) + decryptor.finalize()
```

- ▶ use the Meshtastic defined protobuf<sup>16</sup> to extract fields and contents data organization:

```
d=mesh_pb2.Data();d.ParseFromString(decrypted_bytes)
```

- ▶ Enjoy: 

```
portnum: TEXT_MESSAGE_APP
payload: "Hello"
```

---

<sup>15</sup><https://docs.rs/meshtastic/latest/meshtastic/protobufs/struct.ChannelSettings.html>: “Those bytes are mapped using the following scheme: 0 = No crypto 1 = The special “default” channel key: {0xd4, 0xf1, 0xbb, 0x3a, 0x20, 0x29, 0x07, 0x59, 0xf0, 0xbc, 0xff, 0xab, 0xcf, 0x4e, 0x69, 0x01}”

<sup>16</sup><https://protobuf.dev/programming-guides/encoding/>