

COMMUNICATIONS ENTRE CAPTEURS ET MISE EN RÉSEAU

PROGRAMME DOCTORAL
INTERNATIONAL DE MODÉLISATION
DES SYSTÈMES COMPLEXES

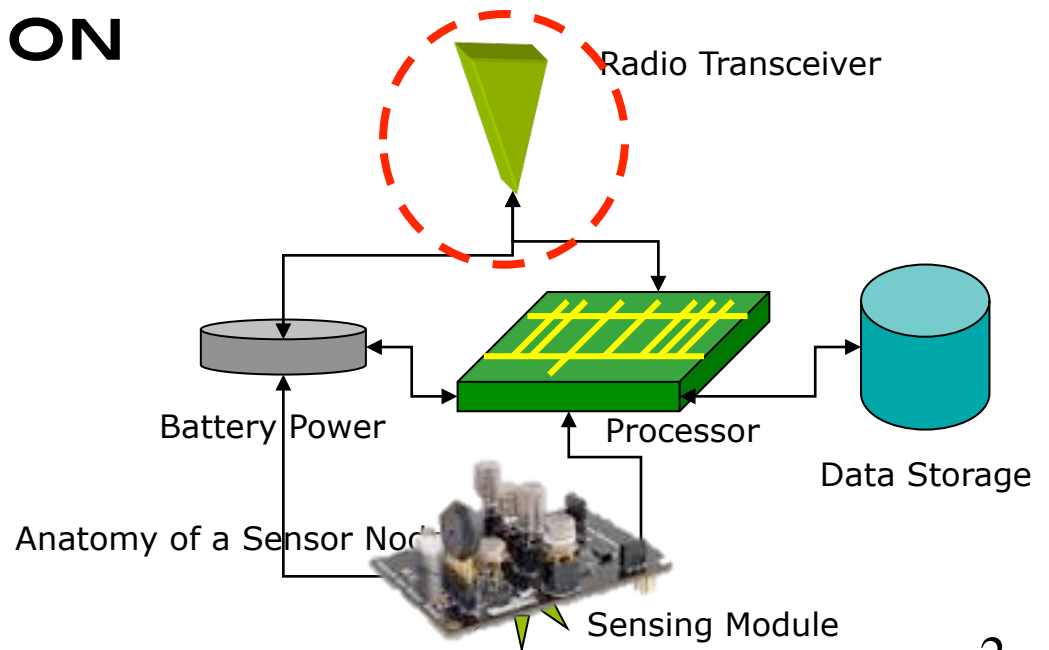
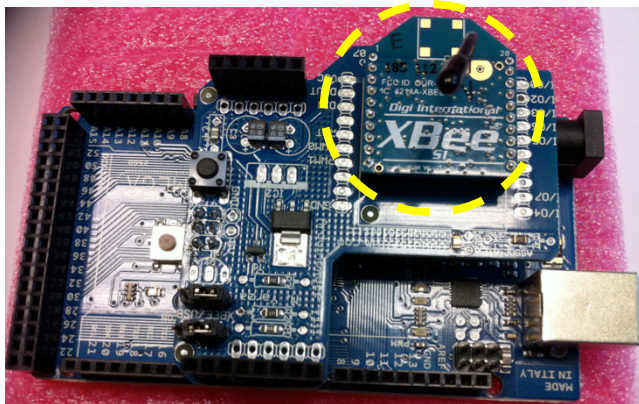


PROF. CONGDUC PHAM
[HTTP://WWW.UNIV-PAU.FR/~CPHAM](http://www.univ-pau.fr/~cpham)
UNIVERSITÉ DE PAU, FRANCE

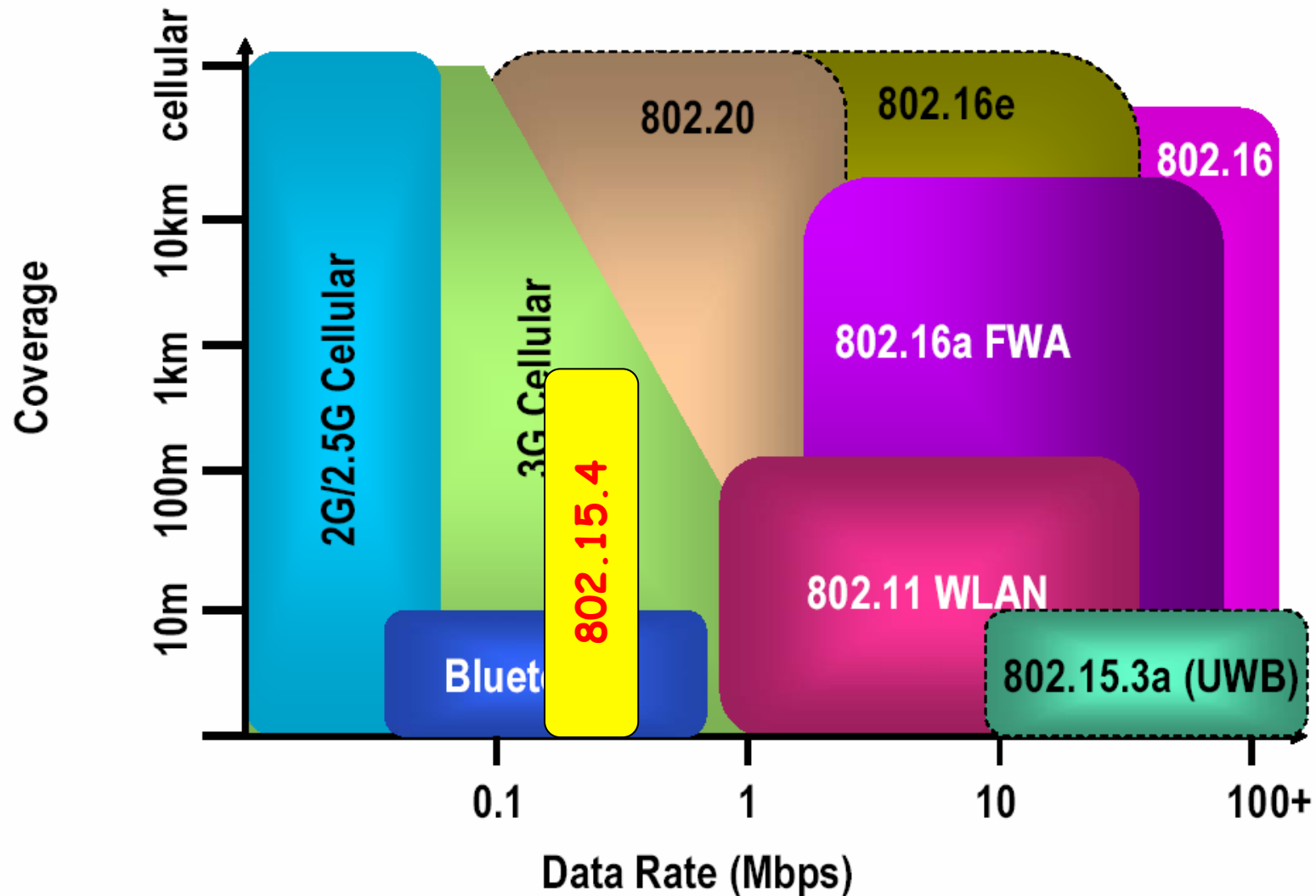


WIRELESS AUTONOMOUS SENSORS

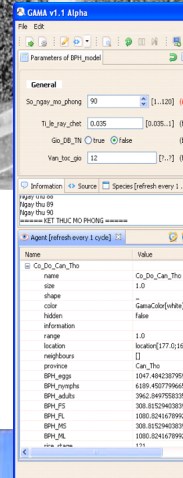
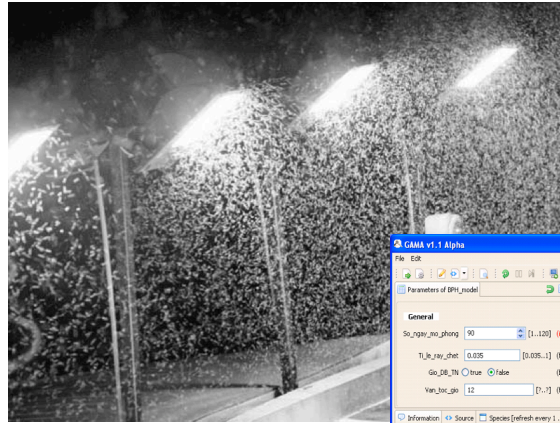
- ❑ IN GENERAL: LOW COST, LOW POWER (THE BATTERY MAY NOT BE REPLACEABLE), SMALL SIZE, PRONE TO FAILURE, POSSIBLY DISPOSABLE
- ❑ ROLE: SENSING, DATA PROCESSING, COMMUNICATION



SUMMARY OF WIRELESS TECHNOLOGIES



APPLICATION EXAMPLE



1 to 30 sensor nodes per cluster

Gateway can interconnect clusters

Communication needs:

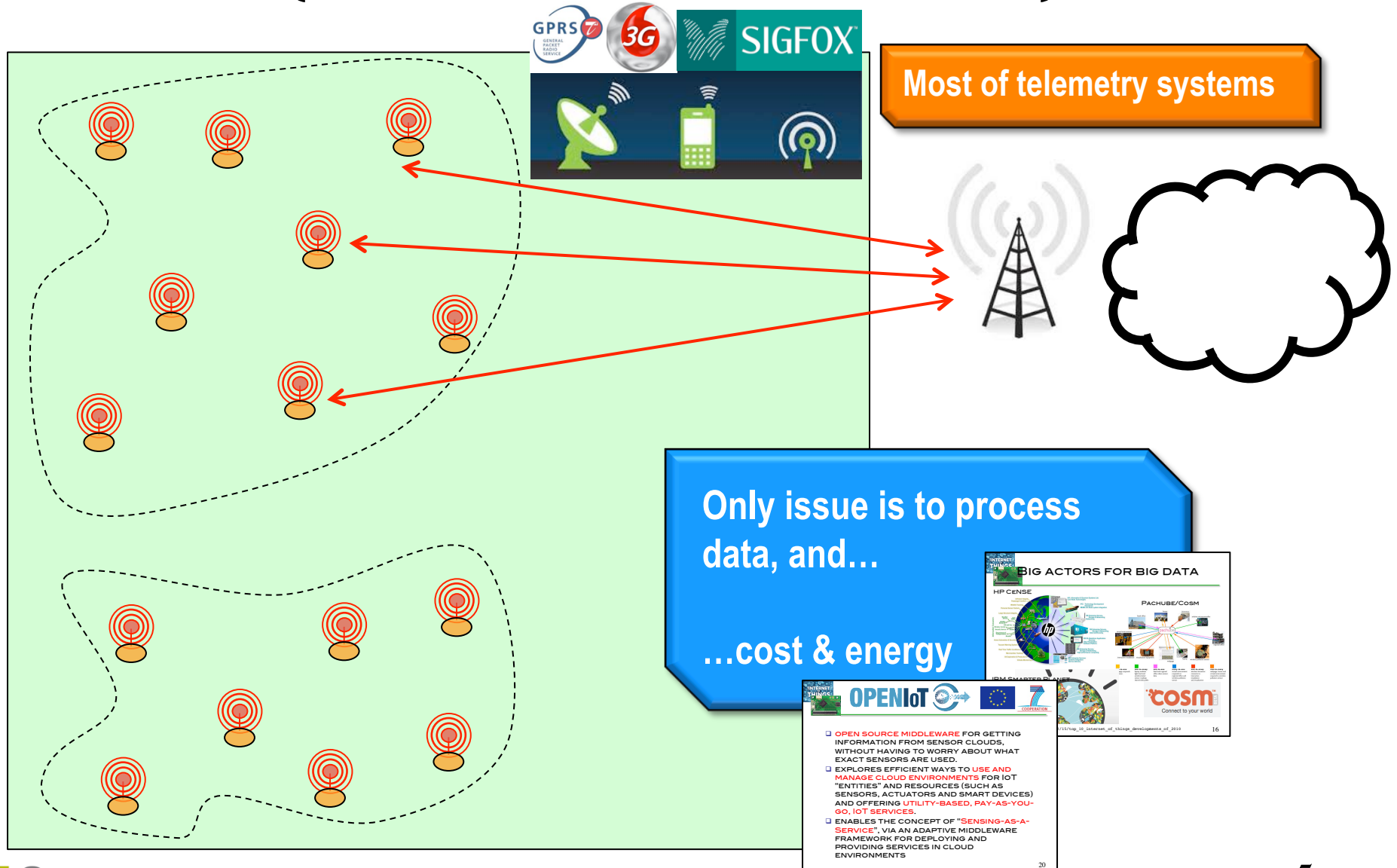
Sensor <-> Sensor

Sensor <-> Gateways

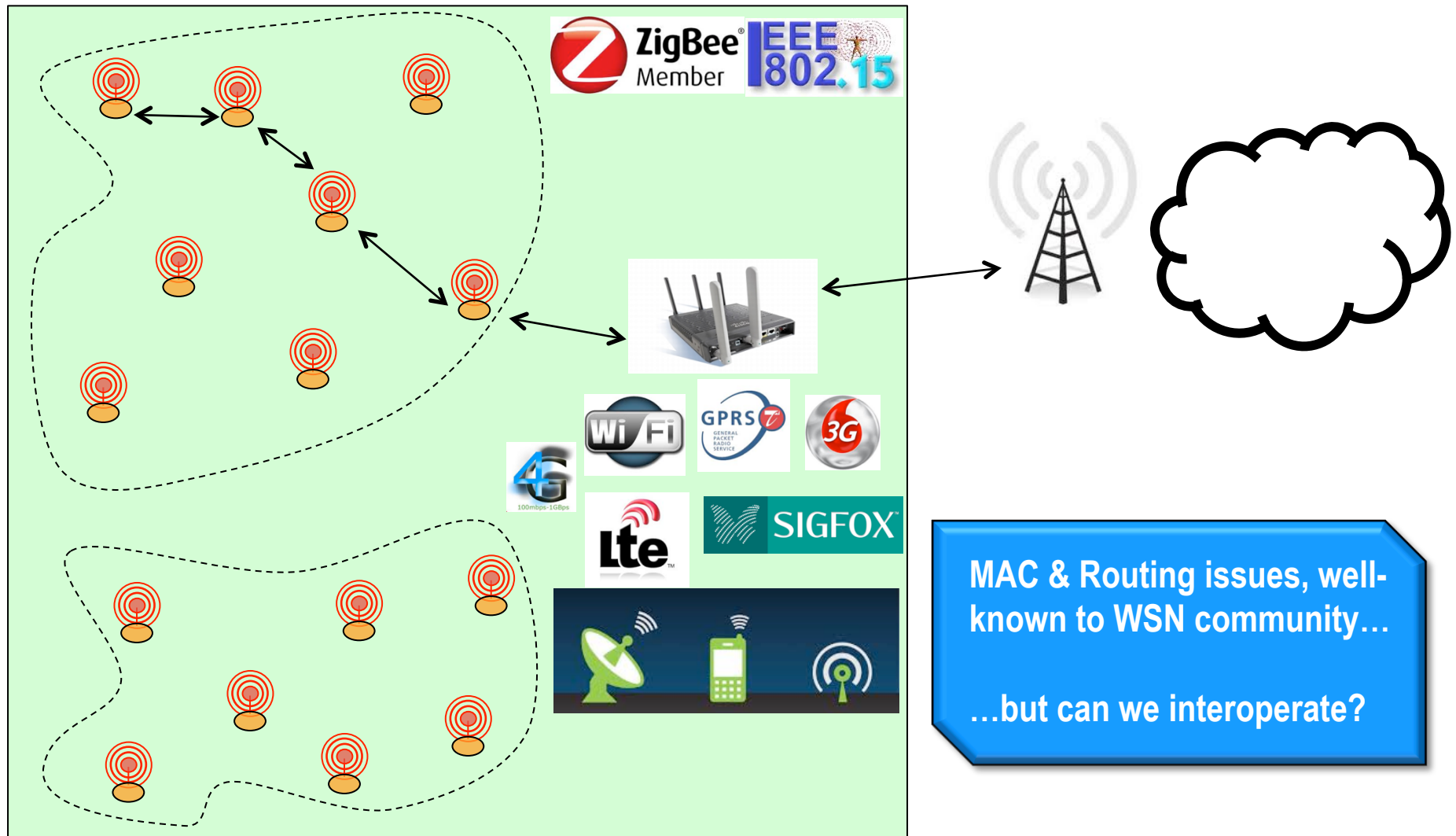
Gateways <-> Internet



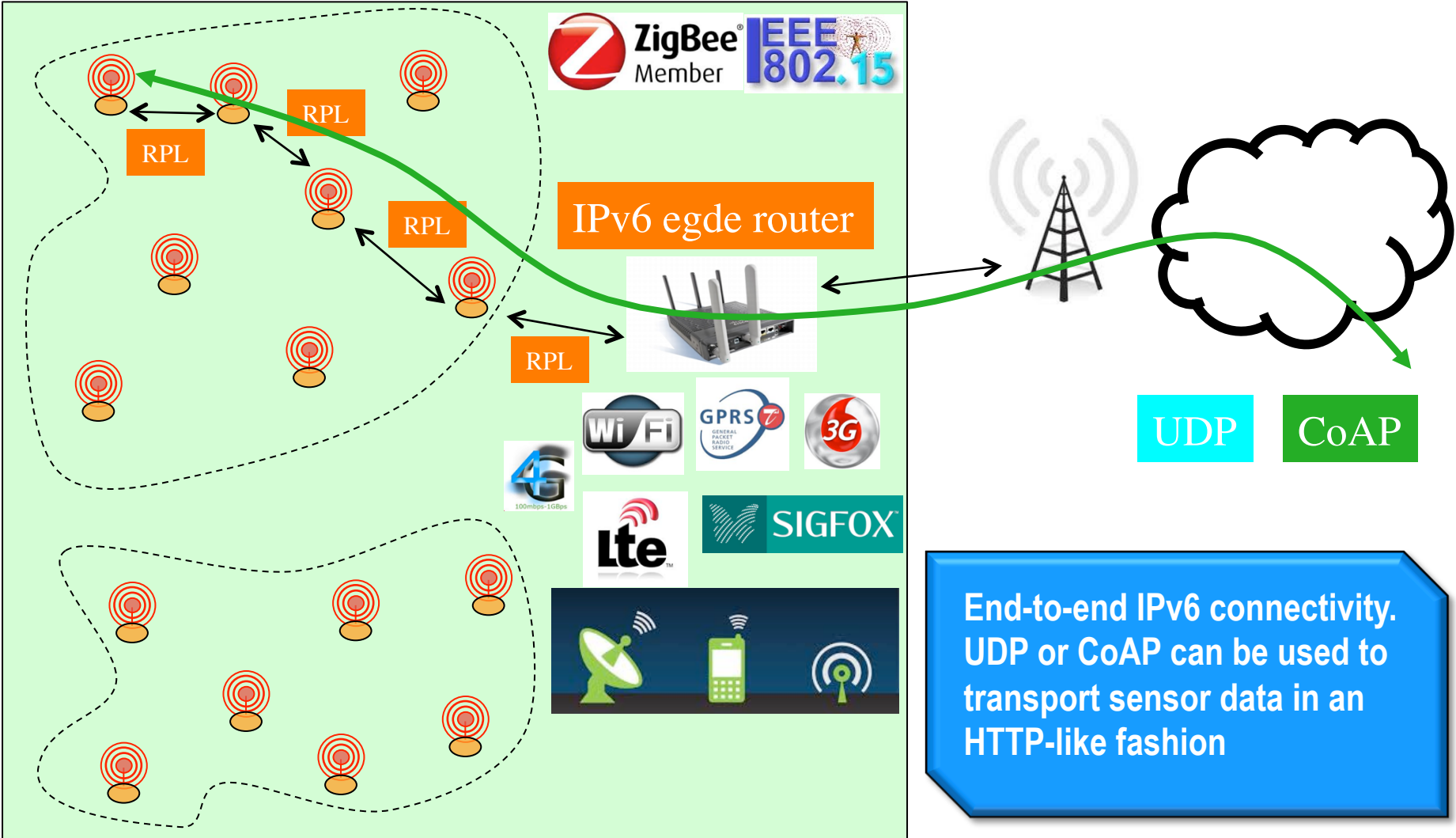
1-HOP COMMUNICATION (CELLULAR MODEL)



MULTI-HOP TO GATEWAYS



USING IP PROTOCOLS



IEEE 802.15.4

Caractéristiques Radio dans les réseaux de capteurs

- Norme ZigBee (IEEE 802.15.4 PHY)

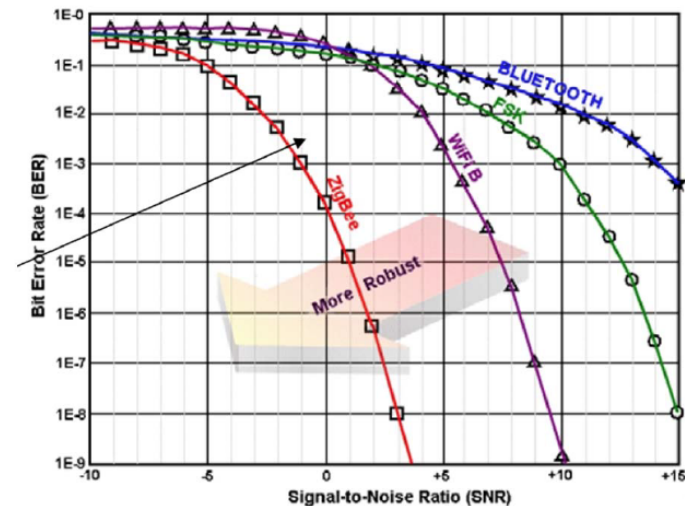
La norme IEEE802.15.4a, adaptées aux réseaux de capteurs, au contrôle industriel et aux dispositifs médicaux (CMI)

IEEE802.15.4 (couches 1 et 2):

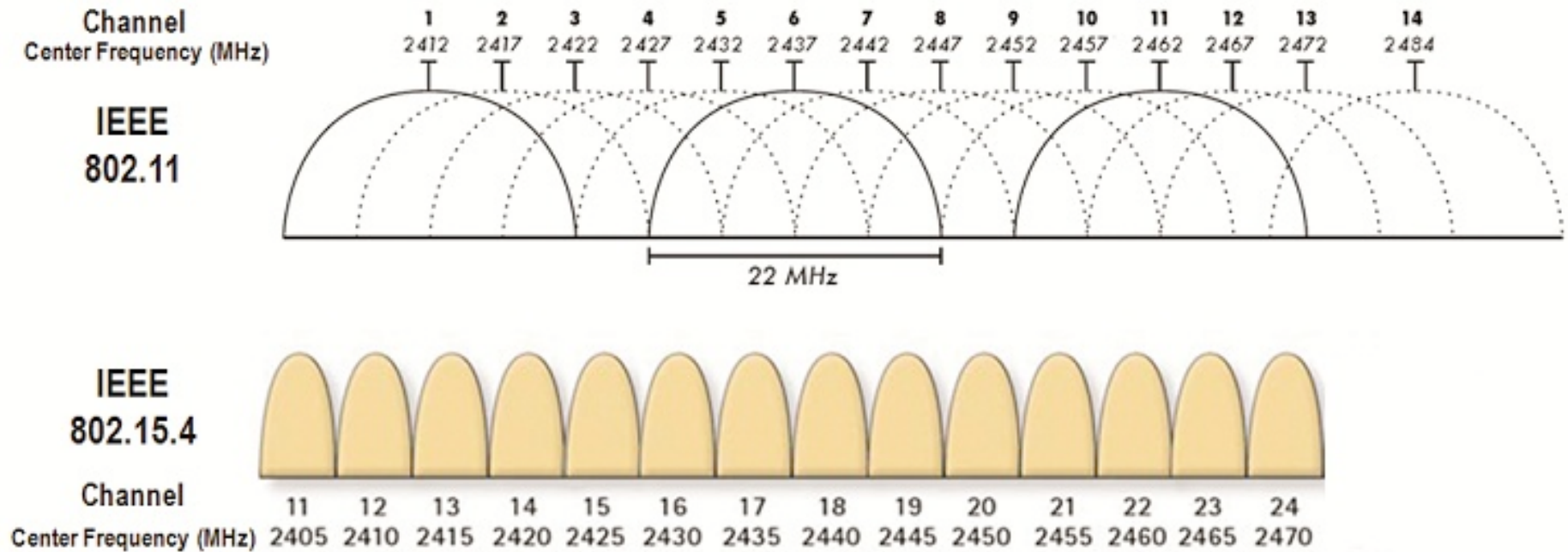
- Three bands, 27 channels specified
 - 2.4 GHz: 16 channels, 250 kbps
 - 868.3 MHz : 1 channel, 20 kbps
 - 902-928 MHz: 10 channels, 40 kbps

Protocole	Zigbee	Bluetooth	Wi-Fi
IEEE	802.15.4	802.15.1	802.11a/b/g
Besoins mémoire	4-32 Kb	250 Kb +	1 Mb +
Autonomie avec pile	Années	Jours	Heures
Nombre de nœuds	65 000+	7	32
Vitesse de transfert	250 Kb/s	1 Mb/s	11-54 et + Mb/s
Portée	100 m	10-100 m	300 m

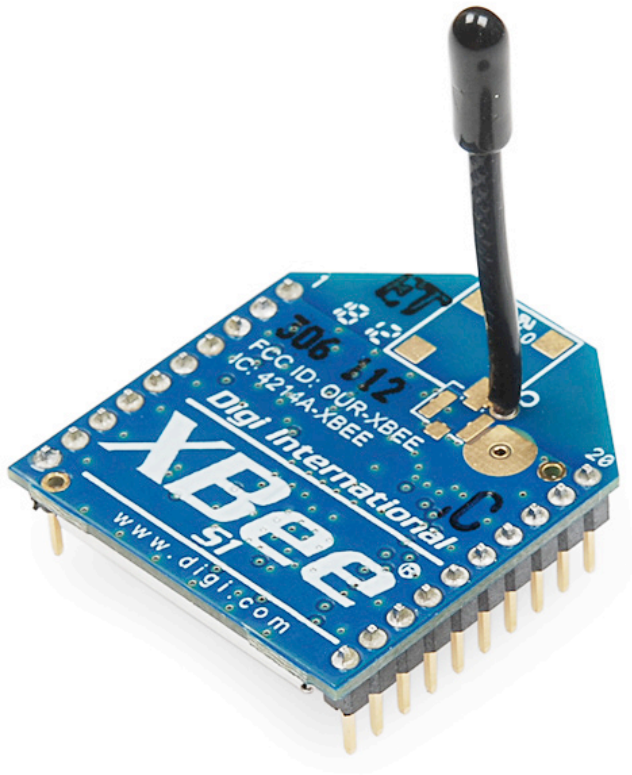
- Comparaison entre les normes ZigBee, Bluetooth et Wifi



SPECTRUM BAND



XBEE RADIO MODULE FROM DIGI



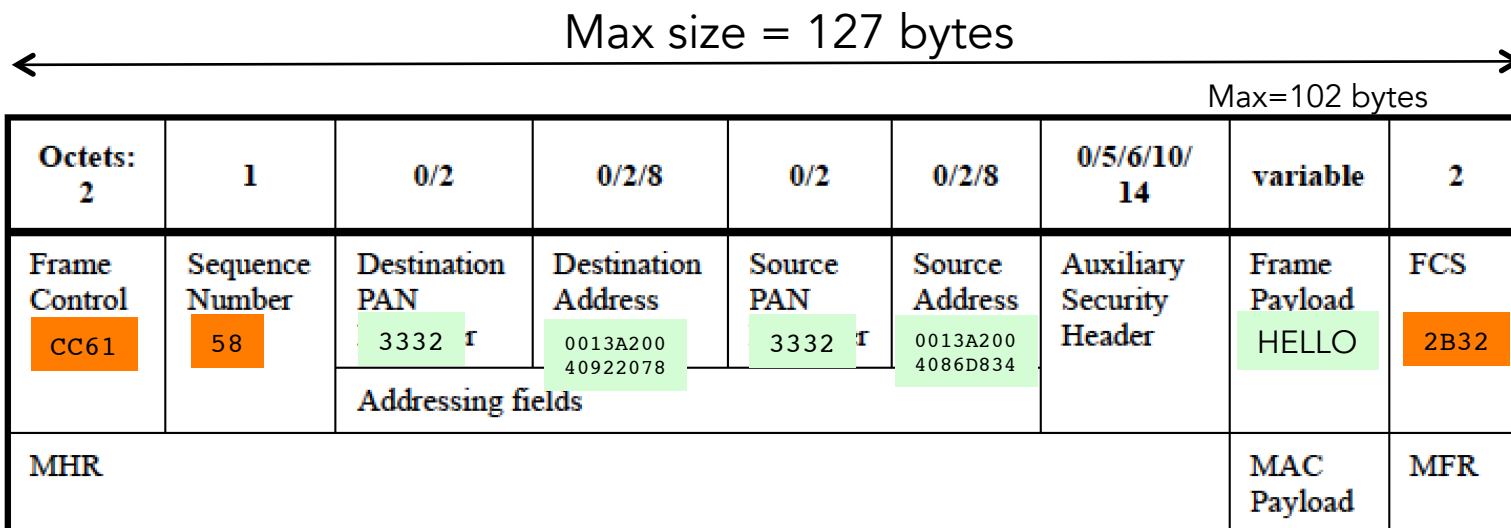
Implements IEEE 802.15.4 standard

64-bit hardware MAC address

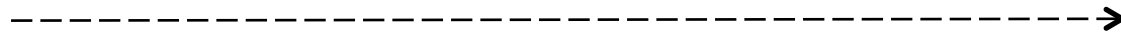
0x0013A200409C0343



MAC FRAME FORMAT



HELLO

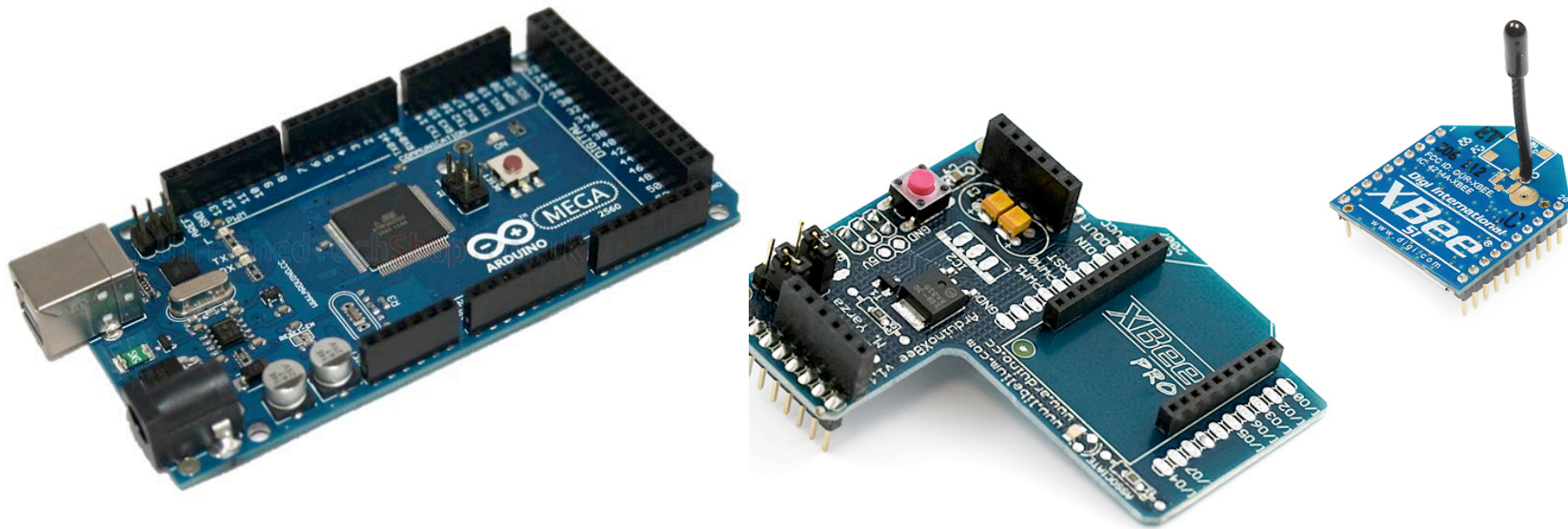


64-bit 0x0013A2004086D834
 16-bit 0x0010
 CHANNEL 0x0C
 PANID 0x3332

Can broadcast if sent to
 0x000000000000FFFF

64-bit 0x0013A20040922078
 16-bit 0x0020
 CHANNEL 0x0C
 PANID 0x3332

ARDUINO AND XBEE

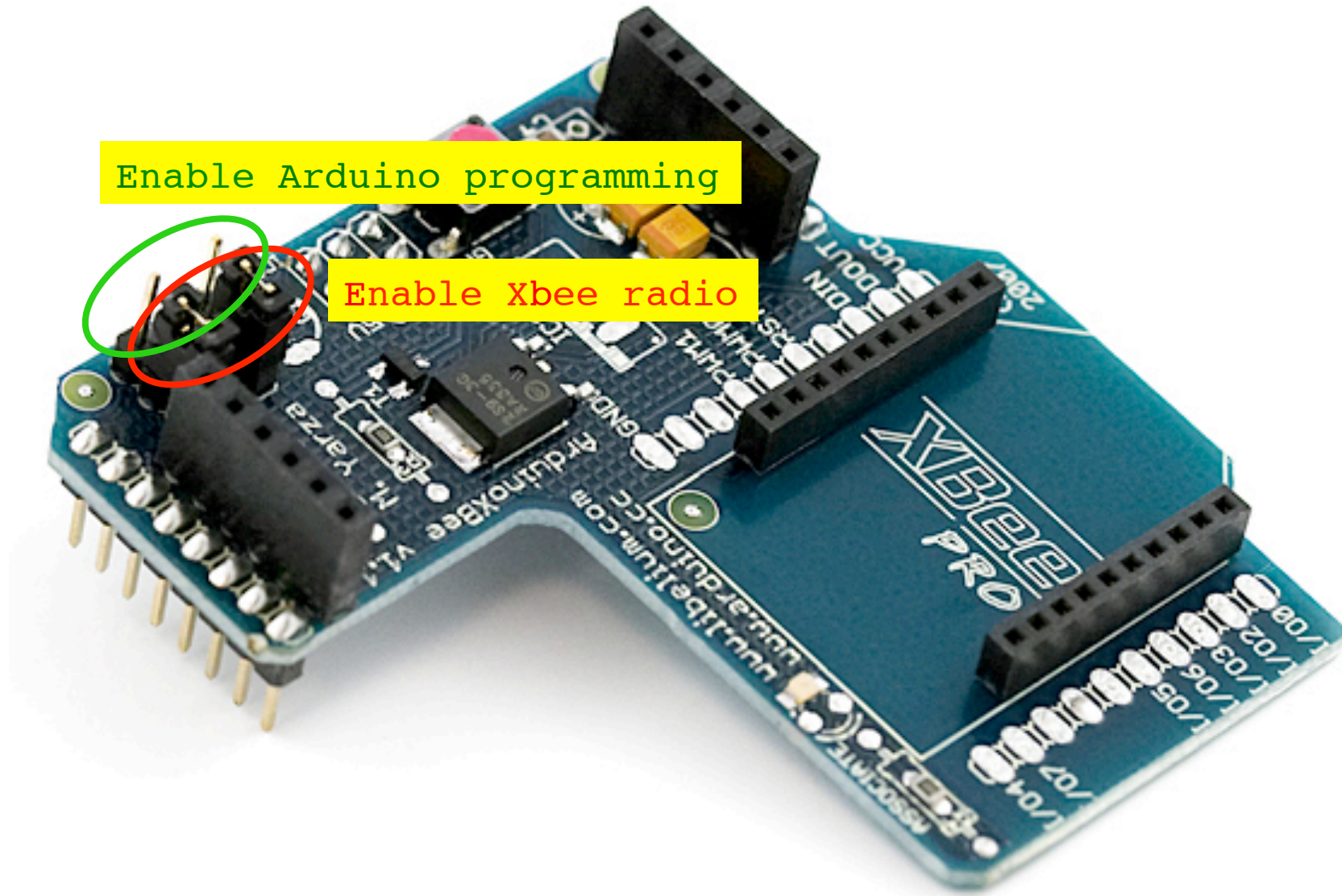


- ❑ USE SOFTWARE COMMUNICATION LIBRARY TO USE XBEE RADIO MODULE : XBEE-ARDUINO
- ❑ INSTALLATION
 - ❑ CREATE A « LIBRARIES » FOLDER IN YOUR ARDUINO SKETCH
 - ❑ COPY THE XBEE LIBRARY FOLDER IN SKETCH/LIBRARIES
- ❑ SOURCE CODE
 - ❑ [HTTPS://CODE.GOOGLE.COM/P/XBEE-ARDUINO/](https://code.google.com/p/xbee-arduino/)

XBEE SHIELD

Enable Arduino programming

Enable Xbee radio



FIRST STEP IN COMMUNICATION: TRANSMIT

```
#include <XBee.h>
#include <SoftwareSerial.h>

XBee xbee = XBee();

uint8_t payload[80];

char data[6]="hello";

void setup() {
  xbee.begin(38400);

  Serial.begin(38400);
  Serial.println("Arduino. Will send packets.");
}

void loop() {

  delay(5000);

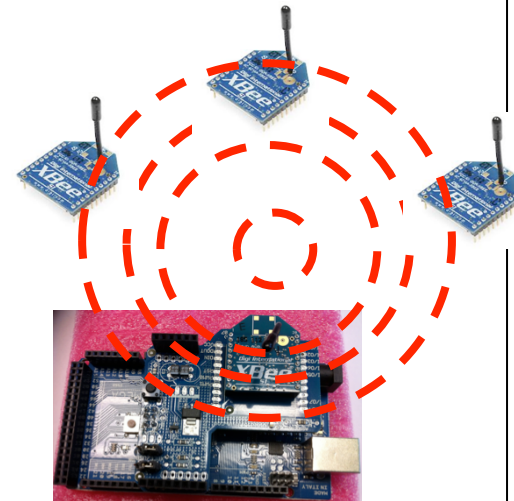
  // 64-bit addressing
  XBeeAddress64 addr64 = XBeeAddress64(0x00000000, 0x0000FFFF);

  for (int i; i<sizeof(data); i++)
    payload[i]=(uint8_t)data[i]);

  // in this way, we know the exact size of the payload
  Tx64Request tx = Tx64Request(addr64, (uint8_t*)data, sizeof(data));

  // Send your request
  xbee.send(tx);

  TxStatusResponse txStatus = TxStatusResponse();
}
```



Use broadcast, send to
0x000000000000FFFF

RADIO SNIFFER

(Untitled) - Wireshark

File Edit View Go Capture Analyze Statistics Telephony Tools Help

Filter: Expression... Clear Apply

Octets:	1	0/2	0/2/8	0/2	0/2/8	0/5/6/10/14	variable	2
2	1	0/2	0/2/8	0/2	0/2/8	0/5/6/10/14	variable	2
Frame Control	Sequence Number	Destination PAN	Destination Address	Source PAN	Source Address	Auxiliary Security Header	Frame Payload	FCS
CC61	58	3332	0013A200 40922078	3332	0013A200 4086D834		HELLO	2B32
Addressing fields								
MHR							MAC Payload	MFR

No. Time Source

1	0.000000	00:13:a2:00:40:76:20:5e
2	2.101024	00:13:a2:00:40:76:20:5e
3	4.200896	00:13:a2:00:40:76:20:5e
4	6.300768	00:13:a2:00:40:76:20:5e
5	7.824096	00:13:a2:00:40:86:d8:34
6	68683.201776	
7	8.400576	00:13:a2:00:40:76:20:5e
8	10.500416	00:13:a2:00:40:76:20:5e
9	11.066176	00:13:a2:00:40:86:d8:34
10	68683.201776	
11	12.600160	00:13:a2:00:40:76:20:5e
12	14.700160	00:13:a2:00:40:76:20:5e
13	15.163840	00:13:a2:00:40:86:d8:34
14	15.166624	00:13:a2:00:40:86:d8:34
15	15.169408	00:13:a2:00:40:86:d8:34
16	15.172224	00:13:a2:00:40:86:d8:34
17	16.799936	00:13:a2:00:40:76:20:5e
18	18.899744	00:13:a2:00:40:76:20:5e
19	20.999616	00:13:a2:00:40:76:20:5e
20	22.030464	00:13:a2:00:40:86:d8:34
21	22.033248	00:13:a2:00:40:86:d8:34
22	22.036032	00:13:a2:00:40:86:d8:34
23	22.038816	00:13:a2:00:40:86:d8:34
24	23.100576	00:13:a2:00:40:76:20:5e

23 22.038816 00:13:a2:00:40:86:d8:34 00:13:a2:00:40:92:20:78 IEEE 802.15.4 Data, Dst: Maxstrea 00:40:92:20:78, Src: Maxstrea 00:40:86:d8:34, Bad FCS

24 23.100576 00:13:a2:00:40:76:20:5e Broadcast IEEE 802.15.4 Data, Dst: Broadcast, Src: Maxstrea 00:40:76:20:5e, Bad FCS

Frame 23 (28 bytes on wire, 28 bytes captured)

Arrival Time: Jan 1, 1970 01:00:58.313760000

[Time delta from previous captured frame: 0.002784000 seconds]

[Time delta from previous displayed frame: 0.002784000 seconds]

[Time since reference or first frame: 22.038816000 seconds]

Frame Number: 23

Frame Length: 28 bytes

Capture Length: 28 bytes

[Frame is marked: False]

[Protocols in frame: wpan:data]

IEEE 802.15.4 Data, Dst: Maxstrea 00:40:92:20:78, Src: Maxstrea 00:40:86:d8:34, Bad FCS

- Frame Control Field: Data (0xcc61)
 - Sequence Number: 88
 - Destination PAN: 0x3332
 - Destination: Maxstrea 00:40:92:20:78 (00:13:a2:00:40:92:20:78)
 - Source: Maxstrea 00:40:86:d8:34 (00:13:a2:00:40:86:d8:34)
 - FCS: 0xffff (Incorrect, expected FCS=0x2b32)
 - [Expert Info (Warn/Checksum): Bad FCS]
- Data (5 bytes)
 - Data: 48454c4c4f
 - [Length: 5]

0000 61 cc 58 32 33 78 20 92 40 00 a2 13 00 34 d8 86 a.X23x . @. . . . 4. . .

0010 40 00 a2 13 00 48 45 4c 4c 4f ff ff @. . . . HEL LO. . .

Frame (frame), 28 bytes Packets: 26 Displayed: 26 Marked: 0 Profile: Default

NEXT STEP IN COMMUNICATION: RECEIVE

```
#include <XBee.h>
#include <SoftwareSerial.h>

XBee xbee = XBee();

uint8_t* payload;
uint8_t payload_len;

Rx64Response rx64 = Rx64Response();

void setup() {
  xbee.begin(38400);
  Serial.begin(38400);
  Serial.println("Arduino. Will receive packets.");
}

void loop() {
  // read incoming pkt
  xbee.readPacket();

  if (xbee.getResponse().isAvailable()) {

    // is it a response to the previously sent packet?
    if (xbee.getResponse().getApiId() == TX_STATUS_RESPONSE) {
    }

    if (xbee.getResponse().getApiId() == RX_64_RESPONSE) {

      xbee.getResponse().getRx64Response(rx64);

      payload = rx64.getData();
      payload_len=rx64.getDataLength();

      for (int i=0; i<payload_len; i++)
        Serial.print((char)payload[i]);

      Serial.println(" ");
    }
  }
}
```


TRANSMITTING BINARY DATA

```
#include <XBee.h>
#include <SoftwareSerial.h>

XBee xbee = XBee();

// allocate two bytes for to hold a 10-bit analog reading
uint8_t payload[] = { 0, 0 };

int pin5 = 0;

void setup() {
  xbee.begin(38400);

  Serial.begin(38400);
  Serial.println("Arduino. Will send binary data.");
}

void loop() {
  delay(5000);

  // break down 10-bit reading into two bytes and place in payload
  pin5 = analogRead(5);
  Serial.println(pin5);
  payload[0] = pin5 >> 8 & 0xff;
  payload[1] = pin5 & 0xff;

  // 64-bit addressing: This is the SH + SL address of remote XBee
  XBeeAddress64 addr64 = XBeeAddress64(0x00000000, 0x0000FFFF);
  // unless you have MY on the receiving radio set to FFFF, this will be received as a RX16 packet

  // in this way, we know the exact size of the payload
  Tx64Request tx = Tx64Request(addr64, payload, sizeof(payload));

  // Send your request
  xbee.send(tx);

  TxStatusResponse txStatus = TxStatusResponse();
}
```

963₁₀ on 10 bits
1111000011₂
0000001111000011₂

000000000000000011₂
11111111₂
payload[0] 00000011₂
0x03

0000001111000011₂
11111111₂
payload[1] 11000011₂
0xC3

RECEIVE BINARY DATA

```
#include <XBee.h>
#include <SoftwareSerial.h>

XBee xbee = XBee();

uint8_t* payload;
uint8_t payload_len;

Rx64Response rx64 = Rx64Response();

void setup() {
  xbee.begin(38400);
  Serial.begin(38400);
  Serial.println("Arduino. Will receive packets.");
}

void loop() {
  // read incoming pkt
  xbee.readPacket();

  if (xbee.getResponse().isAvailable()) {

    // is it a response to the previously sent packet?
    if (xbee.getResponse().getApiId() == TX_STATUS_RESPONSE) {
    }

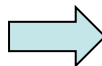
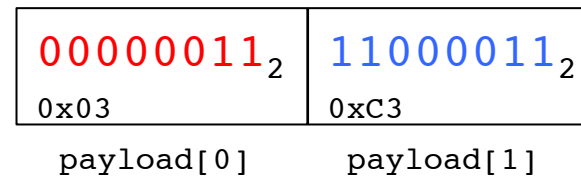
    if (xbee.getResponse().getApiId() == RX_64_RESPONSE) {

      xbee.getResponse().getRx64Response(rx64);

      payload = rx64.getData();
      payload_len=rx64.getDataLength();

      for (int i=0; i<payload_len; i++)
        Serial.print((char)payload[i]);

      Serial.println(" ");
    }
  }
}
```



```
int value;

value = payload[0] << 8 & 0xffff;
value += payload[1];

Serial.print(value);
```