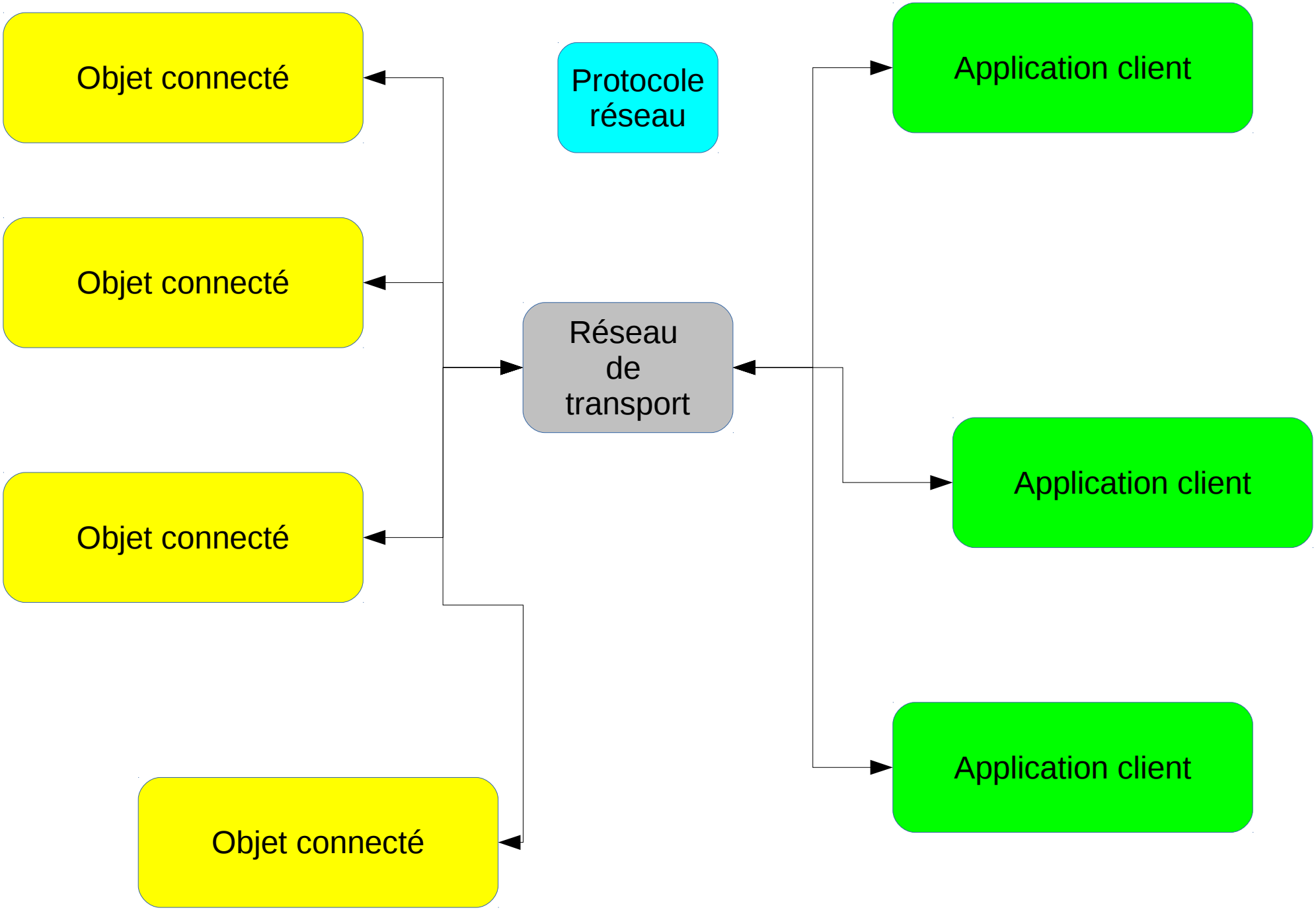
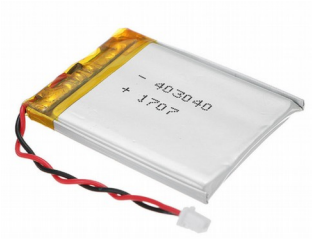
Internet des objets

21/02/2019

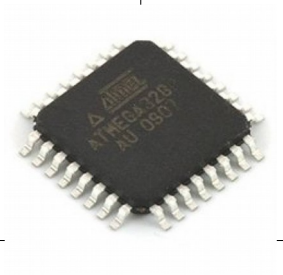

Objet connecté
Objet connecté
Objet connecté
Objet connecté
Protocole réseau
Réseau de transport
Application client
Application client
Application client

Objet connecté ?

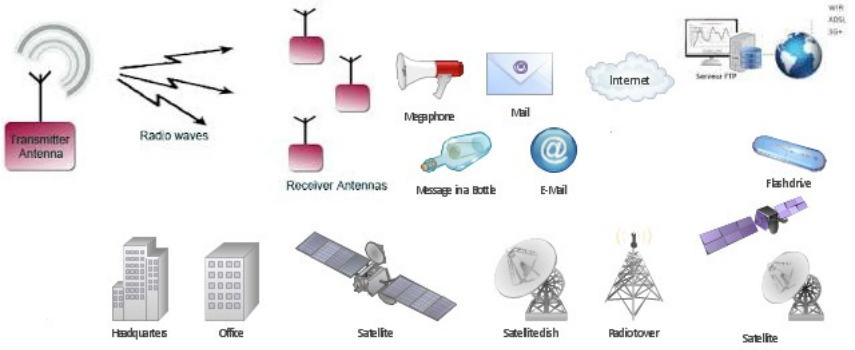
ENERGIE

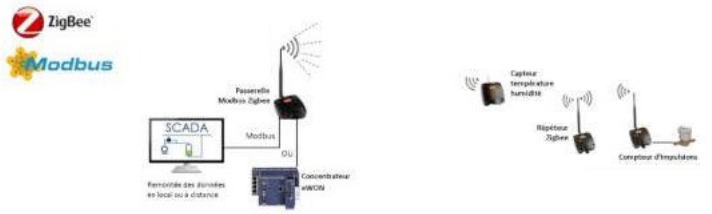


CAPTEURS & ACTIONNEURS



MICRO-
CONTRO
LEUR





MOYEN DE
COMMUNICATION

Applications client ?

Navigateur WEB
Application spécifique :

- C,
- C++,
- python
- *.etc.*

Réseau
de
Transport ?

IEE 802.3

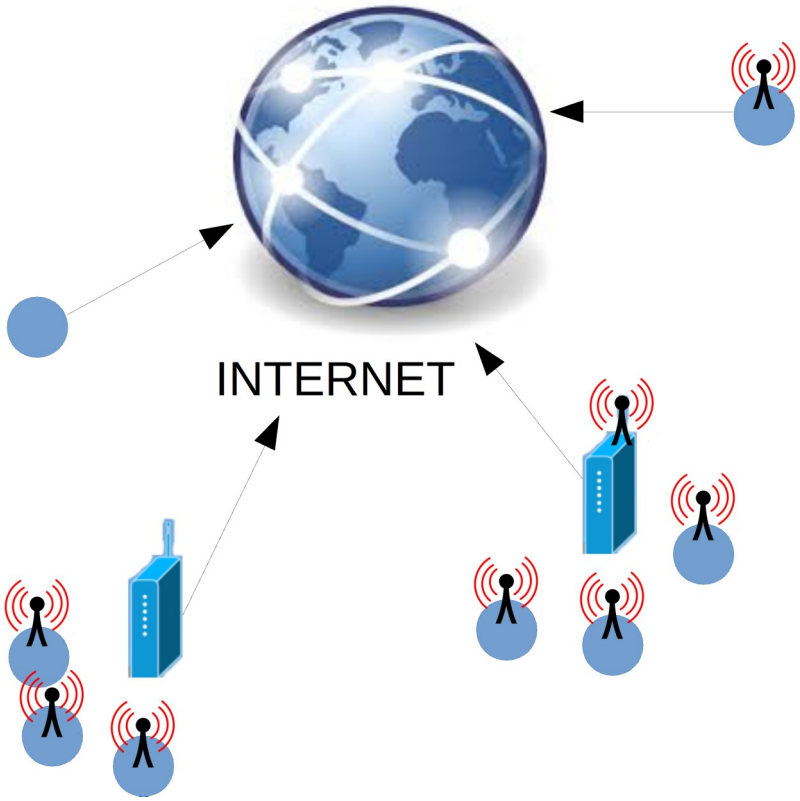
IEEE 802.3ab, 1000BASE-T.
IEEE 802.3ac, VLAN tagging.
IEEE 802.3ad, Link Aggregation.
IEEE 802.3ae, 10Gb/s Ethernet.
IEEE 802.3af, DTE power via MDI.
IEEE 802.3ak, 10GBASE-CX4.
IEEE 802.3z, Gigabit Ethernet.

IEE 802.11

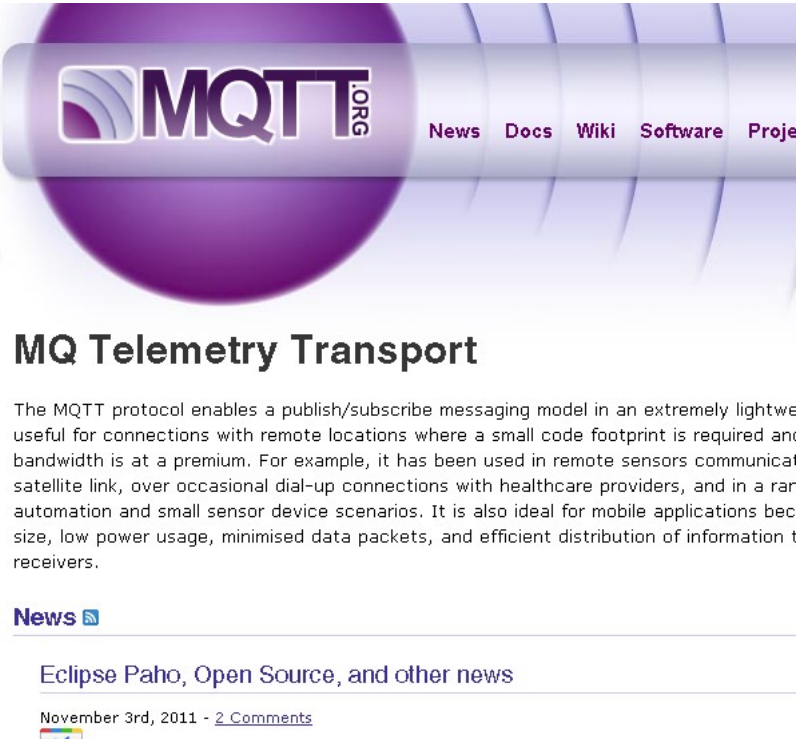
WIFI

Réseaux mobiles 4G, 5G
LORA, ...

Protocole
Réseau ?



Capteurs &
actionneurs



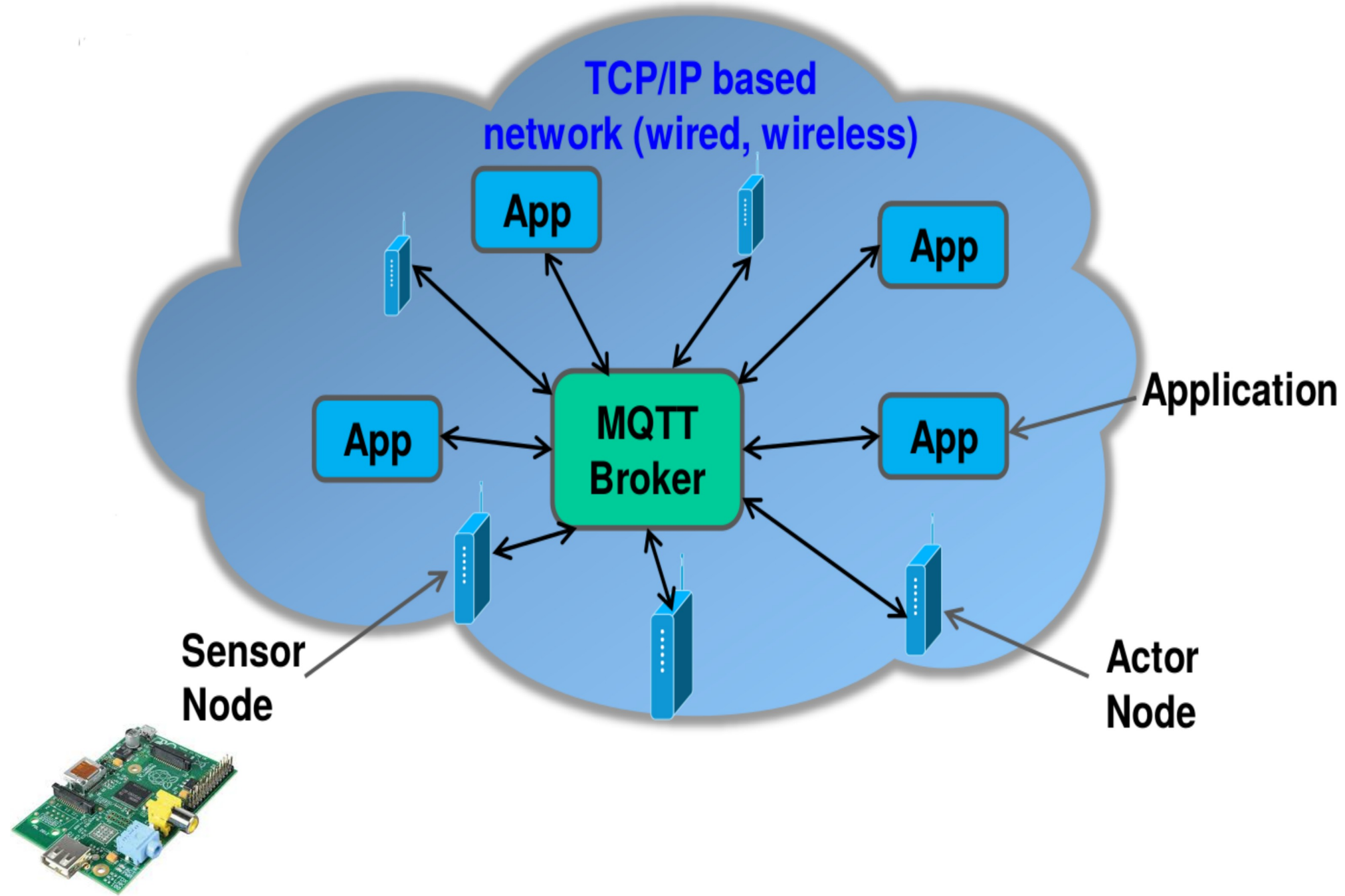
pourquoi MQTT ?
(MQ Telemetry Transport)

protocole de messagerie **léger** qui facilite la transmission des données de télémétrie pour les clients réseau dont les **ressources sont limitées**.

modèle de communication de type **publication/abonnement** via un broker de messages, est utilisé pour les échanges machine-à-machine (M2M).

Il joue un rôle important aujourd'hui dans l'Internet des objets.

A l'origine issu de la nécessité de transmettre les données de capteurs de pipelines situés dans le désert à des systèmes de supervision industrielle SCADA, il est basé sur une topologie réseau de type TCP/IP, et sur les **événements,** afin de limiter les coûts en énergie et en liaison satellite.


TCP/IP based
network (wired, wireless)
App
App
App
App
App
MQTT
Broker
Application
Sensor
Node
Actor
Node

Sécurité MQTT

Trois concepts sont fondamentaux pour la sécurité MQTT :

- identification,
- authentification et
- l'autorisation.

L'identification consiste à **nommer** le client auquel on donne des droits d'accès. **L'authentification** cherche à prouver l'identité du client, et **l'autorisation** consiste à gérer ses droits.

Mot de passe et/ou cryptographie

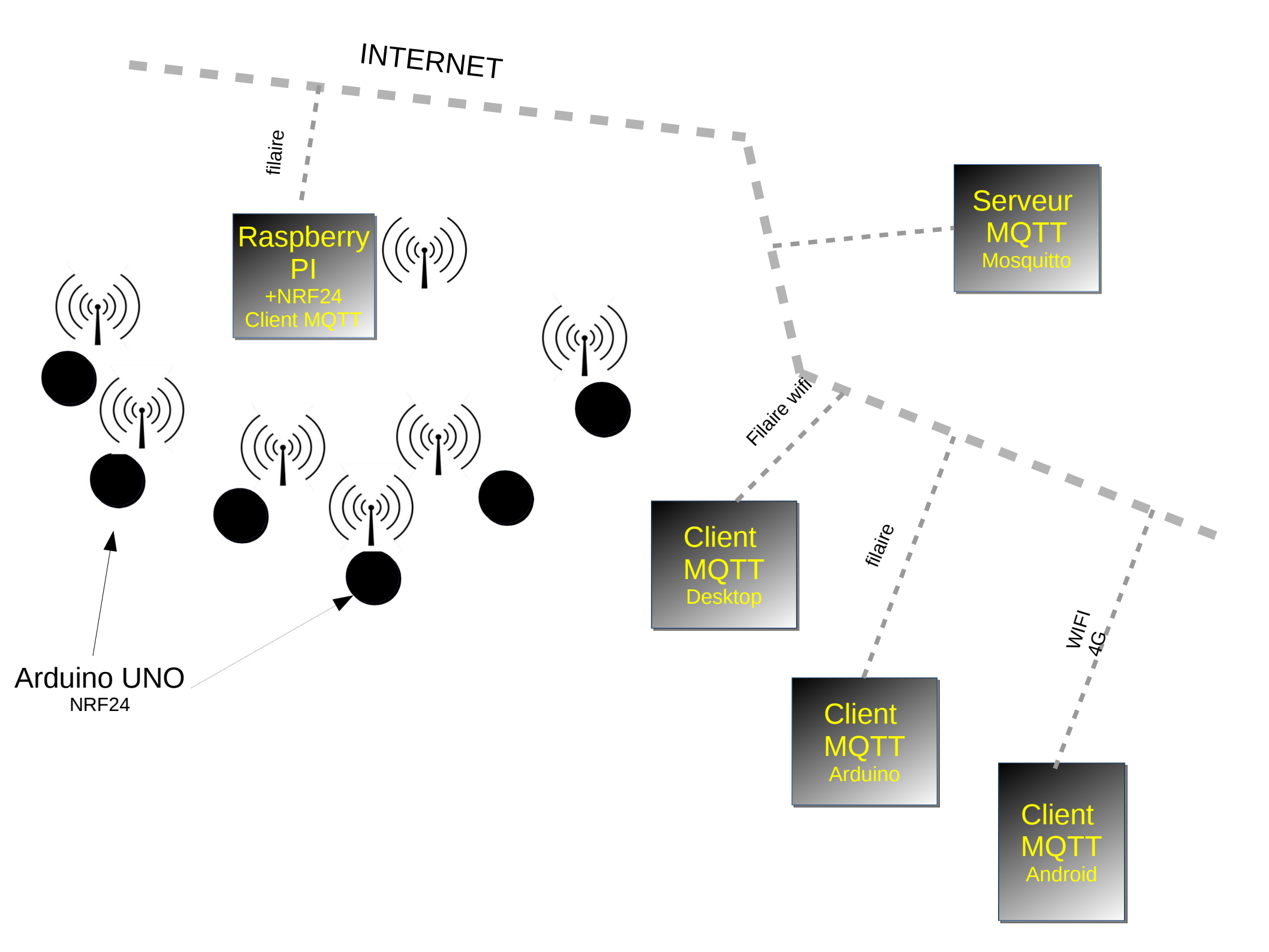
cryptographie asymétrique, ou cryptographie à clef publique:

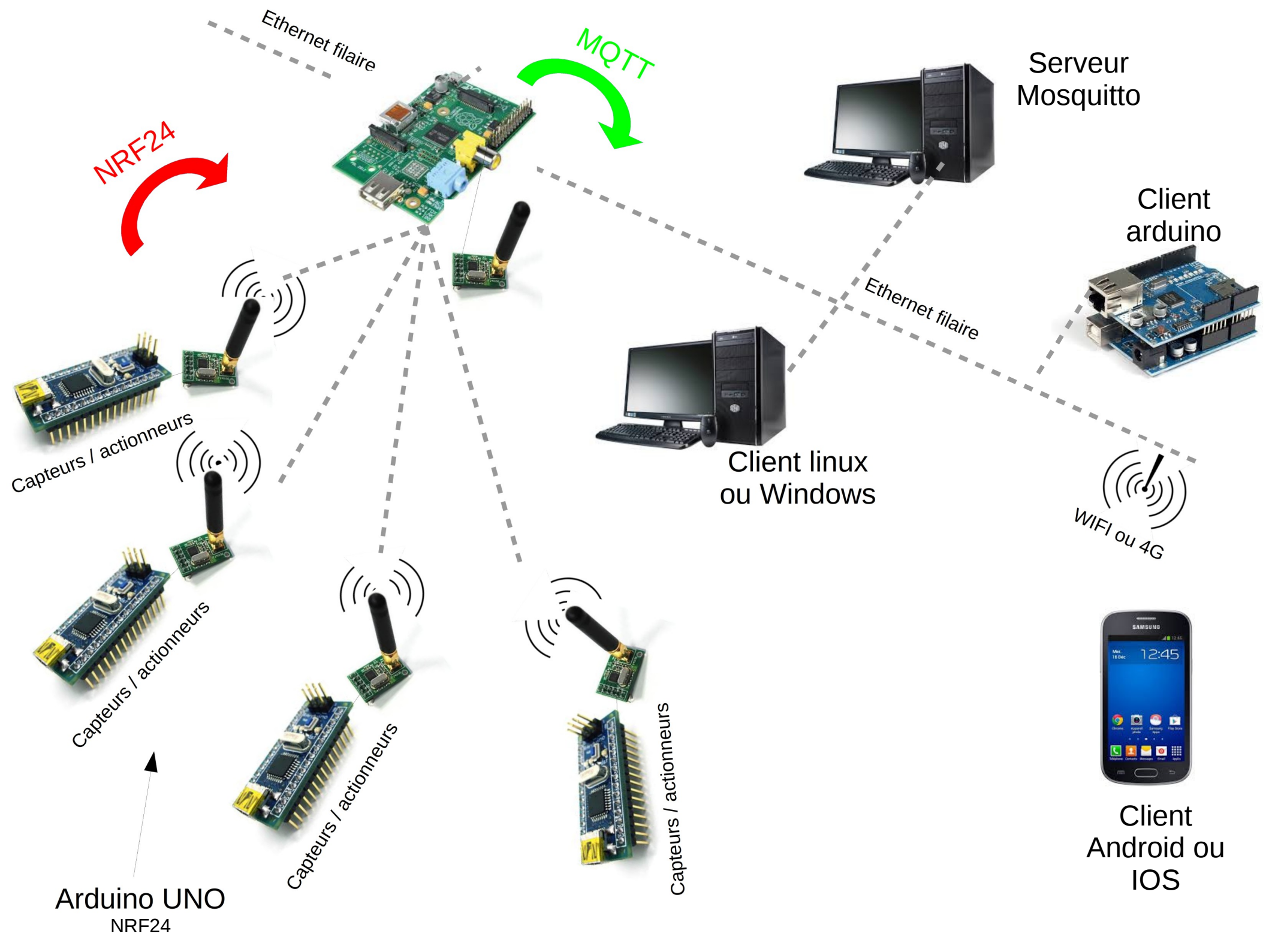
- fonctions à "sens unique", dite **clef publique**, à "brèche secrète", dite **clef privée.**
  - Une fois appliquées à un message, il est extrêmement difficile de retrouver le message original.
  - L'existence d'une "brèche secrète" permet cependant à la personne qui a conçu la fonction à sens unique de décoder le message grâce à la clef privée.



Ronald **R**ivest, Adi **S**hamir et Leonard **A**dleman.

Et nous dans tout ça ?


INTERNET
filaire
Raspberry
PI
+NRF24
Client MQTT
Serveur
MQTT
Mosquitto
Filaire wifi
Client
MQTT
Desktop
filaire
Client
MQTT
Arduino
WIFI
4G
Client
MQTT
Android
Arduino UNO
NRF24


Ethernet filaire
MQTT
NRF24
Serveur
Mosquitto
Client
arduino
Ethernet filaire
Client linux
ou Windows
WIFI ou 4G
Capteurs / actionneurs
Capteurs / actionneurs
Capteurs / actionneurs
Capteurs / actionneurs
Arduino UNO
NRF24
Client
Android ou
IOS

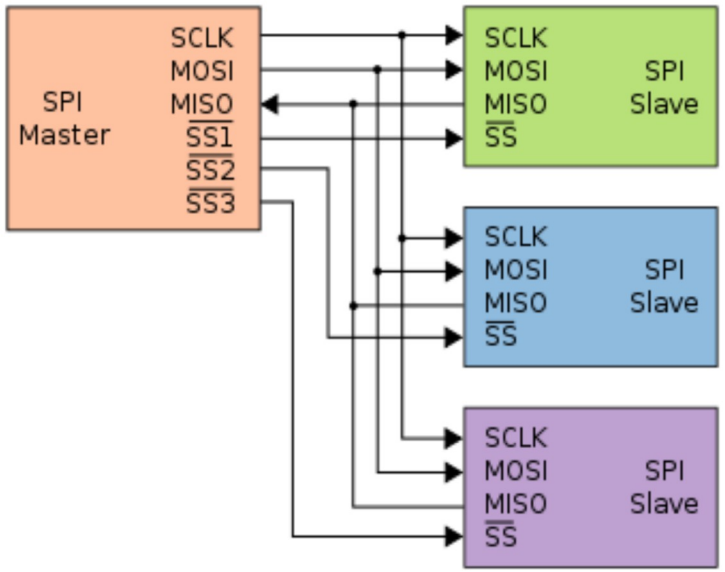
- Low cost, single-chip 2.4GHz GFSK RF transceiver IC
- Worldwide license-free 2.4GHz ISM band operation
- 250kbps, 1Mbps and 2Mbps on-air data-rate options
- Enhanced ShockBurst™ hardware protocol accelerator
- Ultra low power consumption – months to years of battery lifetime
- Drop-in compatible with the Nordic nRF24L01 previous generation Nordic RF transceiver IC
- Fully on-air compatible with all Nordic nRF24L Series, as well as nRF24E and nRF240 Series

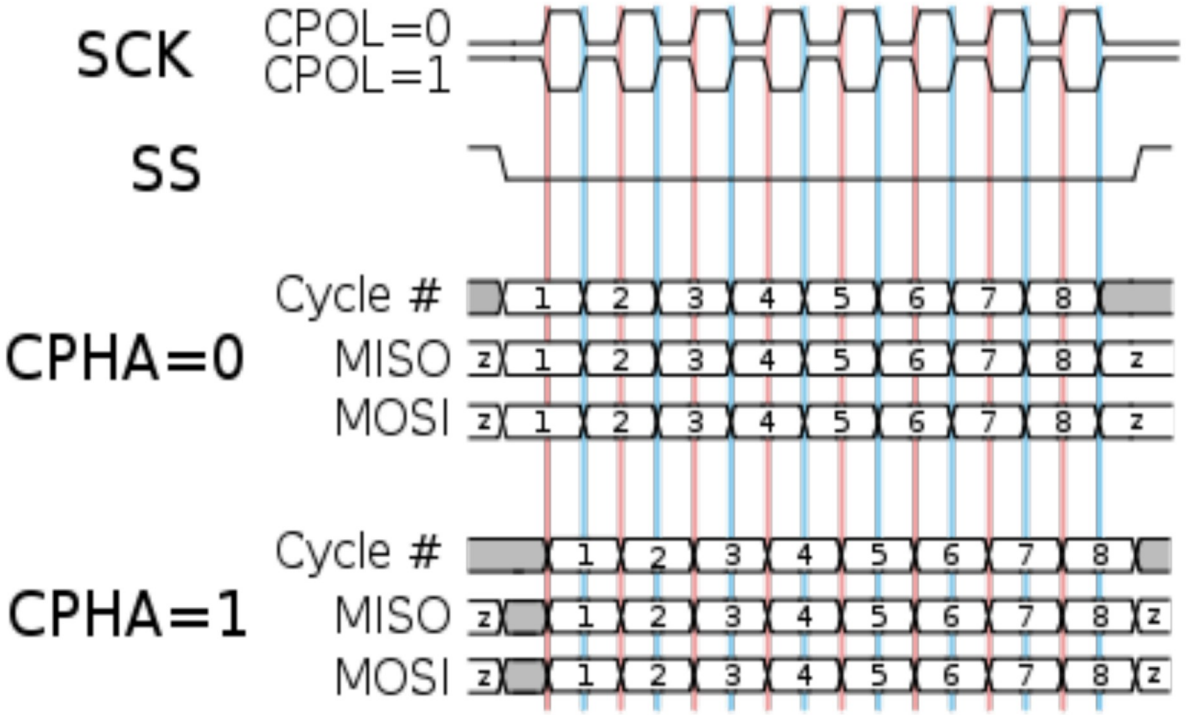
NORDIC SEMICONDUCTOR

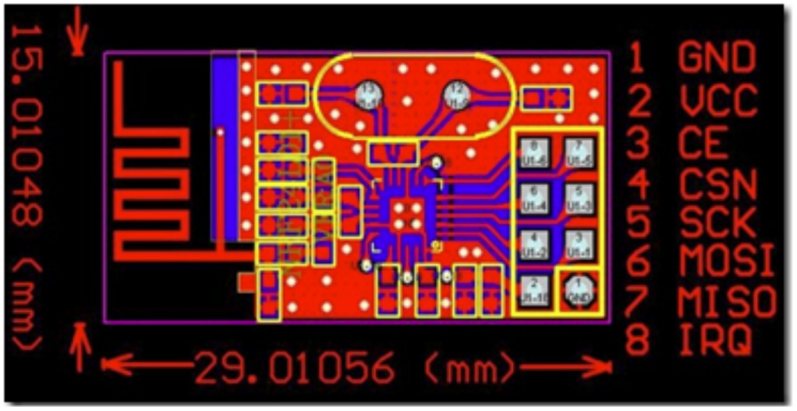
SPI (*Serial Peripheral Interface*)

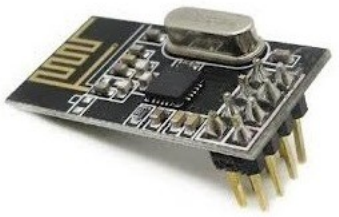
Arduino nano

NRF24









Power Down Mode

In power down mode nRF24L01 is disabled with minimal current consumption. In power down mode all the register values available from the SPI are maintained and the SPI can be activated.

Power down mode is entered by setting the PWR_UP bit in the CONFIG register low.

Standby Modes

By settting the PWR_UP bit in the CONFIG register to 1, the device enters standby-I mode. Standby-I mode is used to minimize average current consumption while maintaining short start up times. In this mode part of the crystal oscillator is active. This is the mode the nRF24L01 returns to from TX or RX mode when CE is set low.

RX mode

The RX mode is an active mode where the nRF24L01 radio is a receiver. To enter this mode, the nRF24L01 must have the PWR_UP bit set high, PRIM_RX bit set high and the CE pin set high.

In this mode the receiver demodulates the signals from the RF channel, constantly presenting the demodulated data to the baseband protocol engine. The baseband protocol engine constantly searches for a valid packet. If a valid packet is found (by a matching address and a valid CRC) the payload of the packet is presented in a vacant slot in the RX FIFO. If the RX FIFO is full, the received packet is discarded.

TX mode

The TX mode is an active mode where the nRF24L01 transmits a packet. To enter this mode, the nRF24L01 must have the PWR_UP bit set high, PRIM_RX bit set low, a payload in the TX FIFO and, a high pulse on the CE for more than 10µs.

The nRF24L01 stays in TX mode until it finishes transmitting a current packet.

RF channel frequency

The RF channel frequency determines the center of the channel used by the nRF24L01. The channel occupies a bandwidth of 1MHz at 1Mbps and 2MHz at 2Mbps. nRF24L01 can operate on frequencies from 2.400GHz to 2.525GHz. The resolution of the RF channel frequency setting is 1MHz.

At 2Mbps the channel occupies a bandwidth wider than the resolution of the RF channel frequency setting.

To ensure non-overlapping channels in 2Mbps mode, the channel spacing must be 2MHz or more. At 1Mbps the channel bandwidth is the same as the resolution of the RF frequency setting.

The RF channel frequency is set by the RF_CH register according to the following formula:

F0 = 2400 + RF_CH [MHz]

A transmitter and a receiver must be programmed with the same RF channel frequency to be able to communicate with each other.

PA control

The PA control is used to set the output power from the nRF24L01 power amplifier (PA). In TX mode PA control has four programmable steps

SPI RF-SETUP (RF_PWR)	RF output power	DC current consumption
11	0dBm	11.3mA
10	-6dBm	9.0mA
01	-12dBm	7.5mA
00	-18dBm	7.0mA

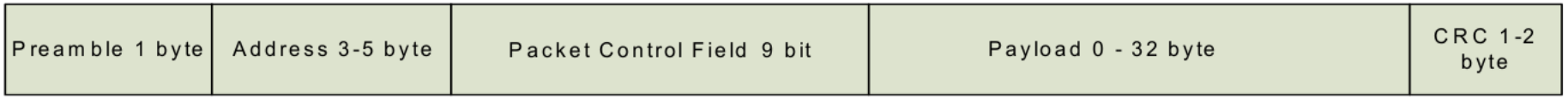
LNA gain

The gain in the Low Noise Amplifier (LNA) in the nRF24L01 receiver is controlled by the LNA gain setting.

The LNA gain makes it possible to reduce the current consumption in RX mode with 0.8mA at the cost of 1.5dB reduction in receiver sensitivity.

Enhanced Shockburst TM packet format

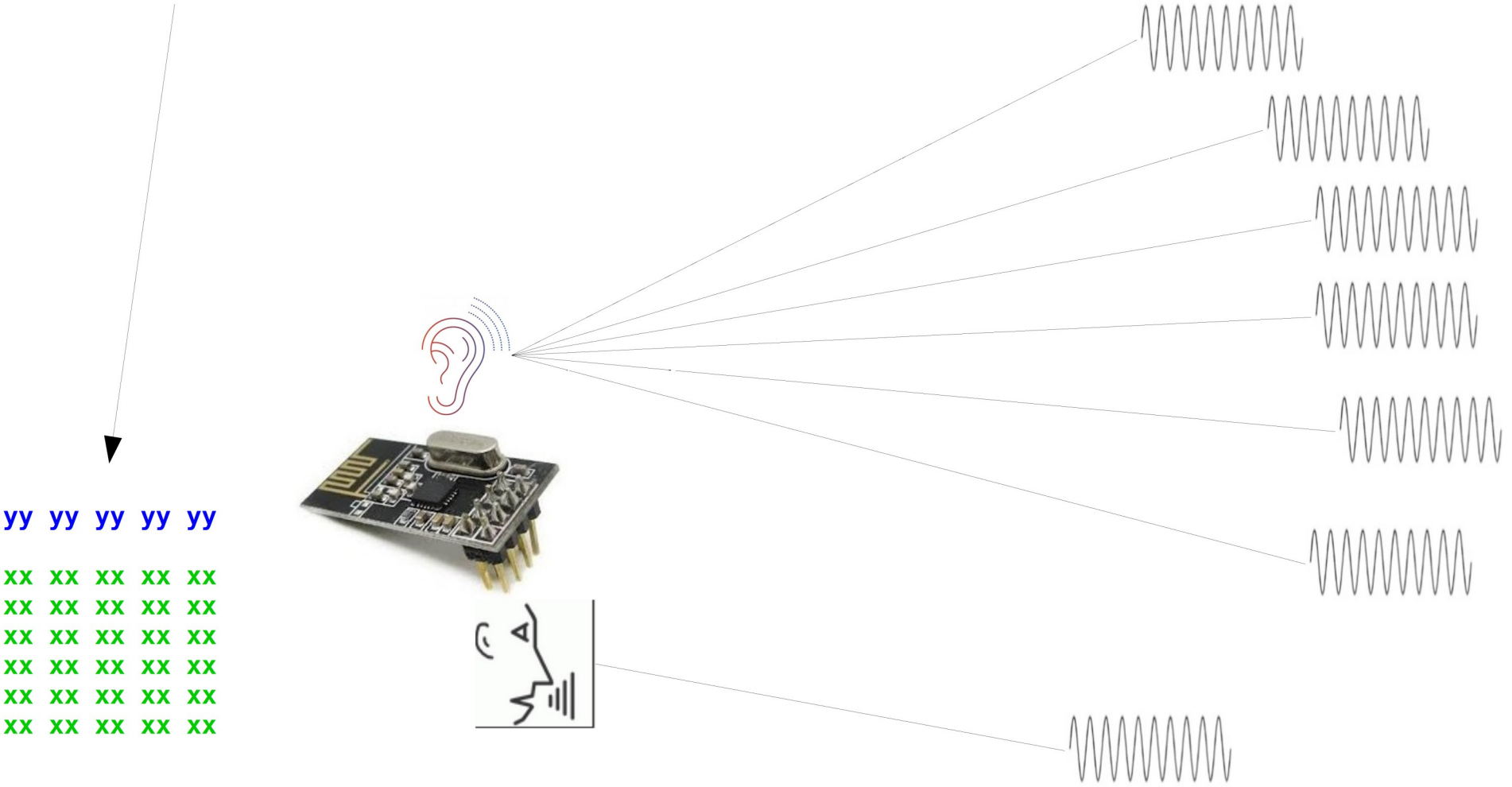
The format of the Enhanced ShockBurstTM packet is described in this chapter. The Enhanched ShockBurstTM packet contains a preamble field, address field, packet control field, payload field and a CRC field.



Payload length

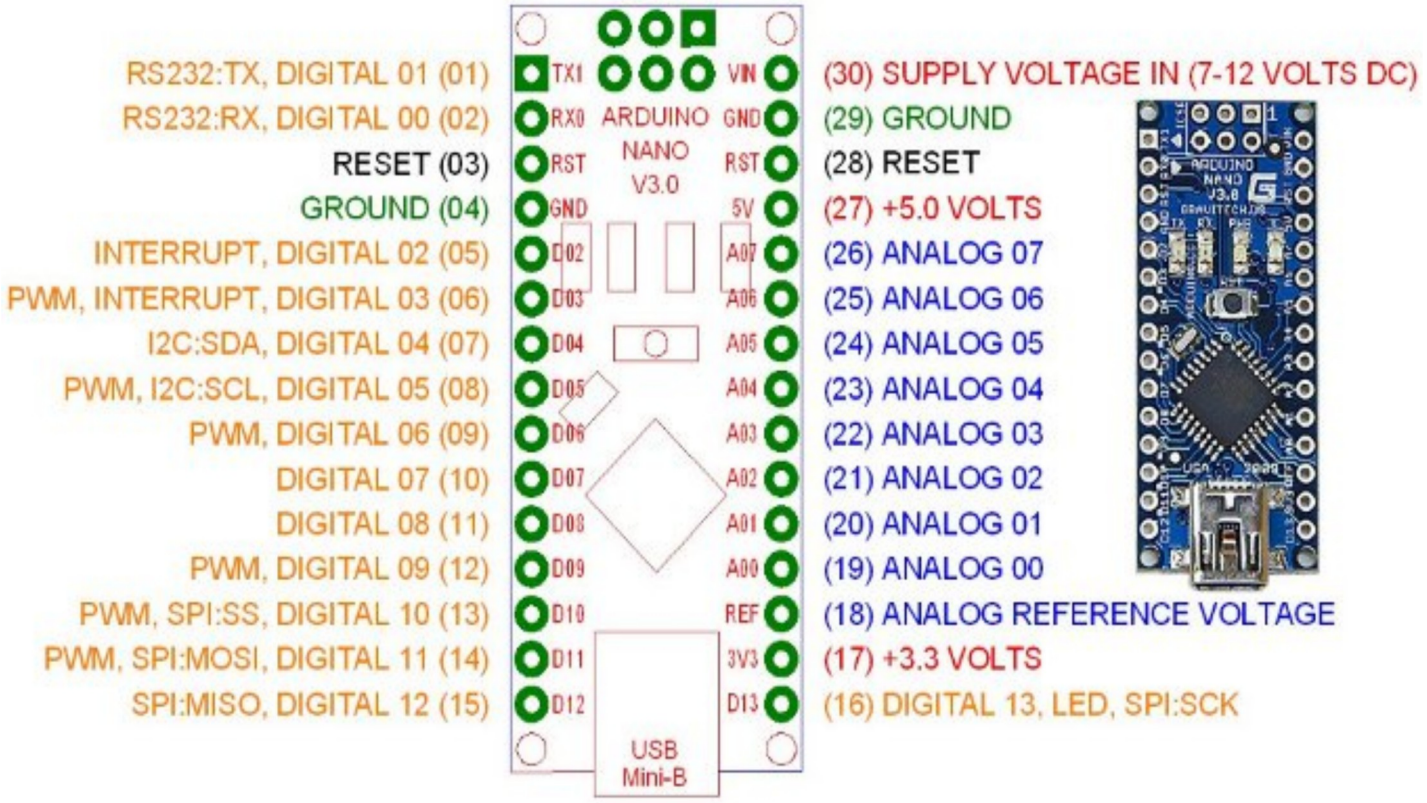
This 6 bit field specifies the length of the payload in bytes. The length of the payload can be from 0 to 32 bytes.

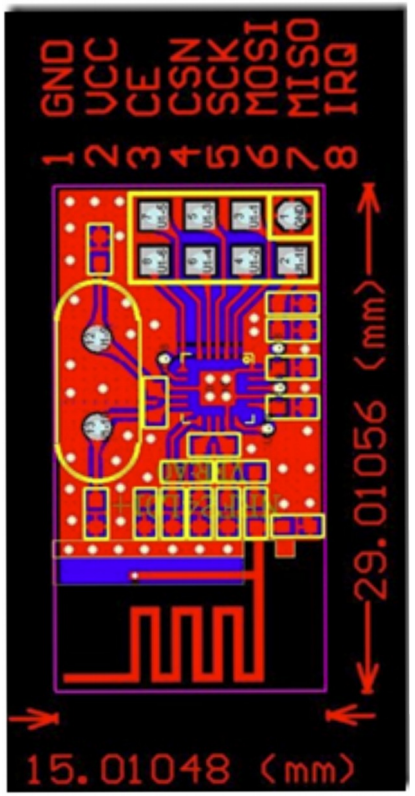
Canaux logiques - pipes



1 pipe d'émission : tx_pipe

6 pipes de réception : rx_pipe





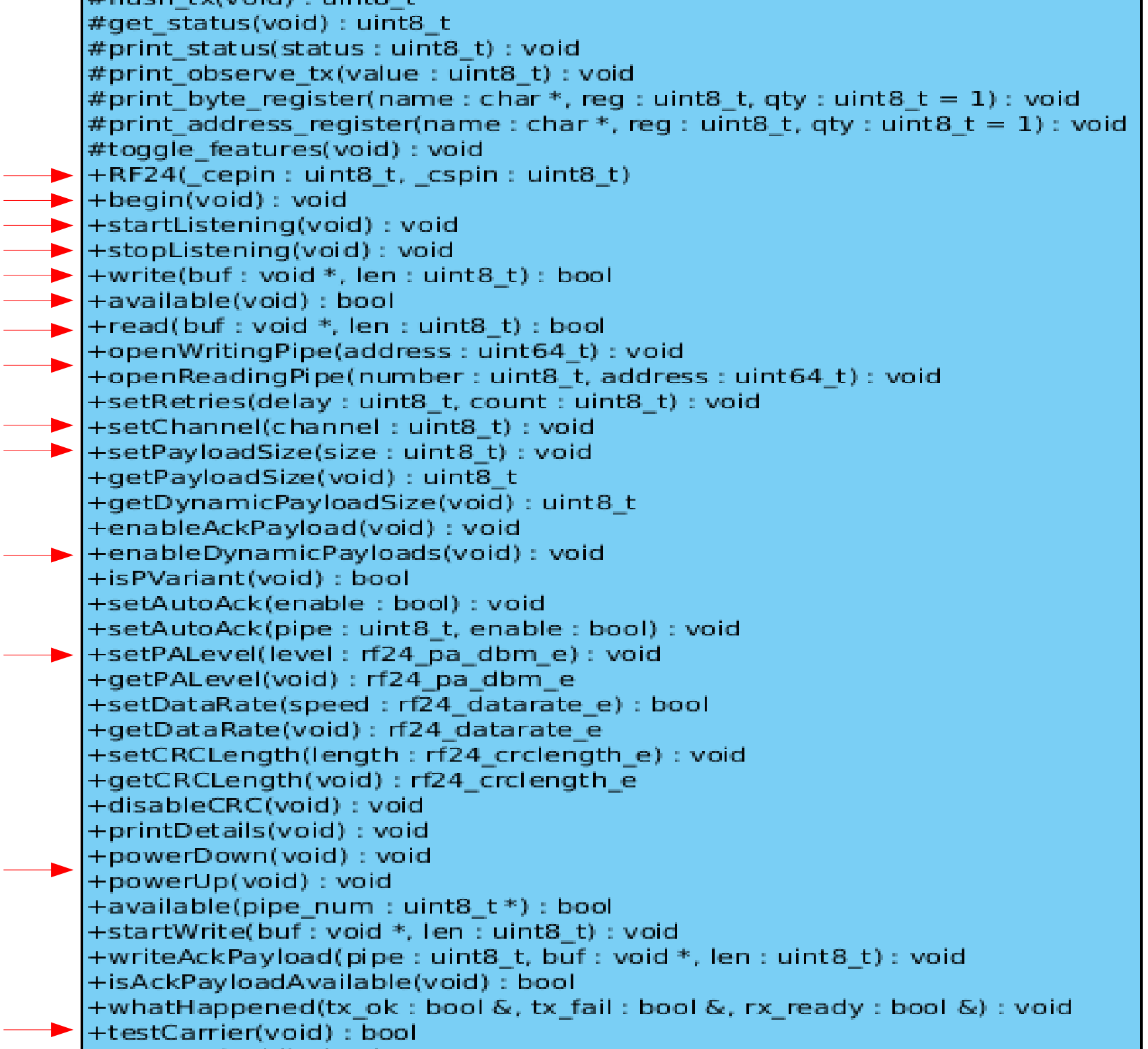
câblage des NRF24 sur Arduino Nano Broche Module	Broche Arduino
(1) GND	(4) GND
(2) VCC	(17) 3,3v
(3) CE	(9) D06
(4) CSN	(10) D07
(5) SCK	(16) SPI :SCK
(6) MOSI	(14) SPI:MOSI
(7) MISO	(15) SPI:MISO
(8) IRQ	

```
/*
 Copyright (C) 2011 J. Coliz <maniacbug@ymail.com>

 This program is free software; you can redistribute it and/or
 modify it under the terms of the GNU General Public License
 version 2 as published by the Free Software Foundation.
 */

class RF24
{
private:
  uint8_t ce_pin; /**< "Chip Enable" pin, activates the RX o
  uint8_t csn_pin; /**< SPI Chip select */
  bool wide_band; /* 2Mbs data rate in use? */
  bool p_variant; /* False for RF24L01 and true for RF24L01P
  uint8_t payload_size; /**< Fixed size of payloads */
  bool ack_payload_available; /**< Whether there is an ack p
  bool dynamic_payloads_enabled; /**< Whether dynamic payloa
  uint8_t ack_payload_length; /**< Dynamic size of pending a
  uint64_t pipe0_reading_address; /**< Last address set on p

protected:
  /**
   * @name Low-level internal interface.
   *
   *  Protected methods that address the chip directly.  Reg
   *  ever call these.  They are documented for completeness
   *  may want to extend this class.
   */
  /**@{*/

  /**
   * Set chip select pin
   *
   * Running SPI bus at PI_CLOCK_DIV2 so we don't waste time
   * and best of all, we make use of the radio's FIFO buffer
   * means we're less likely to effectively leverage our FIF
   * AVR runtime cost as toll.
   *
   * @param mode HIGH to take this unit off the SPI bus, LOW
   */
  void csn(int mode);

  /**
   * Set chip enable
   *
   * @param level HIGH to actively begin transmission or LOW
   * for a much more detailed description of this pin.
   */
  void ce(int level);

  . . .
```

RF24.h

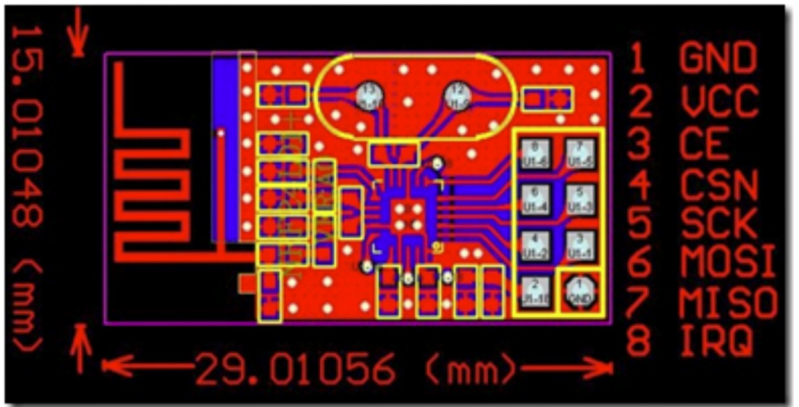
RF24.cpp

```
/*
 Copyright (C) 2011 J. Coliz <maniacbug@ymail.com>

 This program is free software; you can redistribute it and/or
 modify it under the terms of the GNU General Public License
 version 2 as published by the Free Software Foundation.
 */

#include "nRF24L01.h"
#include "RF24_config.h"
#include "RF24.h"

/****************************************************************************/

void RF24::csn(int mode)
{
  // Minimum ideal SPI bus speed is 2x data rate
  // If we assume 2Mbs data rate and 16Mhz clock, a
  // divider of 4 is the minimum we want.
  // CLK:BUS 8Mhz:2Mhz, 16Mhz:4Mhz, or 20Mhz:5Mhz
#ifdef ARDUINO
  SPI.setBitOrder(MSBFIRST);
  SPI.setDataMode(SPI_MODE0);
  SPI.setClockDivider(SPI_CLOCK_DIV4);
#endif
  digitalWrite(csn_pin,mode);
}

/****************************************************************************/

void RF24::ce(int level)
{
  digitalWrite(ce_pin,level);
}

/****************************************************************************/

uint8_t RF24::read_register(uint8_t reg, uint8_t* buf, uint8_t len)
{
  uint8_t status;

  csn(LOW);
  status = SPI.transfer( R_REGISTER | ( REGISTER_MASK & reg ) );
  while ( len-- )
    *buf++ = SPI.transfer(0xff);

. . .
```


#get_status(void) : uint8_t
#print_status(status : uint8_t) : void
#print_observe_tx(value : uint8_t) : void
#print_byte_register(name : char *, reg : uint8_t, qty : uint8_t = 1) : void
#print_address_register(name : char *, reg : uint8_t, qty : uint8_t = 1) : void
#toggle_features(void) : void
+RF24(_cepin : uint8_t, _cspin : uint8_t)
+begin(void) : void
+startListening(void) : void
+stopListening(void) : void
+write(buf : void *, len : uint8_t) : bool
+available(void) : bool
+read(buf : void *, len : uint8_t) : bool
+openWritingPipe(address : uint64_t) : void
+openReadingPipe(number : uint8_t, address : uint64_t) : void
+setRetries(delay : uint8_t, count : uint8_t) : void
+setChannel(channel : uint8_t) : void
+setPayloadSize(size : uint8_t) : void
+getPayloadSize(void) : uint8_t
+getDynamicPayloadSize(void) : uint8_t
+enableAckPayload(void) : void
+enableDynamicPayloads(void) : void
+isPVariant(void) : bool
+setAutoAck(enable : bool) : void
+setAutoAck(pipe : uint8_t, enable : bool) : void
+setPALevel(level : rf24_pa_dbm_e) : void
+getPALevel(void) : rf24_pa_dbm_e
+setDataRate(speed : rf24_datarate_e) : bool
+getDataRate(void) : rf24_datarate_e
+setCRCLength(length : rf24_crclength_e) : void
+getCRCLength(void) : rf24_crclength_e
+disableCRC(void) : void
+printDetails(void) : void
+powerDown(void) : void
+powerUp(void) : void
+available(pipe_num : uint8_t *) : bool
+startWrite(buf : void *, len : uint8_t) : void
+writeAckPayload(pipe : uint8_t, buf : void *, len : uint8_t) : void
+isAckPayloadAvailable(void) : bool
+whatHappened(tx_ok : bool &, tx_fail : bool &, rx_ready : bool &) : void
+testCarrier(void) : bool

Instanciation d'un objet RF24

```
// NRF24 communication
#include "nRF24L01.h"
#include "RF24.h"

#define RF24_CEPIN 6                    // pin ce du spi
#define RF24_CSPIN 7                    // pin cs du spi

RF24 radio(RF24_CEPIN, RF24_CSPIN);


char payload[32+1] = {0};
```



Initialisation (setup)

```
radio.begin();

//                                        Default radio settings
radio.enableDynamicPayloads();
radio.setDataRate(RF24_2MBPS);
radio.setPALevel(RF24_PA_MAX);
radio.setChannel(0x2A);

// delay 1ms, 3 retries
// 0 - 250us, 15 - 4000us
radio.setRetries(15,15);

radio.openWritingPipe(txPipe);
radio.openReadingPipe(1,rxPipe);

radio.setAutoAck(true);

radio.startListening();

radio.printDetails();
```

Canaux logiques - pipes

Each pipe can have up to 5 byte configurable address. Data pipe 0 has a unique 5 byte address. Data pipes 1-5 share the 4 most significant address bytes. The LSByte must be unique for all 6 pipes. Figure 11. is an example of how data pipes 0-5 are addressed.

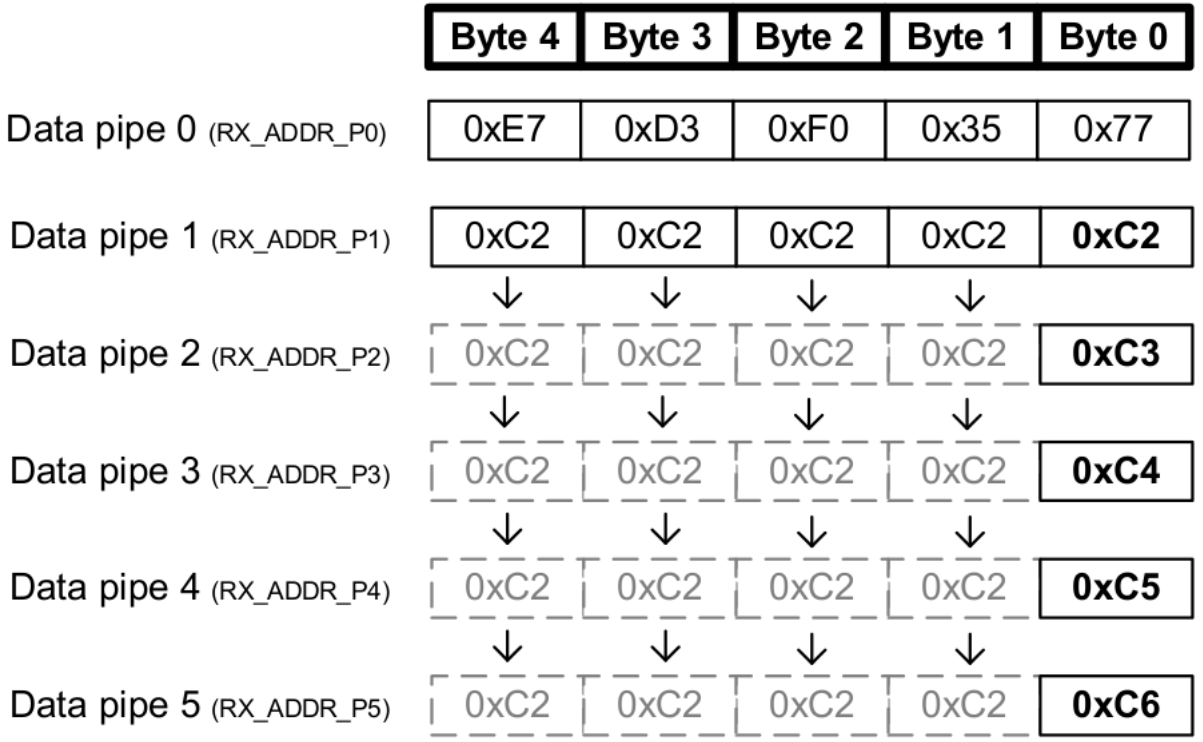
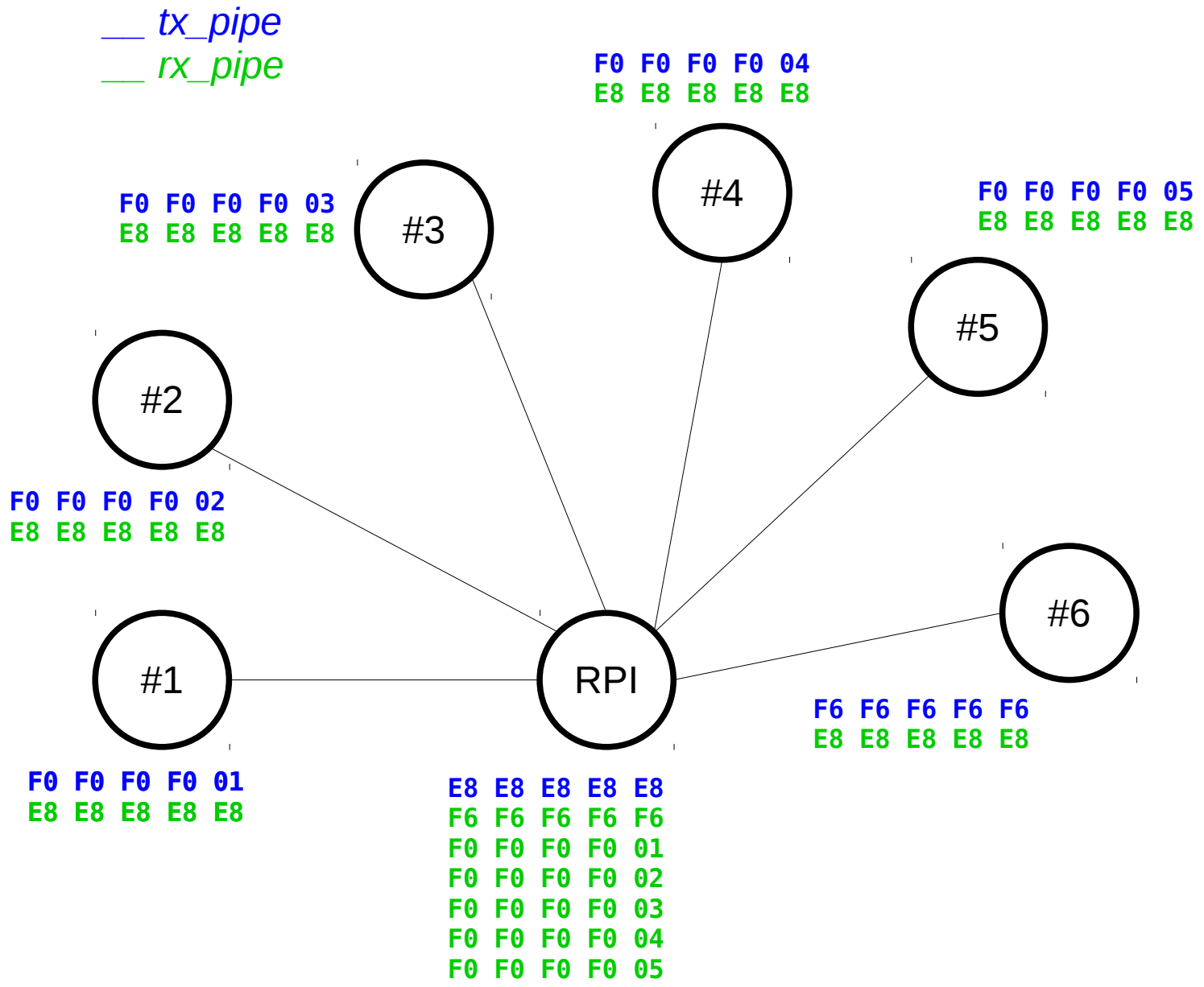


Figure 11. Addressing data pipes 0-5

Configuration des pipes



RPI = raspberry PI
#i = device n° i

Maximum : 6 *devices*

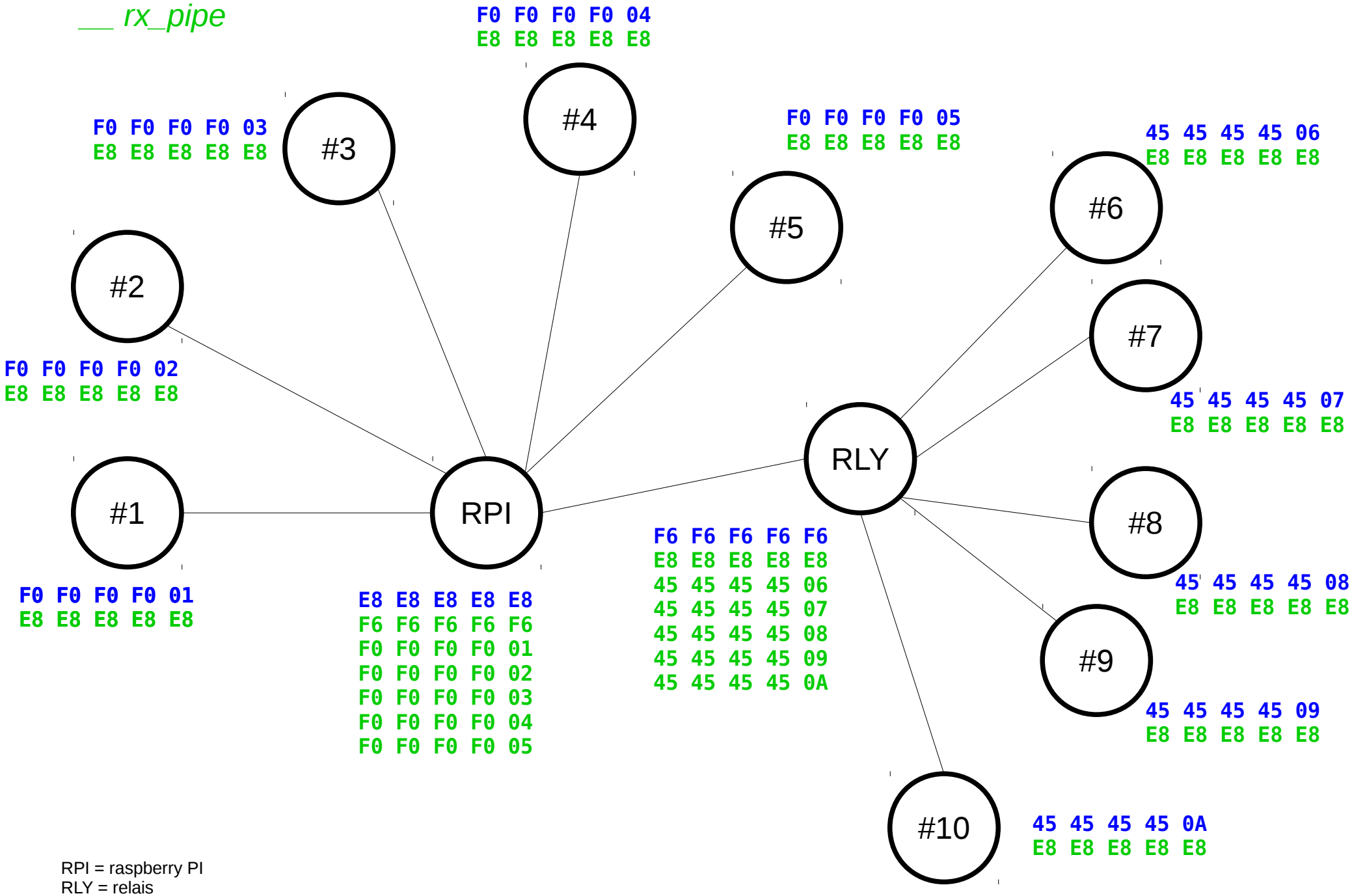
```
#define DEV_NUMBER 1

const uint64_t rxPipe = 0xE8E8E8E8E8LL;
const uint64_t txPipe = (DEV_NUMBER ==6 ? 0xF6F6F6F6F6LL :
                           0xF0F0F0F000LL | (DEV_NUMBER & 0xff));
```

Device Number	Rx pipe	Tx pipe
1	E8E8E8E8E8	F0F0F0F001
2	E8E8E8E8E8	F0F0F0F002
3	E8E8E8E8E8	F0F0F0F003
4	E8E8E8E8E8	F0F0F0F004
5	E8E8E8E8E8	F0F0F0F005
6	E8E8E8E8E8	F6F6F6F6F6

Au dessus de 6 devices, il faut mettre en oeuvre un relais

__ *tx_pipe*
__ *rx_pipe*

jusqu'à 10 devices



RPI = raspberry PI
RLY = relais
#i = device n° i

Relayé : maximum 10 *devices*

```
#define DEV_NUMBER 6


const uint64_t rxPipe = 0xE8E8E8E8E8LL;
const uint64_t txPipe = (DEV_NUMBER <6 ? 0xF0F0F0F000LL |
                        (DEV_NUMBER & 0xff) :
                        0x4545454500LL | (DEV_NUMBER & 0xff));
```

Au dessus de 10 devices, il faut mettre en oeuvre deux relais, .etc.

Petit programme de vérification

```
#include <iostream>
#include <inttypes.h>
using namespace std;

#define WITH_RELAY       // commenter la ligne suivante si <6 devices
#define DEV_NUMBER 6

int main(void){

#ifdef WITH_RELAY                          // > 6 devices
const uint64_t rxPipe = 0xE8E8E8E8E8LL;                  // rx from raspberry
const uint64_t txPipe = (DEV_NUMBER <6 ? 0xF0F0F0F000LL |  (DEV_NUMBER & 0xff) :
                                          0x4545454500LL | (DEV_NUMBER & 0xff));
#else                                      // <= 6 devices

const uint64_t rxPipe = 0xE8E8E8E8E8LL;                  // rx from raspberry
const uint64_t txPipe = (DEV_NUMBER ==6 ? 0xF6F6F6F6F6LL : 0xF0F0F0F000LL |
                                                            (DEV_NUMBER & 0xff));

#endif

cout<<"dev #:"<<DEV_NUMBER<<endl;
cout<<"rx:"<<hex<<rxPipe<<endl;
cout<<"tx:"<<hex<<txPipe<<endl;
return 0;
}
```

```
dev #:6
rx:e8e8e8e8e8
tx:4545454506
```

La boucle principale (loop)

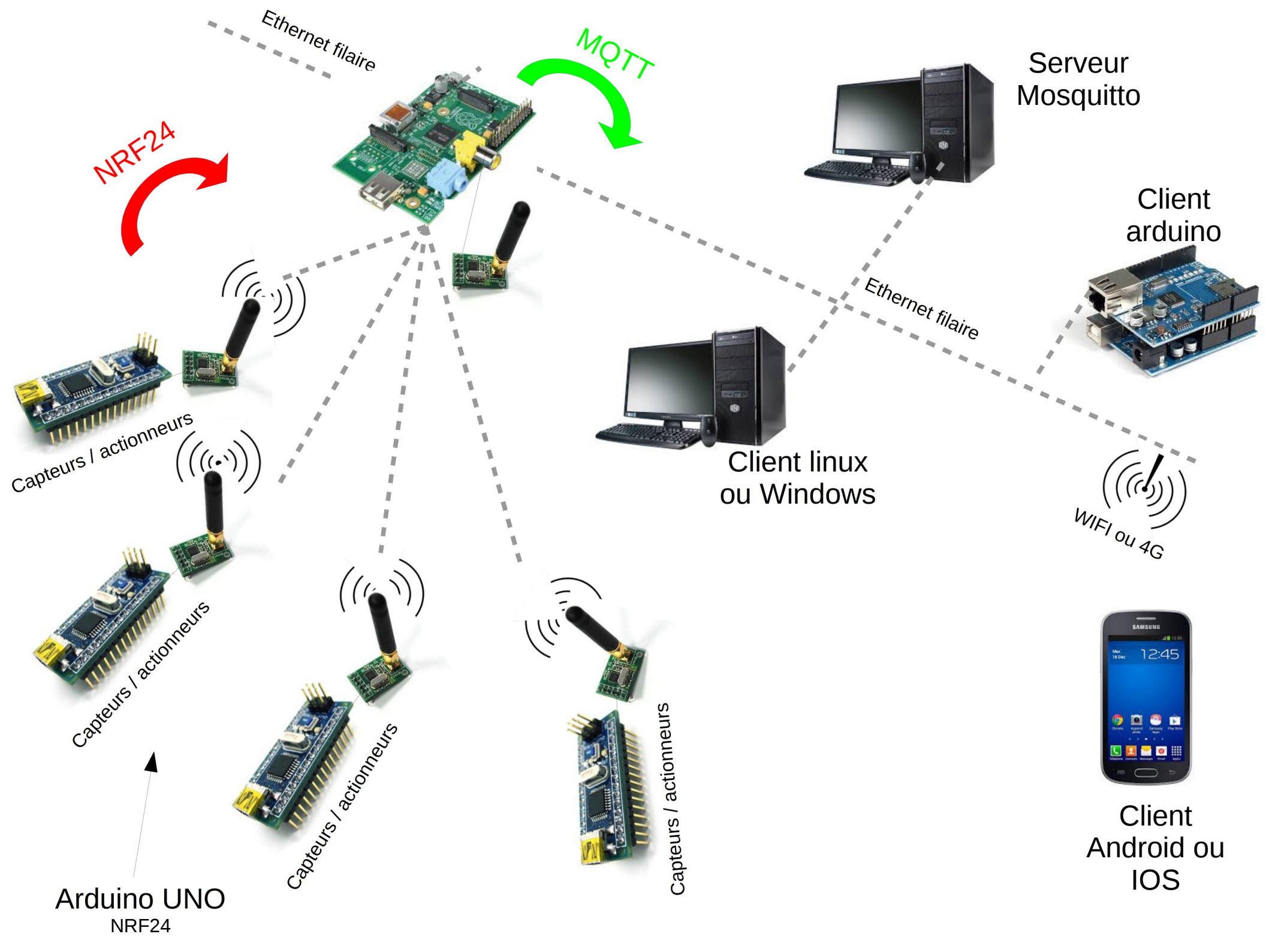
```
//                                    formation du payload
sprintf(payload,"...", ...);

// verify if channel is free
if (!radio.testCarrier()){
              // send payload
              radio.stopListening();
              radio.write((void *)payload, strlen(payload)+1);
              radio.startListening();
              }
  }
```

Dépend de l'application

```
//                                 quelque chose est il arrivé ?
if (radio.available()){
              int pls = radio.getDynamicPayloadSize();
              if (pls>=1 && pls<32){
                        radio.read((void*)payload, pls);
                        payload[pls]=0;
                        Serial.println(payload);
                        // traitement du payload
                        }
              }
```

Dépend de l'application

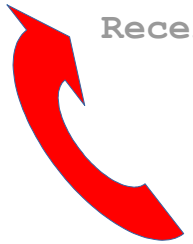
Transfert des données vers le réseau


Ethernet filaire
MQTT
NRF24
Serveur
Mosquitto
Client
arduino
Ethernet filaire
Client linux
ou Windows
WIFI ou 4G
Capteurs / actionneurs
Capteurs / actionneurs
Capteurs / actionneurs
Capteurs / actionneurs
Arduino UNO
NRF24
Client
Android ou
IOS

Protocole MQTT

MQTT

```
> sudo apt-get install mosquitto

> mosquitto -h
mosquitto is an MQTT v3.1 broker.

Usage: mosquitto [-c config_file] [-d] [-h] [-p port]

 -c : specify the broker config file.
 -d : put the broker into the background after starting.
 -h : display this help.
 -p : start the broker listening on the specified port.
      Not recommended in conjunction with the -c option.

See http://mosquitto.org/ for more information.

> mosquitto_sub -d -h localhost -t sensors/light
Received CONNACK
Received SUBACK
Subscribed (mid: 1): 0
 . . . etc . . .

Received PUBLISH (d0, q0, r0, m0, 'sensors/light', ... (7 bytes))
100 Lux
Sending PINGREQ
Received PINGRESP
```



```
> mosquitto_pub -d -t sensors/light -m "100 Lux" -h localhost
Received CONNACK
Sending PUBLISH (d0, q0, r0, m1, 'sensors/light', ... (7 bytes))
```

```
mosquitto_sub -d -h localhost -t sensors/light
Received CONNACK
Received SUBACK
Subscribed (mid: 1): 0
Sending PINGREQ
Received PINGRESP
Sending PINGREQ
```

Capturing from Loopback: lo [Wireshark 1.10.6 (v1.10.6 from master-1.10)]

File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help

Filter: tcp.port == 1883 Expression... Clear Apply Enregistrer

802.11 Channel: Channel Offset: FCS Filter: All Frames None

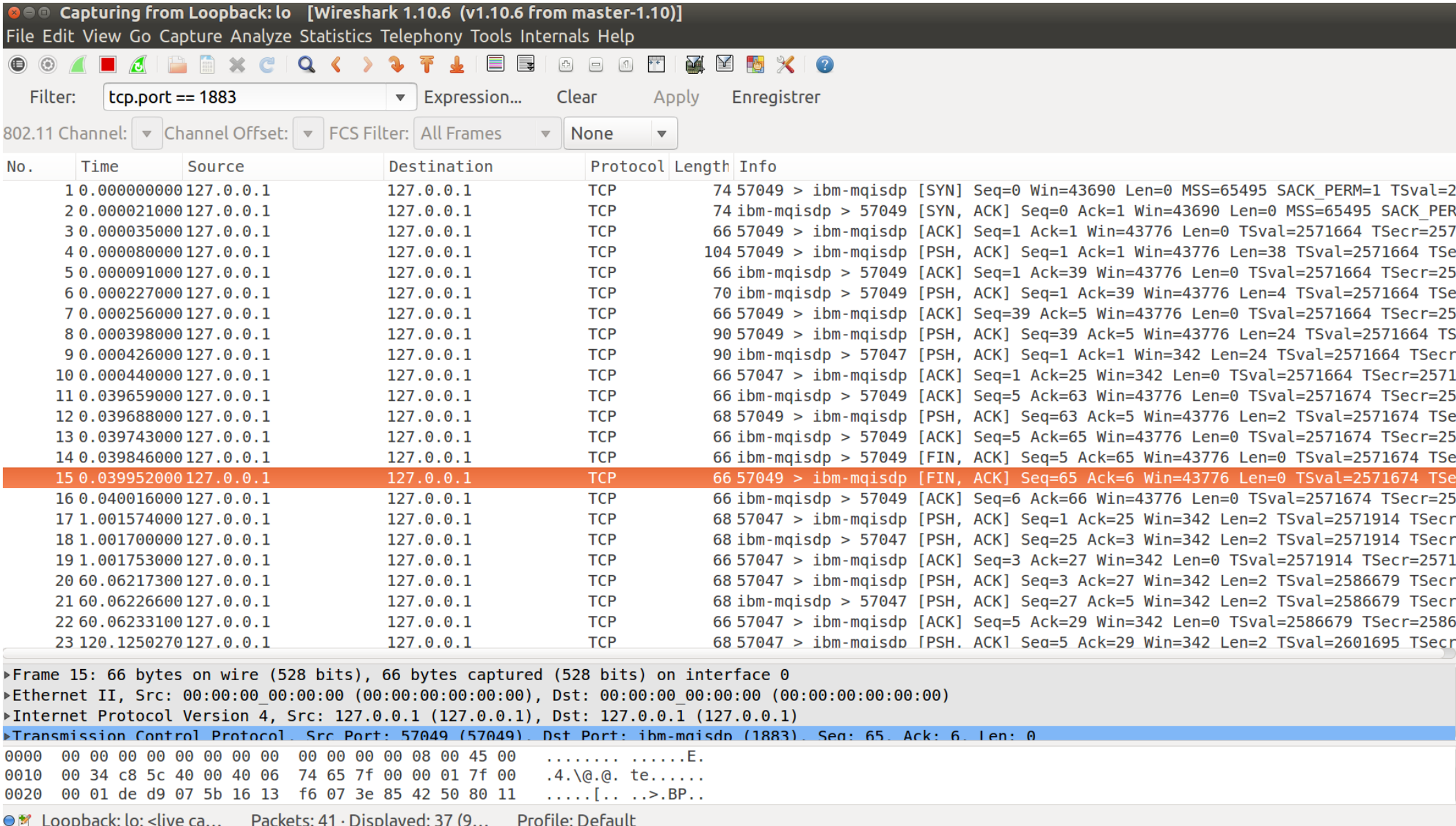
No.	Time	Source	Destination	Protocol	Length	Info
35	6.981003000	127.0.0.1	127.0.0.1	TCP	74	56796 > ibm-mqisdp [SYN] Seq=0 Win=43690 Len=0 MSS=65495 SACK_PERM=1 TSval=2
36	6.981034000	127.0.0.1	127.0.0.1	TCP	74	ibm-mqisdp > 56796 [SYN, ACK] Seq=0 Ack=1 Win=43690 Len=0 MSS=65495 SACK_PER
37	6.981062000	127.0.0.1	127.0.0.1	TCP	66	56796 > ibm-mqisdp [ACK] Seq=1 Ack=1 Win=43776 Len=0 TSval=2204263 TSecr=220
38	6.981146000	127.0.0.1	127.0.0.1	TCP	104	56796 > ibm-mqisdp [PSH, ACK] Seq=1 Ack=1 Win=43776 Len=38 TSval=2204263 TSe
39	6.981176000	127.0.0.1	127.0.0.1	TCP	66	ibm-mqisdp > 56796 [ACK] Seq=1 Ack=39 Win=43776 Len=0 TSval=2204263 TSecr=22
40	6.981905000	127.0.0.1	127.0.0.1	TCP	70	ibm-mqisdp > 56796 [PSH, ACK] Seq=1 Ack=39 Win=43776 Len=4 TSval=2204263 TSe
41	6.981935000	127.0.0.1	127.0.0.1	TCP	66	56796 > ibm-mqisdp [ACK] Seq=39 Ack=5 Win=43776 Len=0 TSval=2204263 TSecr=22
42	6.982100000	127.0.0.1	127.0.0.1	TCP	86	56796 > ibm-mqisdp [PSH, ACK] Seq=39 Ack=5 Win=43776 Len=20 TSval=2204263 TS
43	6.982212000	127.0.0.1	127.0.0.1	TCP	71	ibm-mqisdp > 56796 [PSH, ACK] Seq=5 Ack=59 Win=43776 Len=5 TSval=2204263 TSe
44	7.018936000	127.0.0.1	127.0.0.1	TCP	66	56796 > ibm-mqisdp [ACK] Seq=59 Ack=10 Win=43776 Len=0 TSval=2204273 TSecr=2
45	67.04332400	127.0.0.1	127.0.0.1	TCP	68	56796 > ibm-mqisdp [PSH, ACK] Seq=59 Ack=10 Win=43776 Len=2 TSval=2219279 TS
46	67.04357500	127.0.0.1	127.0.0.1	TCP	68	ibm-mqisdp > 56796 [PSH, ACK] Seq=10 Ack=61 Win=43776 Len=2 TSval=2219279 TS
47	67.04369700	127.0.0.1	127.0.0.1	TCP	66	56796 > ibm-mqisdp [ACK] Seq=61 Ack=12 Win=43776 Len=0 TSval=2219279 TSecr=2
48	127.1048070	127.0.0.1	127.0.0.1	TCP	68	56796 > ibm-mqisdp [PSH, ACK] Seq=61 Ack=12 Win=43776 Len=2 TSval=2234294 TS
49	127.1048730	127.0.0.1	127.0.0.1	TCP	68	ibm-mqisdp > 56796 [PSH, ACK] Seq=12 Ack=63 Win=43776 Len=2 TSval=2234294 TS
50	127.1049040	127.0.0.1	127.0.0.1	TCP	66	56796 > ibm-mqisdp [ACK] Seq=63 Ack=14 Win=43776 Len=0 TSval=2234294 TSecr=2
57	187.1678830	127.0.0.1	127.0.0.1	TCP	68	56796 > ibm-mqisdp [PSH, ACK] Seq=63 Ack=14 Win=43776 Len=2 TSval=2249310 TS
58	187.1680680	127.0.0.1	127.0.0.1	TCP	68	ibm-mqisdp > 56796 [PSH, ACK] Seq=14 Ack=65 Win=43776 Len=2 TSval=2249310 TS
59	187.1681570	127.0.0.1	127.0.0.1	TCP	66	56796 > ibm-mqisdp [ACK] Seq=65 Ack=16 Win=43776 Len=0 TSval=2249310 TSecr=2
70	247.2305650	127.0.0.1	127.0.0.1	TCP	68	56796 > ibm-mqisdp [PSH, ACK] Seq=65 Ack=16 Win=43776 Len=2 TSval=2264325 TS
71	247.2306520	127.0.0.1	127.0.0.1	TCP	68	ibm-mqisdp > 56796 [PSH, ACK] Seq=16 Ack=67 Win=43776 Len=2 TSval=2264325 TS

```
▸Frame 35: 74 bytes on wire (592 bits), 74 bytes captured (592 bits) on interface 0
▸Ethernet II, Src: 00:00:00_00:00:00 (00:00:00:00:00:00), Dst: 00:00:00_00:00:00 (00:00:00:00:00:00)
▸Internet Protocol Version 4, Src: 127.0.0.1 (127.0.0.1), Dst: 127.0.0.1 (127.0.0.1)
▾Transmission Control Protocol, Src Port: 56796 (56796), Dst Port: ibm-mqisdp (1883), Seq: 0, Len: 0
  Source port: 56796 (56796)
  Destination port: ibm-mqisdp (1883)
```

```
0000  00 00 00 00 00 00 00 00  00 00 00 00 08 00 45 00   ........ ......E.
0010  00 3c b8 9d 40 00 40 06  84 1c 7f 00 00 01 7f 00   .<..@.@. ........
0020  00 01 dd dc 07 5b 93 a1  1a fa 00 00 00 00 a0 02   .....[.. ........
```

Loopback: lo: <live ca... Packets: 616 · Displayed: 52 (... Profile: Default

```
mosquitto_pub -d -t sensors/light -m "100 Lux" -h localhost
Received CONNACK
Sending PUBLISH (d0, q0, r0, m1, 'sensors/light', ... (7 bytes))
```



```
mosquitto_sub -d -h localhost -t sensors/light
Received CONNACK
Received SUBACK
Subscribed (mid: 1): 0
Sending PINGREQ
Received PINGRESP
Sending PINGREQ
Received PINGRESP
Received PUBLISH (d0, q0, r0, m0, 'sensors/light', ...
(7 bytes))
100 Lux
Sending PINGREQ
Received PINGRESP
Sending PINGREQ
Received PINGRESP
Sending PINGREQ
Received PINGRESP
Sending PINGREQ
Received PINGRESP
```

Client en langage c (geany)

```
/*
 * client-simple.cpp
 *
 * Copyright 2015 Michel GRIMALDI <grimaldi@grimaldi-UX31A>
 *
 * */

#include <stdio.h>
#include <inttypes.h>
#include <mosquitto.h>
#include <string.h>
#include <iostream>
using namespace std;
```

Arguments de compilation :

g++ -Wall -o "%e" "%f" -lm **-I/usr/include -L/usr/lib -lmosquitto**

Dépendance : libmosquitto

sudo apt-get install libmosquitto0 libmosquitto0.dev

Documentation de l'API:

http://mosquitto.org/api/files/mosquitto-h.html

Client en langage c - callback

```
//    fonction appelée à l'arrivée de mesg MQTT
void my_message_callback(void *mosq,  const struct mosquitto_message *message)
{
    if(message->payloadlen){
        cout<<message->topic<<'='<<message->payload<<endl;
    }else{
        cout<<message->topic<<'='<<"null"<<endl;
    }
}
```

```
//  fonction appelée lors de la connexion à un serveur MQTT
void my_connect_callback(void *mosq, int result)
{
    int i;
    if(!result){
        /* Subscribe to broker information topics on successful connect. */
        //mosquitto_subscribe((mosquitto *)mosq, NULL, "$SYS/#", 2);
    }else{
        cout<<"Connect failed"<<endl;
    }
}
```

```
//  fonction appelée lors d'une nouvelle souscription
void my_subscribe_callback(void *mosq, uint16_t mid, int qos_count, const uint8_t *granted_qos)
{
    int i;

    cout<<"Subscribed (mid: "<<mid<<"d): "<<granted_qos[0]<<endl;
    for(i=1; i<qos_count; i++){
        cout<<", "<<granted_qos[i];
    }
    cout<<endl;
}
```

Client en langage c - main

```
int main(int argc, char *argv[])
{

const char *host = "192.168.1.99";
int port = 1883;
int keepalive = 60;
bool clean_session = true;
void *obj = NULL;
struct mosquitto *mosq = NULL;
int major, minor, revision;

// version de la bibliothèque libmosquitto
mosquitto_lib_version(&major, &minor, &revision);
cout<<"lib found:"<<major<<'.'<<minor<<'.'<<revision<<endl;

// initialisation/allocation lib MQTT
mosquitto_lib_init();
mosq = mosquitto_new("grimaldi", obj);
if(!mosq){
     cout<<"Error: Out of memory"<<endl;
     return 1;
     }

// mise en place des callBack functions
mosquitto_message_callback_set(mosq, my_message_callback);
mosquitto_subscribe_callback_set(mosq, my_subscribe_callback);

//  connexion au serveur mosquitto
if(mosquitto_connect(mosq, host, port, keepalive, clean_session)){
     cout<<"Unable to connect"<<endl;
     return 1;
     }
```

Client en langage c – main

```
// deux souscriptions à de topics QoS=1
uint16_t mid;
mosquitto_subscribe(mosq, &mid, "sensors/toto", 1);
mosquitto_subscribe(mosq, &mid, "sensors/tata", 1);

// boucle d'attente
int i=0;
while(!mosquitto_loop(mosq, 10)){     // super boucle, sans fin pour l'exemple

    // une publication dans la boucle pour tester
    if (!(i%100)){
            char payload[100];      // les données à envoyer
            sprintf(payload,"mesg n°%d", i);
            mosquitto_publish(mosq, &mid, "sensors/tata", strlen(payload),
                              (uint8_t*)payload, 0, true );
            }
    i++;
    } // super boucle

//  pour le cas où la boucle s'arrêterait
mosquitto_destroy(mosq);
mosquitto_lib_cleanup();
return 0;
}
```

Le Topic $SYS

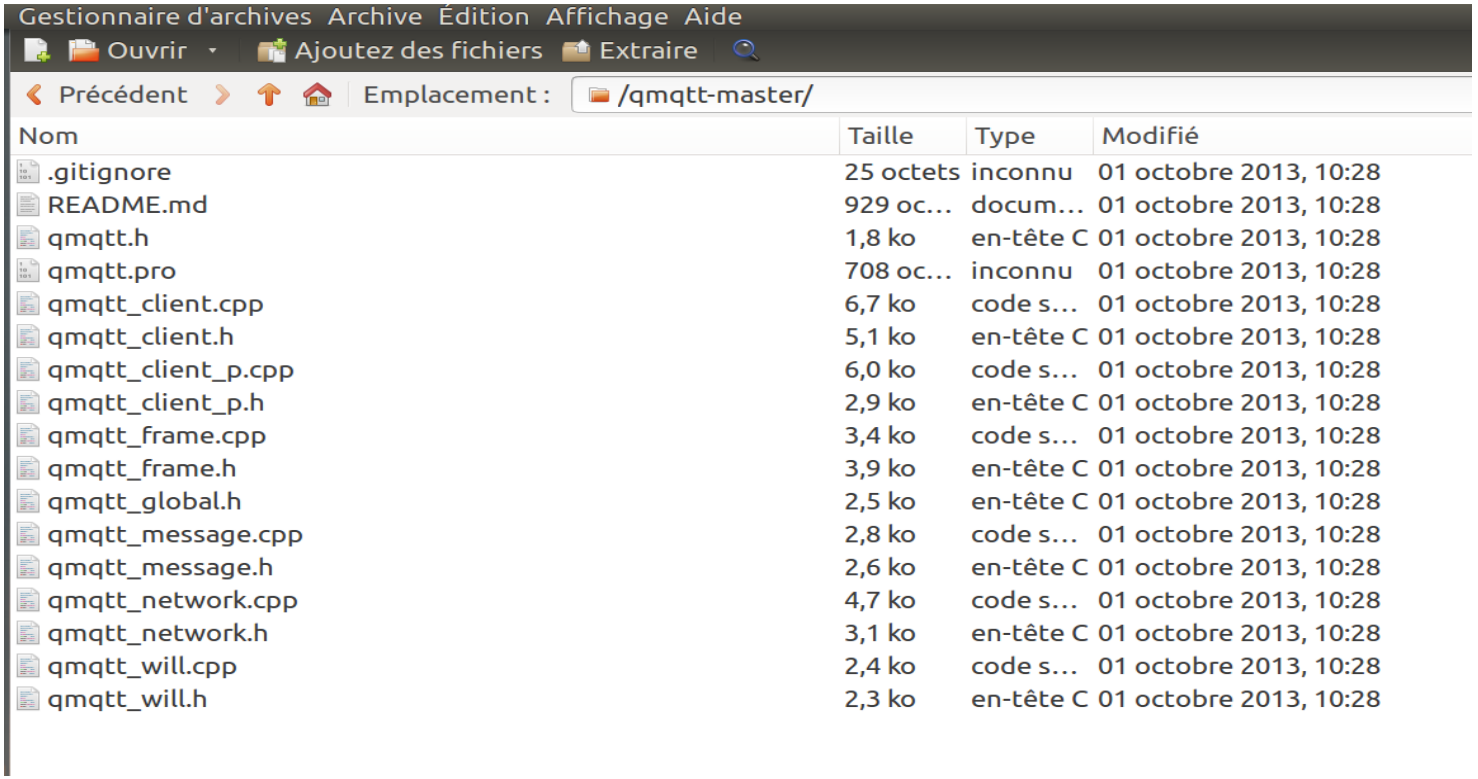
```
grimaldi@grimaldi-UX31A:~$ mosquitto_sub -v -t \$SYS/#

$SYS/broker/version mosquitto version 0.15
$SYS/broker/timestamp 2013-08-23 19:23:41+0000
$SYS/broker/changeset $Revision: e745e1ab5007 $
$SYS/broker/uptime 2376 seconds
$SYS/broker/messages/stored 20
$SYS/broker/messages/received 52
$SYS/broker/messages/sent 206
$SYS/broker/messages/per second/received 0
$SYS/broker/messages/per second/sent 0
$SYS/broker/clients/total 0
$SYS/broker/clients/inactive 0
$SYS/broker/clients/active 0
$SYS/broker/clients/maximum 2
$SYS/broker/heap/current size 3380 bytes
$SYS/broker/heap/maximum size 5904 bytes
$SYS/broker/bytes/received 980
$SYS/broker/bytes/sent 5862
$SYS/broker/bytes/per second/received 0
$SYS/broker/bytes/per second/sent 2
$SYS/broker/uptime 2387 seconds
```

Client Python

```
#!/usr/bin/python

import pynotify
import mosquitto

#define what happens after connection
def on_connect(rc):
    print "Connected";

#On recipt of a message create a pynotification and show it
def on_message(msg):
    n = pynotify.Notification (msg.topic, msg.payload)
    n.show ()

#create a broker (optional)
#mqttc = mosquitto.Mosquitto("python_sub")

#define the callbacks
mqttc.on_message = on_message
mqttc.on_connect = on_connect

#connect to a localhost broker
mqttc.connect("localhost", 1883, 60, True)

#subscribe to topic test
mqttc.subscribe("sensors/toto", 2)

#keep connected to broker
while mqttc.loop() == 0:
    pass
```

Client c++ sous Qt

Téléchargement sur github:
https://github.com/toni1991/QtMqtt



Nom	Taille	Type	Modifié
.gitignore	25 octets	inconnu	01 octobre 2013, 10:28
README.md	929 oc…	docum…	01 octobre 2013, 10:28
qmqtt.h	1,8 ko	en-tête C	01 octobre 2013, 10:28
qmqtt.pro	708 oc…	inconnu	01 octobre 2013, 10:28
qmqtt_client.cpp	6,7 ko	code s…	01 octobre 2013, 10:28
qmqtt_client.h	5,1 ko	en-tête C	01 octobre 2013, 10:28
qmqtt_client_p.cpp	6,0 ko	code s…	01 octobre 2013, 10:28
qmqtt_client_p.h	2,9 ko	en-tête C	01 octobre 2013, 10:28
qmqtt_frame.cpp	3,4 ko	code s…	01 octobre 2013, 10:28
qmqtt_frame.h	3,9 ko	en-tête C	01 octobre 2013, 10:28
qmqtt_global.h	2,5 ko	en-tête C	01 octobre 2013, 10:28
qmqtt_message.cpp	2,8 ko	code s…	01 octobre 2013, 10:28
qmqtt_message.h	2,6 ko	en-tête C	01 octobre 2013, 10:28
qmqtt_network.cpp	4,7 ko	code s…	01 octobre 2013, 10:28
qmqtt_network.h	3,1 ko	en-tête C	01 octobre 2013, 10:28
qmqtt_will.cpp	2,4 ko	code s…	01 octobre 2013, 10:28
qmqtt_will.h	2,3 ko	en-tête C	01 octobre 2013, 10:28

Un exemple Qt à :

http://grimaldi.univ-tln.fr/category/files/client-MQTT.zip