Contiki: Radio Communication

1st Workshop on Appl. of WSN for Env. Monitoring in Developing Countries

Trieste (Italy), 03.03.2011

Structure of the Lectures

Weekday	10:00 – 11:00	11:30 – 12:30	14:00 – 16:00	16:30 – 18:30
Wednesday, 02.03.2011	Introduction to Contiki	Contiki: Basic Programming	WSN Lab Contiki: Basic Examples	WSN Lab Contiki: Basic Examples
Thursday, 03.03.2011	Contiki: Radio Communication	Contiki: Radio Communication	WSN Lab Project I	WSN Lab Project I
Friday, 04.03.2011	Simple Electronics Crash Course	Building Applications Crash Course	WSN Lab Project II	WSN Lab Project II

Quick Summary

- **What we learnt so far…**
  - Contiki basics
  - Creating a new application
  - Compiling and running a program
  - Interfacing with the serial port
  - Using timers
  - Using LEDs
  - Reading basic data from the sensors

Quick Summary

- **What's next?**
  - Wireless Sensor Networks

Quick Summary

- **What's next?**
  - Wireless Sensor Networks
  - Networks = Communications

Quick Summary

- **What's next?**
  - Wireless Sensor Networks
  - Networks = Communications
  - Wireless communications = Troubles
    - Limited range
    - Obstacles (no LoS)
    - Unreliable asymmetric links
    - Radio propagation
    - Interference

Contiki: Radio Communication

➢ Outline

- Wireless communications
  - Benefits
  - Drawbacks
  - IEEE 802.15.4
  - Interference and coexistence
- Low-power radios
  - TI CC2420
  - Link quality and link estimation
- Communicating using Contiki
  - The Rime Stack

Contiki: Radio Communication

➢ Outline

- Wireless communications
  - Benefits
  - Drawbacks
  - IEEE 802.15.4
  - Interference and coexistence
- Low-power radios
  - TI CC2420
  - Link quality and link estimation
- Communicating using Contiki
  - The Rime Stack

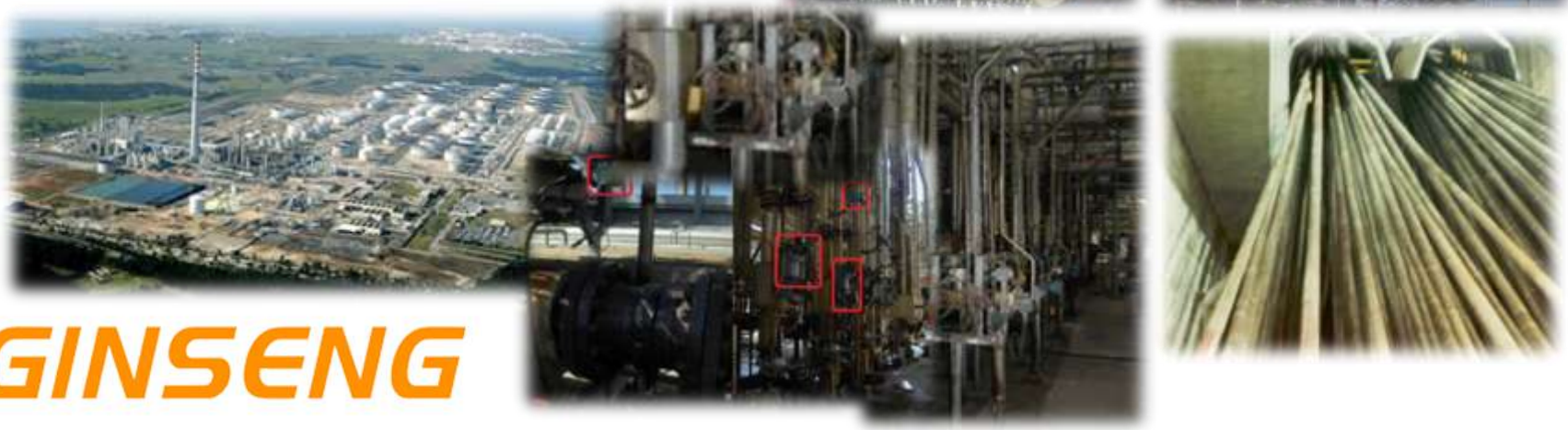
Wireless Communications

- **In a world of wires…**

- **… the flexibility offered by Wireless Sensor Networks is extremely welcome!**

Wireless Communications

- **One of the greatest advantages of Wireless Sensor Networks is their flexibility**
  - Industrial settings
  - Example: the Ginseng project



GINSENG

Wireless Communications

- **One of the greatest advantages of Wireless Sensor Networks is their flexibility**
  - Place them wherever you like
  - Example: lighting in buildings

Wireless Communications

- **Wired vs. Wireless**

Wired Systems	Wireless Systems
Unflexible (mantainance)	Highly flexible
Cabling is expensive	Cheap radios
Reliable connections	Unreliable connections
Security & Privacy	Energy-consuming
Fast data rates	Low capacity
Complete coverage not possible	Coverage also in case of obstacles

Wireless Communications

- **Wired vs. Wireless**

Wireless Communications

- **However there is a major barrier for the wide adoption of Wireless Sensor Networks from the Industry…**
  - Safety-critical applications have stringent requirements
    - Delivery rate, latency, robustness, reliability

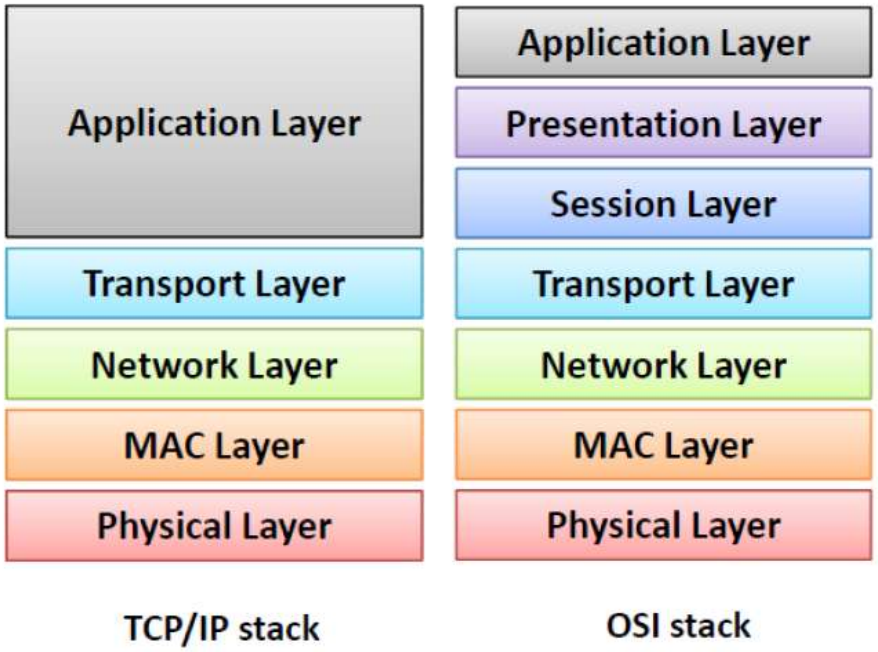
Wireless communications are unreliable

Wireless Communications

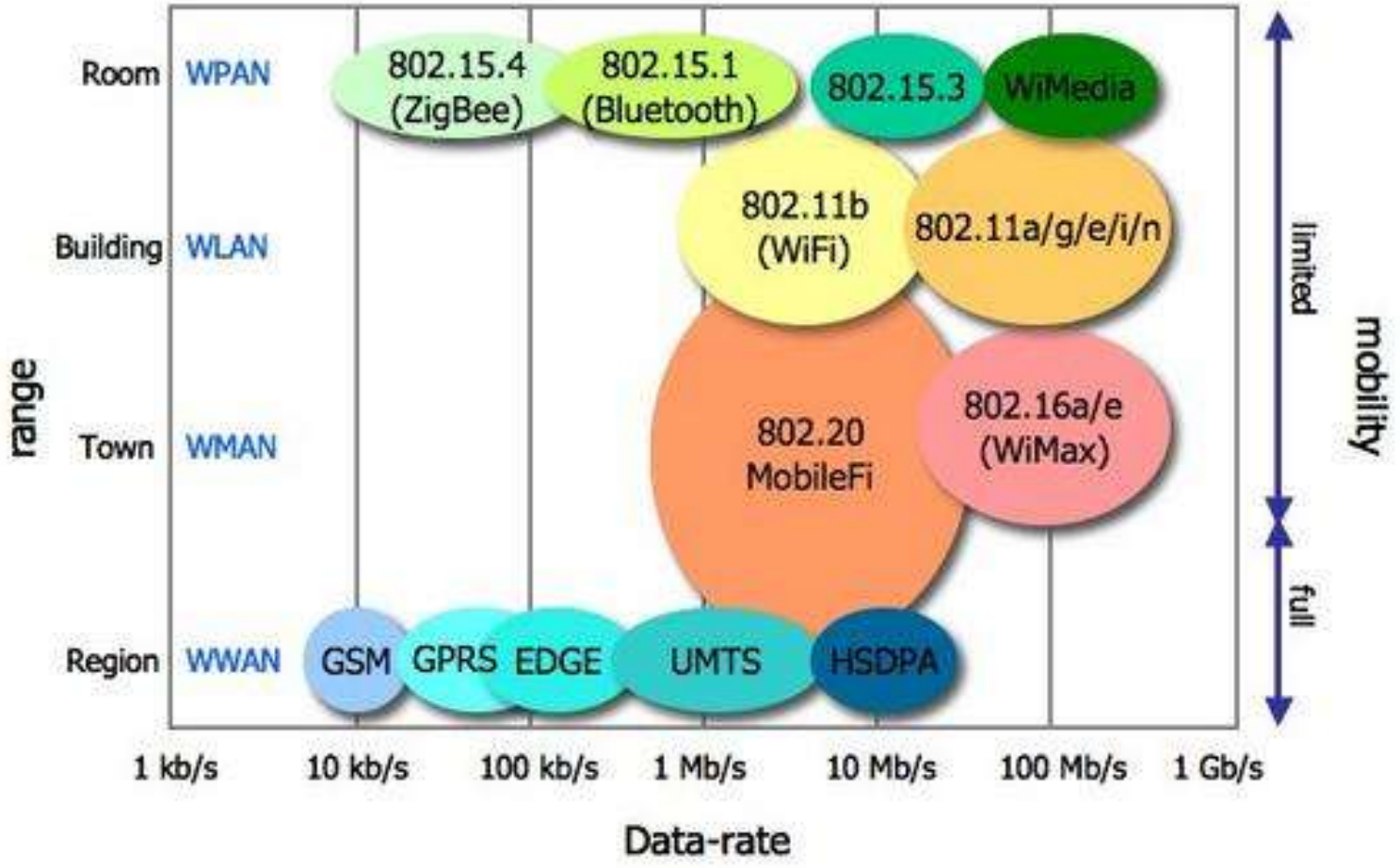
- **Unreliable Wireless Communications**
  - Crowded ISM bands
  - Example: the 2.4 GHz freq.

Wireless Communications

- **IEEE 802.15.4**
  - The standard for wireless sensor networks
  - MAC and PHY specifications
  - PHY: three ISM Bands
    - 868.0 – 868.6 MHz (EU)
    - 902 – 928 MHz (USA)
    - 2400 – 2483.5 MHz



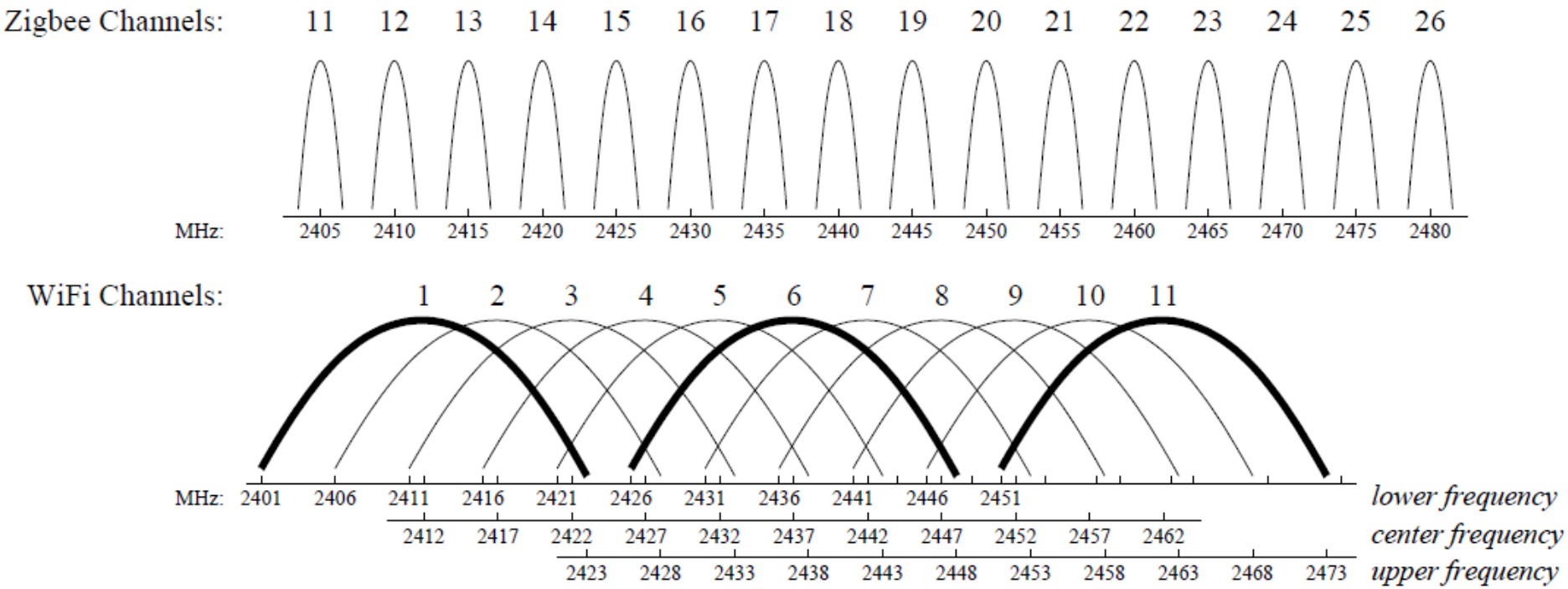
Wireless Communications



Wireless Communications

➢ Coexistence in the 2.4 GHz Band

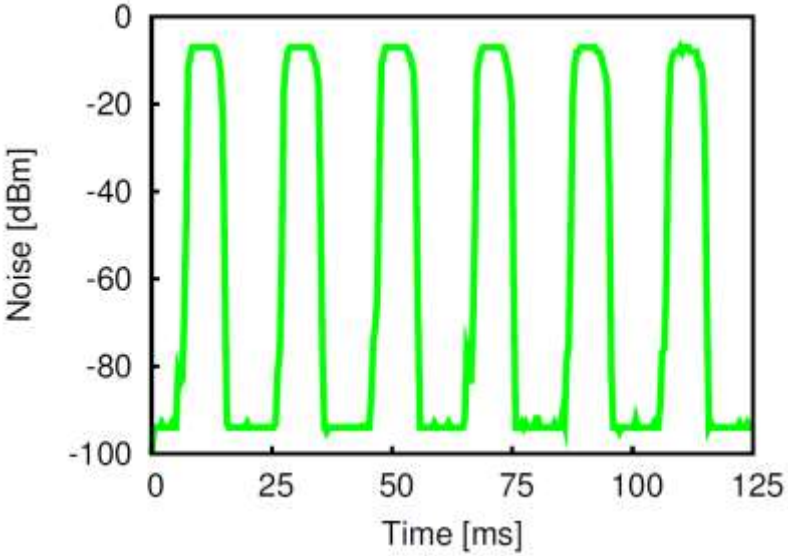
- Wi-Fi and IEEE 802.15.4



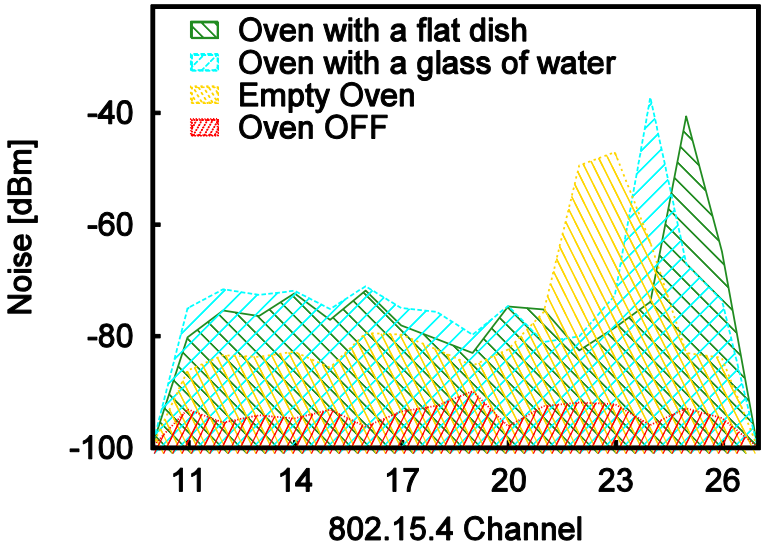
Wireless Communications

➢ Coexistence in the 2.4 GHz Band

- Microwave ovens



a) Time



b) Frequency

C.A. Boano, K. Römer, T. Voigt, C. Noda, and M. Zuniga. **JamLab: Augmenting Sensornet Testbeds with Realistic and Controlled Interference Generation.** *To appear in Proceedings of the 10th IPSN, Chicago, USA. April 2011.*

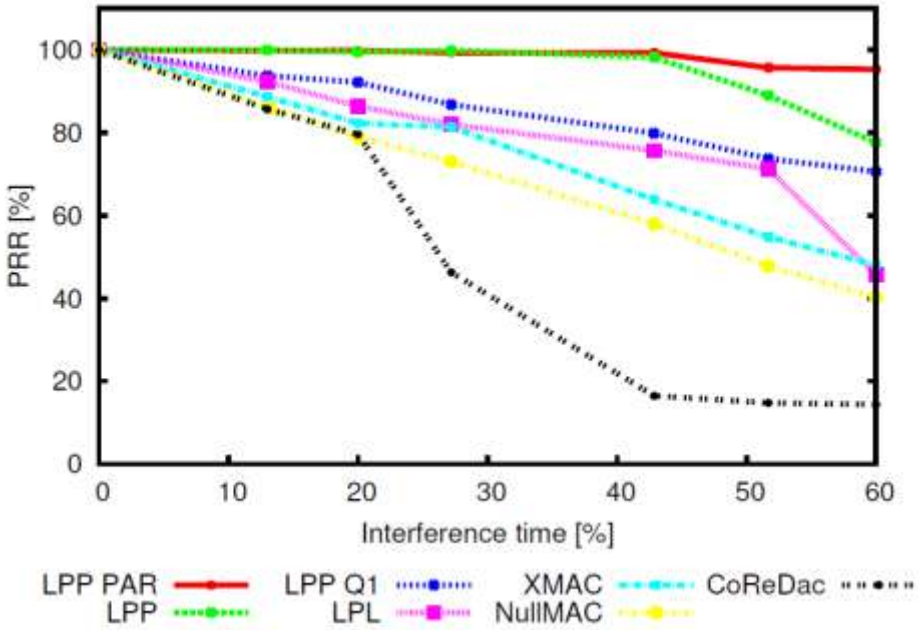
Wireless Communications

- **Interference deteriorated quality of communicatoins**
  - Packet loss
  - Lack of connectivity
  - High latencies
  - Retransmissions
  - Worse performance
  - Reduced energy-efficiency

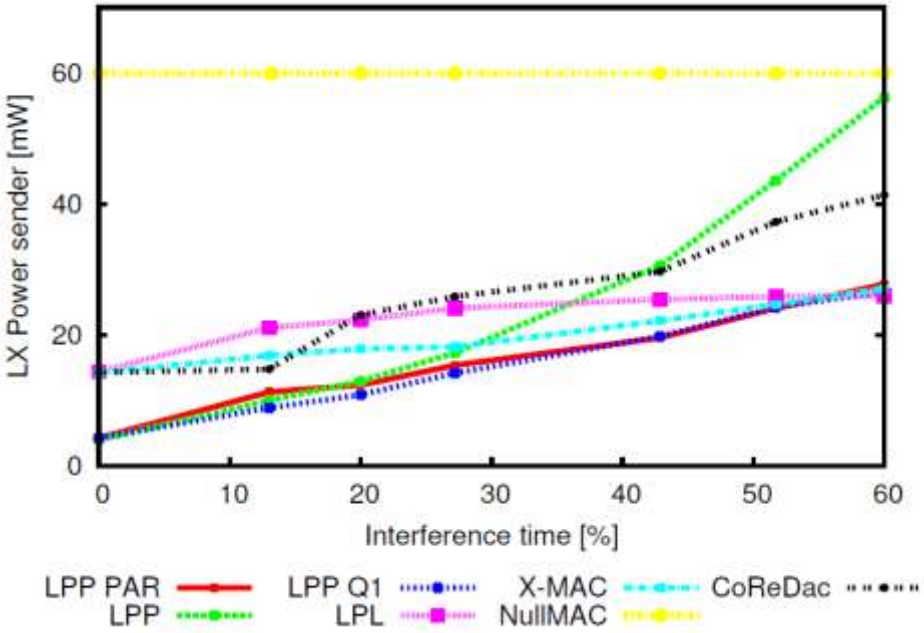
Wireless Communications

- **Interference deteriorated quality of communicatoins**
  - Experimental study on sensornet MAC protocols

Packet Reception Rate (RX)



Energy consumption (TX)



C.A. Boano, T. Voigt, N. Tsiftes, L. Mottola, K. Römer, and M. Zuniga. **Making Sensornet MAC Protocols Robust Against Interference.** *In Proc. of the 7th Conf. on Wireless Sensor Networks (EWSN).* Coimbra, Portugal. February 2010.

Wireless Communications

- **It's not only a matter of interference…**
  - Multipath
  - Fading
  - Link asymmetries
  - Attenuations
  - Obstacles
  - Mobility of nodes
  - …

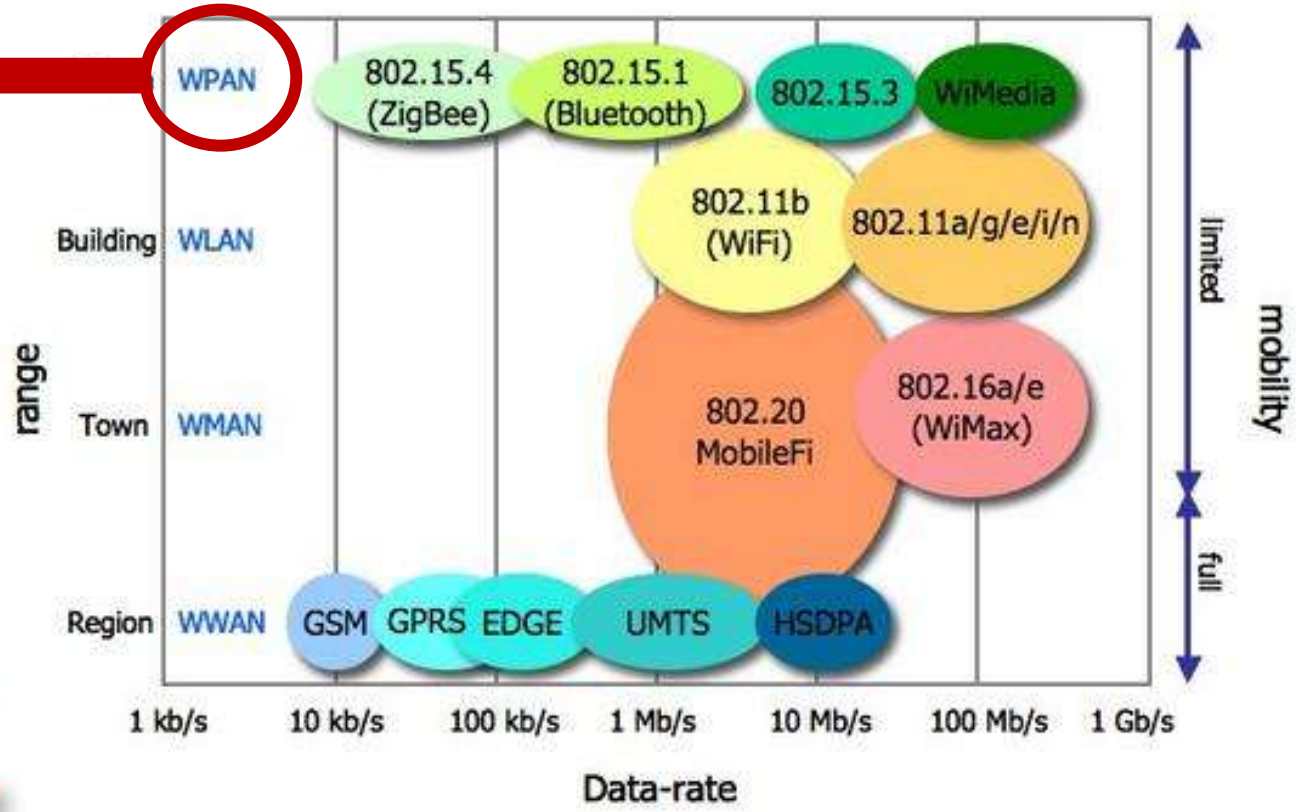
Contiki: Radio Communication

➢ Outline

- Wireless communications
  - Benefits
  - Drawbacks
  - IEEE 802.15.4
  - Interference and coexistence
- Low-power radios
  - TI CC2420
  - Link quality and link estimation
- Communicating using Contiki
  - The Rime Stack

Low-Power Radios

What is a WPAN?

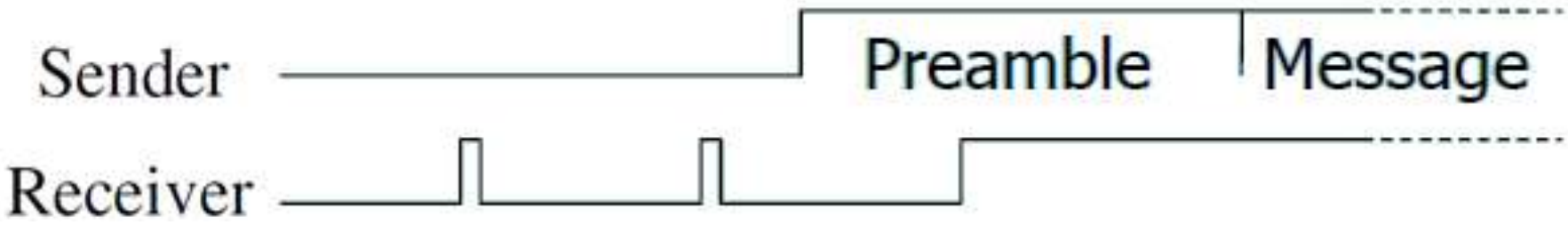


Low-Power Radios

- **Why low-power radios?**
  - We want our sensor nodes to last months or years with common batteries
  - The typical consumption of a low-power radio is still in the order of tens of mA for receiving and transmitting
  - So… how to save energy?

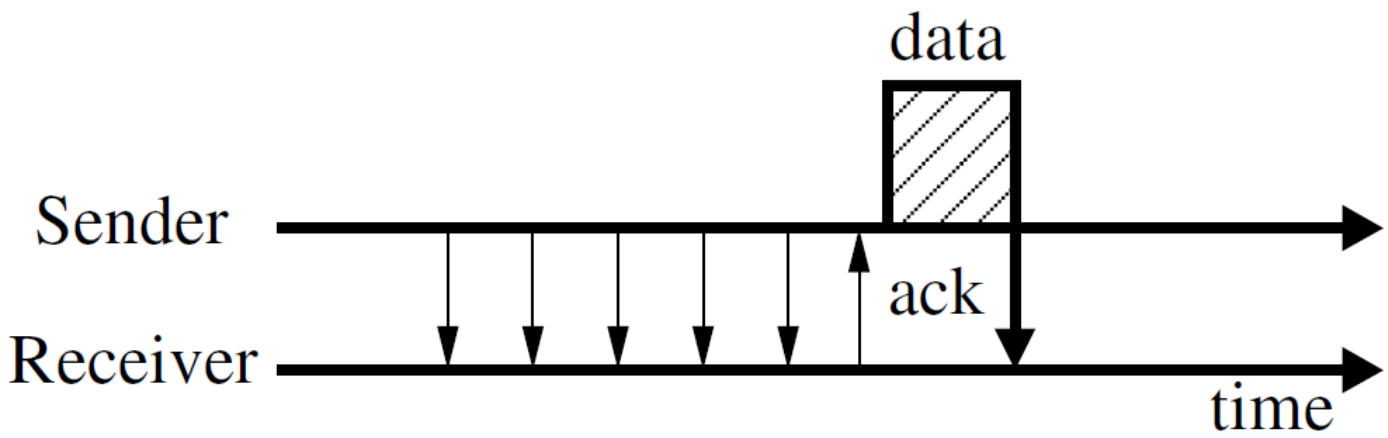
Low-Power Radios

- **Radio Duty-Cycling**
  - Turn off the radio in sleep mode
  - Low-power Listening



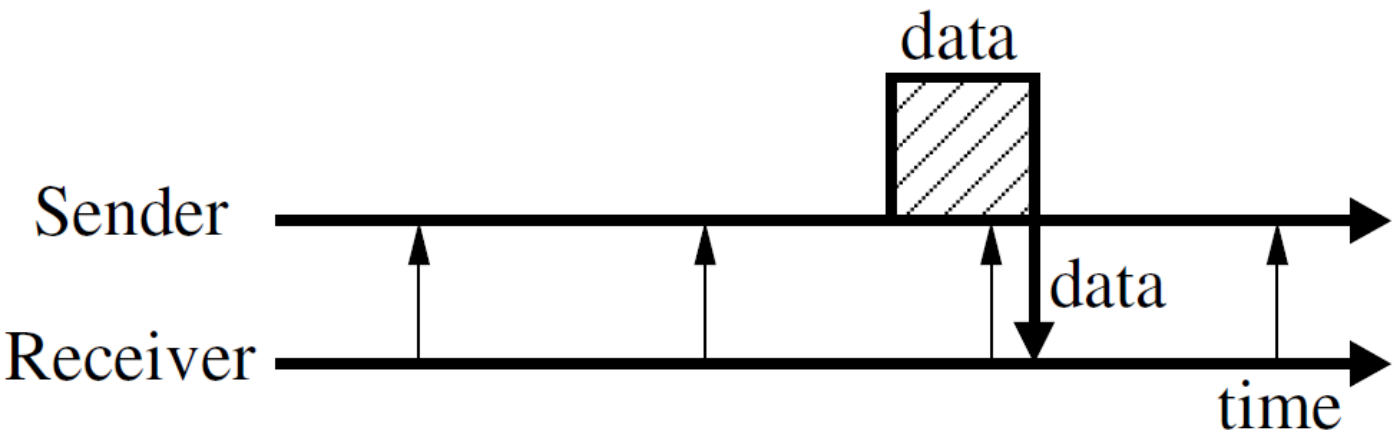
Low-Power Radios

- **Radio Duty-Cycling**
  - Examples of duty-cycling MAC protocols
  - X-MAC (Buettner et al. - 2006)
    - Sender-initiatied



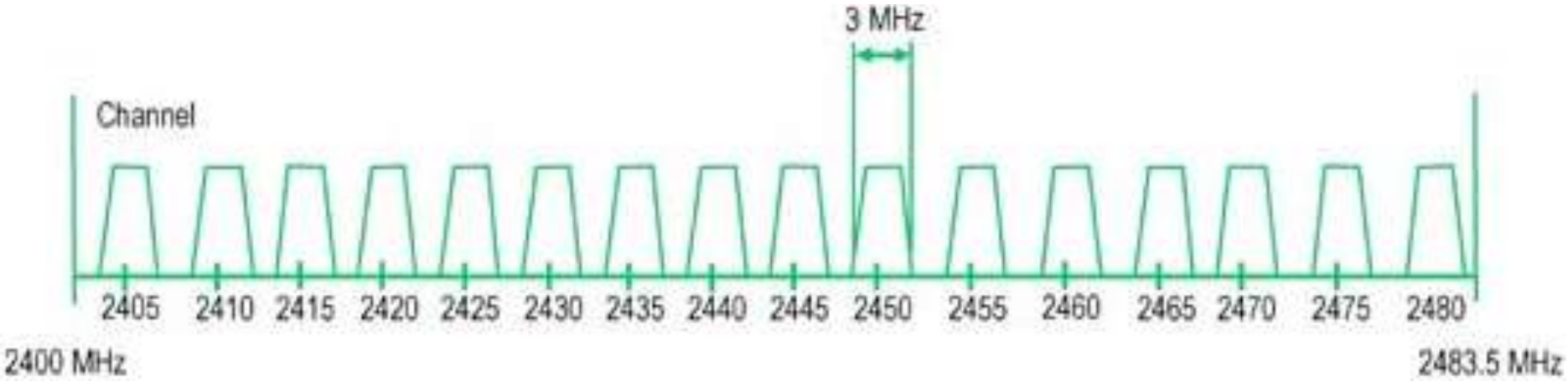
Low-Power Radios

- **Radio Duty-Cycling**
  - Examples of duty-cycling MAC protocols
  - LPP (Musaloiu et al. - 2008)
    - Receiver-initiatied



Low-Power Radios

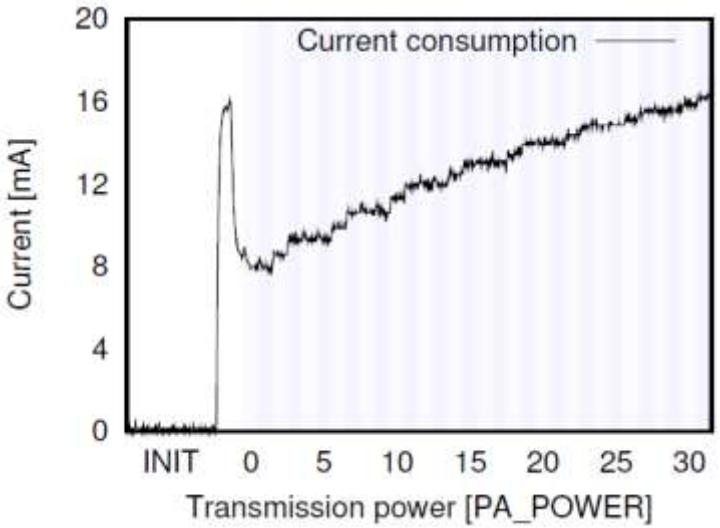
- **The CC2420 Radio**
  - Operates in the 2.4 GHz ISM band
  - 16 radio channels
    - Numbered from 11 to 26
    - Channel 11 centered at 2405 MHz, channel 12 at 2410 MHz, etc.



Low-Power Radios

➢ The CC2420 Radio

- Tunable transmission power
  - 32 PA_POWER power levels (from 0 to 31)
  - Characteristic is not linear



PA_LEVEL	Output Power [dBm]	Current Consumption [mA]
31	0	17.4
27	-1	16.5
23	-3	15.2
19	-5	13.9
15	-7	12.5
11	-10	11.2
7	-15	9.9
3	-25	8.5

Carlo Alberto Boano, James Brown, Nicolas Tsiftes, Utz Roedig, and Thiemo Voigt. **The Impact of Temperature on Outdoor Industrial Sensornet Applications.** *In IEEE Transactions on Industrial Informatics, Volume 6, Number 3, pag. 451-459.* August 2010.

Low-Power Radios

➢ The CC2420 Radio

- Primitives in Contiki
  - #include "dev/cc2420.h"
  - int cc2420_on(void);
  - int cc2420_off(void);
  - int cc2420_init(void);

The radio initialization is automatic at program startup

  - void cc2420_set_txpower(uint8_t power);
  - int cc2420_get_txpower(void);
  - CC2420_TXPOWER_MAX = 31
  - CC2420_TX_POWER_MIN = 0

Setting the transmission power

  - void cc2420_set_channel(int channel);

Setting the physical channel

Low-Power Radios

➢ The CC2420 Radio

- Example on how to set the channel and the power

```
#include <stdio.h>
#include "contiki.h"
#include "dev/cc2420.h“
#define POWER 11
#define CHANNEL 24
PROCESS(name_process, “Name of the process");
AUTOSTART_PROCESSES(&name_process);
PROCESS_THREAD(name_process, ev, data)
{
        PROCESS_EXITHANDLER()
        PROCESS_BEGIN();
        cc2420_set_txpower(POWER);
        cc2420_set_channel(CHANNEL);
        …
        PROCESS_END();
}
```

The radio initialization is automatic at program startup

(if you do not specify anything Contiki starts in channel 26)

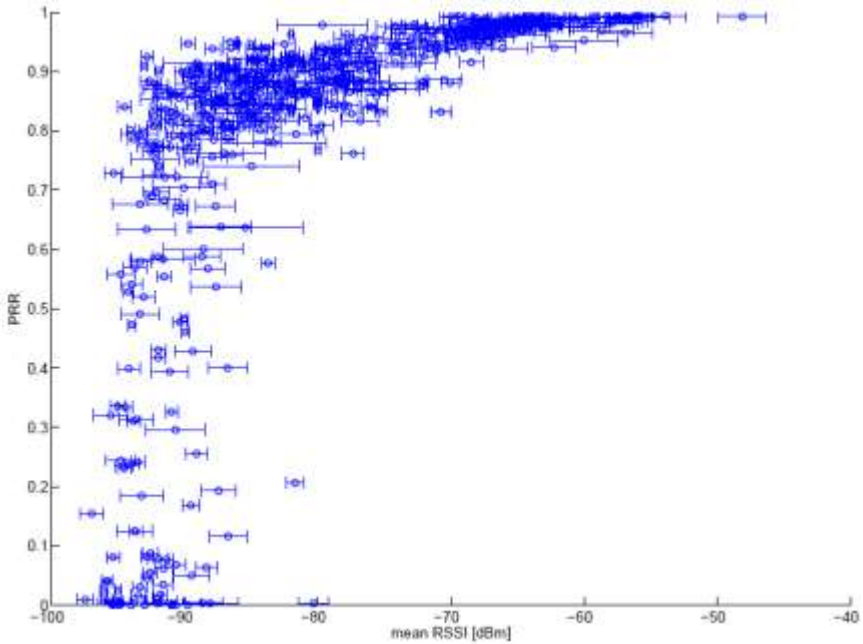
Low-Power Radios

➢ The CC2420 Radio – Hardware Indicators

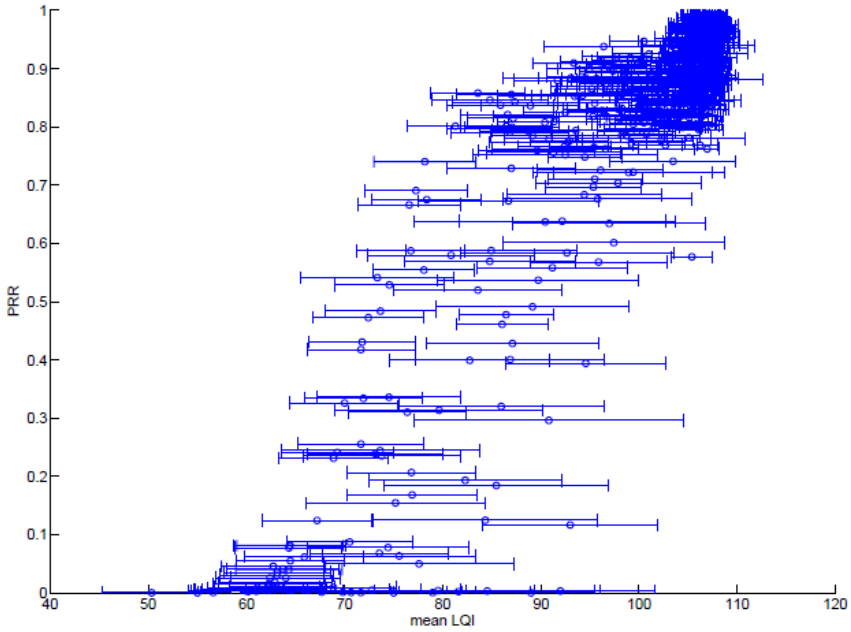
- The CC2420 provides two hardware indicators after the reception of a packet to judge link quality
- Link quality Indicator (LQI)
  - Values from 60 to 110
  - 60: poor link, 110: good link
  - Very good link have stable LQI values
- Received Signal Strength Indicator (RSSI)
  - The higher the RSSI, the stronger the signal
  - Values below -85 dBm might imply an unreliable link

Low-Power Radios

➢ The CC2420 Radio – Hardware Indicators



(RSSI vs. PRR)



(LQI vs. PRR)

Low-Power Radios

- **The CC2420 Radio – Hardware Indicators**
  - Link quality Indicator (LQI)
    - Get it after packet reception
    - extern uint8_t cc2420_last_correlation;
  - Received Signal Strength Indicator (RSSI)
    - Get it after packet reception
    - extern signed char cc2420_last_rssi;

You need to subtract -45 to the RSSI value
Check the CC2420 manual at page 47

Low-Power Radios

➢ The CC2420 Radio – Hardware Indicators

- RSSI in absence of packet transmissions (anytime)
  - RSSI noise floor
  - Energy detection
  - Often used for Clear Channel Assessment (default = -77 dBm)
  - Typical sensitivity threshold around -95 dBm
  - int cc2420_rssi(void)

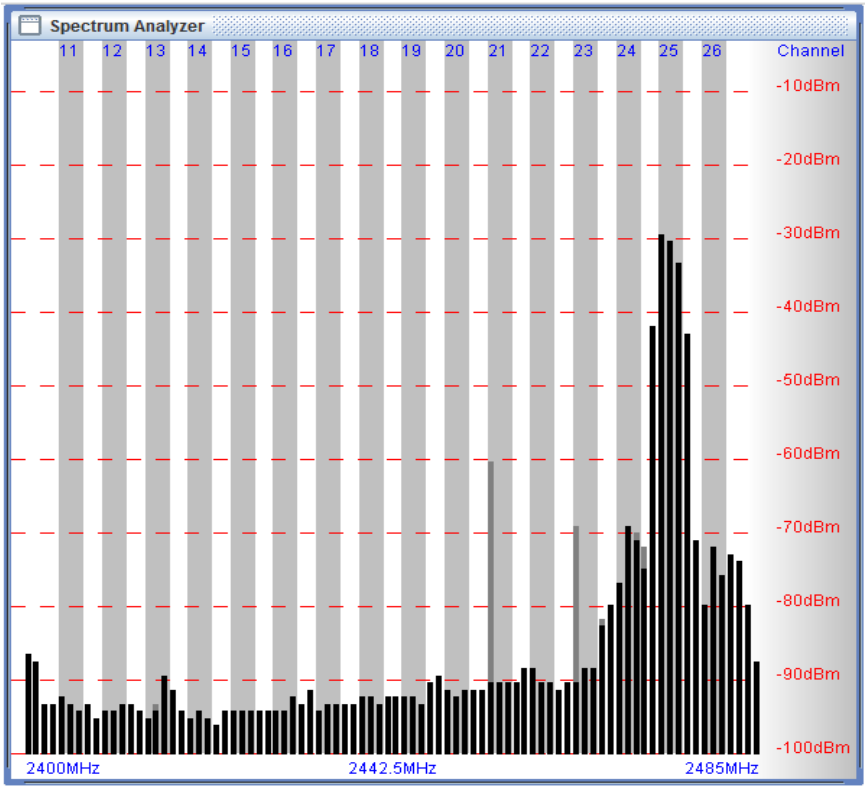
Subtract -45 to the RSSI value
Check the CC2420 manual at page 47

- Different nodes are calibrated differently
  - Might lead to different results

Low-Power Radios

➢ The CC2420 Radio – Hardware Indicators

- RSSI in absence of packet transmissions (anytime)



http://contikiprojects.svn.sourceforge.net/viewvc/contikiprojects/sics.se/rssi-scanner/

Low-Power Radios

➢ The CC2420 Radio – Distance Estimation using RSSI

- The signal strength is a function of the distance between sender and receiver
- A signal gets weaker as long as it travels through the air
  - Antenna gain and polarization, fading, shadowing, ...

$$P_r(d) \approx P_s \frac{1}{d^n}$$

P_r = received power at the antenna terminals

P_s = power at the sender side

d = distance

n = attenuation coefficient (*path loss exponent*)

Typically, $2 \leq n \leq 6$
N is higher in indoor environments, where absorption and reflections are higher

Low-Power Radios

➢ The CC2420 Radio – Distance Estimation using RSSI

- RSSI is not a precise method to estimate distance
- Readings might not be consistent
  - CC2420 has an accuracy of ±6dBm
- The RSSI indicator returned by the radio, can be defined as:

$$RSSI(d) = P_T - P_L(d_0) - 10 \cdot n \cdot \log\left(\frac{d}{d_0}\right) + K_0$$

P_T is the transmission power in dBm

$P_L(d_0)$ is the path-loss measured at a reference distance d_0

n is the path loss exponent.

K_0 represents shadowing and fading, which we will not consider

Typical path loss measured at a reference distance d_0 = 2 m is $P_L(d_0) \approx 46$ dBm.

Low-Power Radios

- **The CC2420 Radio – Hardware Indicators**
  - Signal Strength and LQI strongly depend on the environment
    - Obstacles (walls, hills, people, obstacles, height from ground)
    - Temperature, humidity, weather conditions

If interested: http://www.carloalbertoboano.com/documents/Boano_Cork_2010-01.zip

Carlo Alberto Boano, James Brown, Nicolas Tsiftes, Utz Roedig, and Thiemo Voigt. **The Impact of Temperature on Outdoor Industrial Sensornet Applications.** *In IEEE Transactions on Industrial Informatics, Volume 6, Number 3, pag. 451-459.* August 2010.

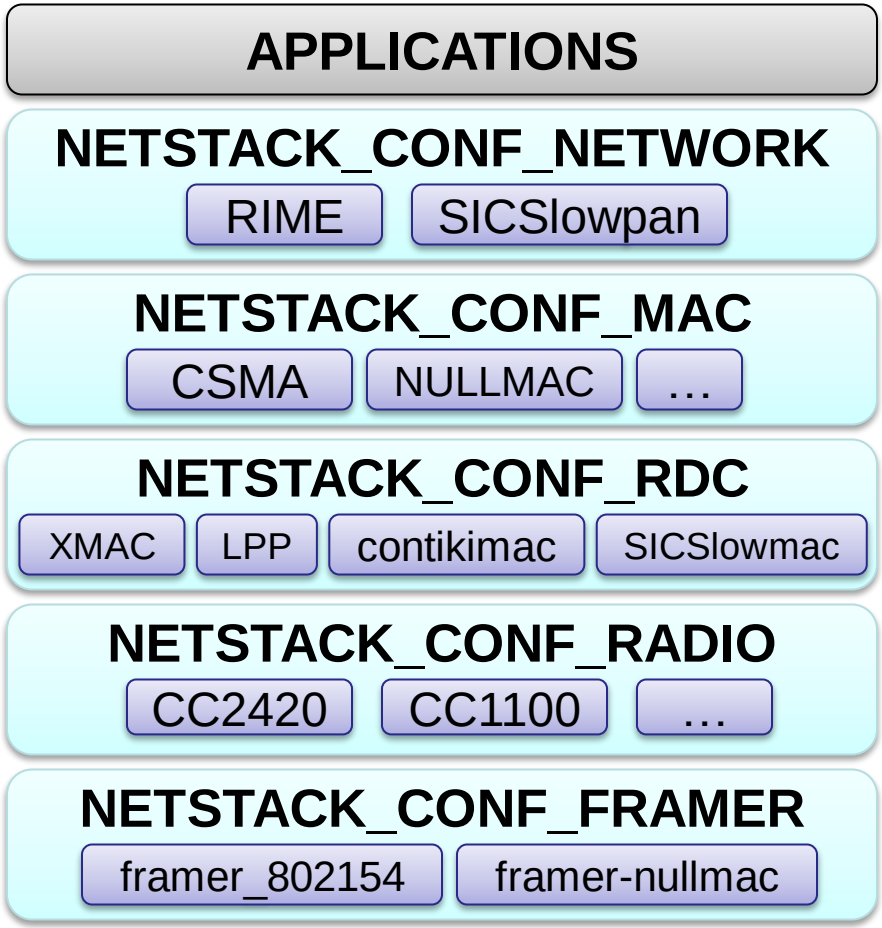
Contiki: Radio Communication

➢ Outline

- Wireless communications
  - Benefits
  - Drawbacks
  - IEEE 802.15.4
  - Interference and coexistence
- Low-power radios
  - TI CC2420
  - Link quality and link estimation
- Communicating using Contiki
  - The Rime Stack

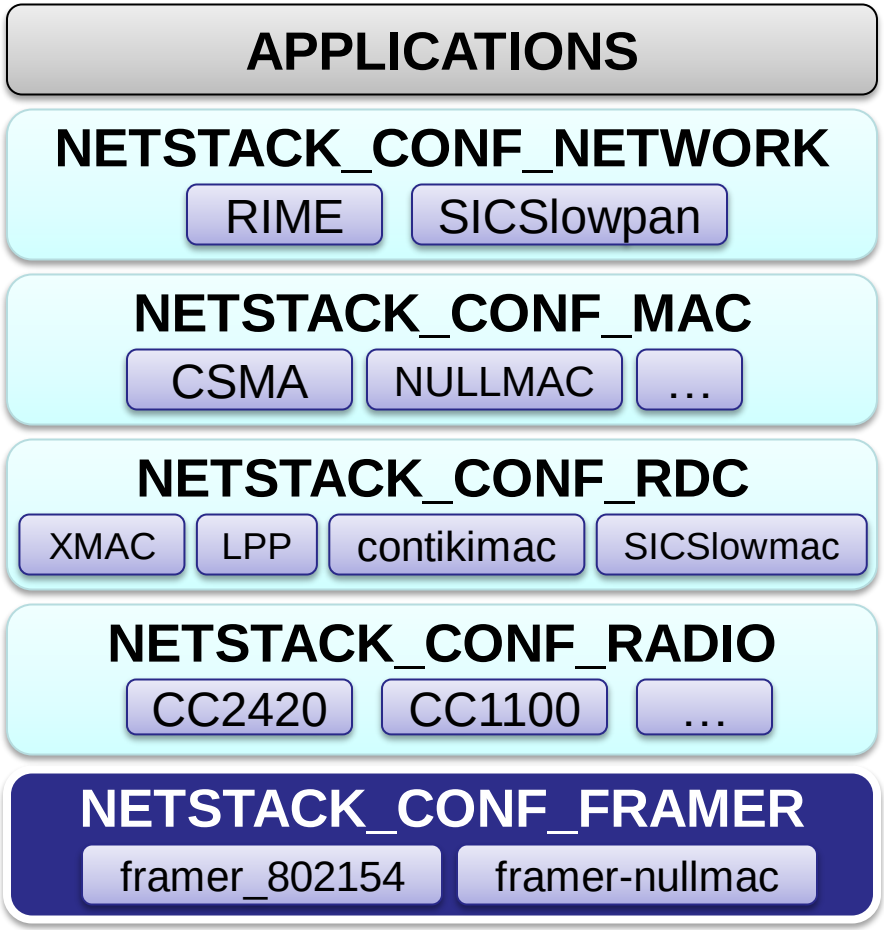
The Contiki Communication Stack

- **Stack of configurable layers**
  - Packet format conversion
    - NETSTACK_CONF_FRAMER
  - Radio driver protocol
    - NETSTACK_CONF_RADIO
  - Radio Duty-Cycling protocol
    - NETSTACK_CONF_RDC
  - MAC protocol
    - NETSTACK_CONF_MAC
  - Network stack
    - NETSTACK_CONF_NETWORK



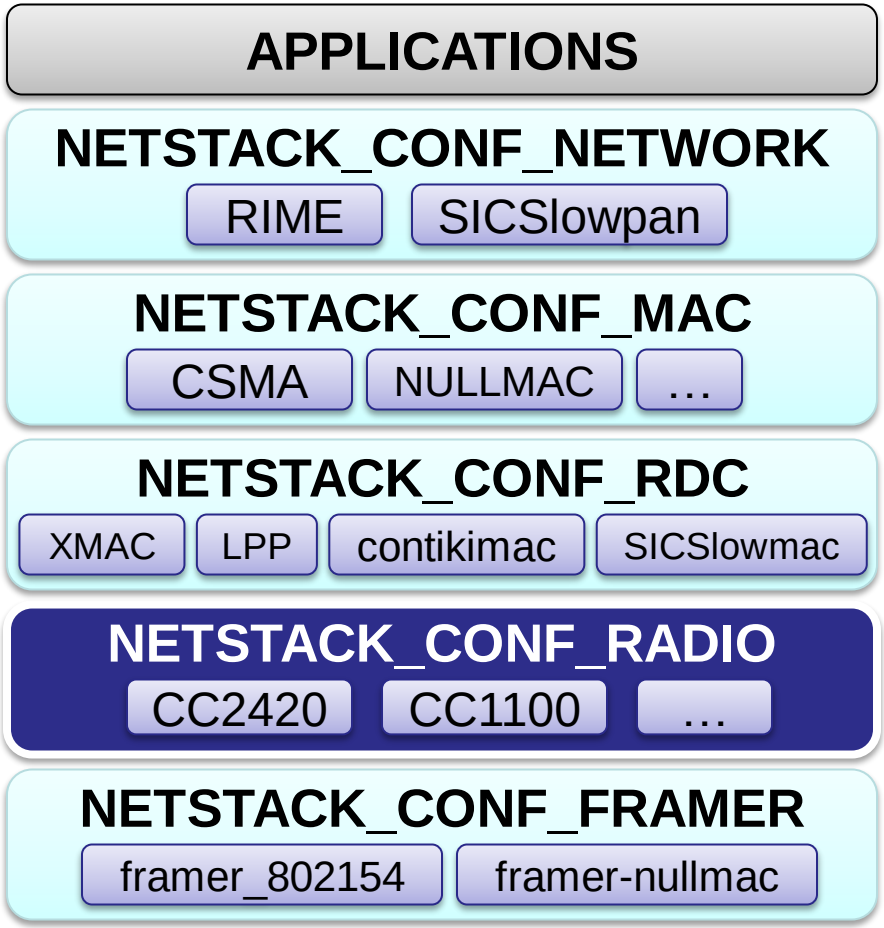
The Contiki Communication Stack

- **Stack of configurable layers**
  - Packet format conversion
    - NETSTACK_CONF_FRAMER
  - **NETSTACK_CONF_FRAMER**
    - framer_802154: IEEE 802.15.4
    - framer-nullmac: no added frames



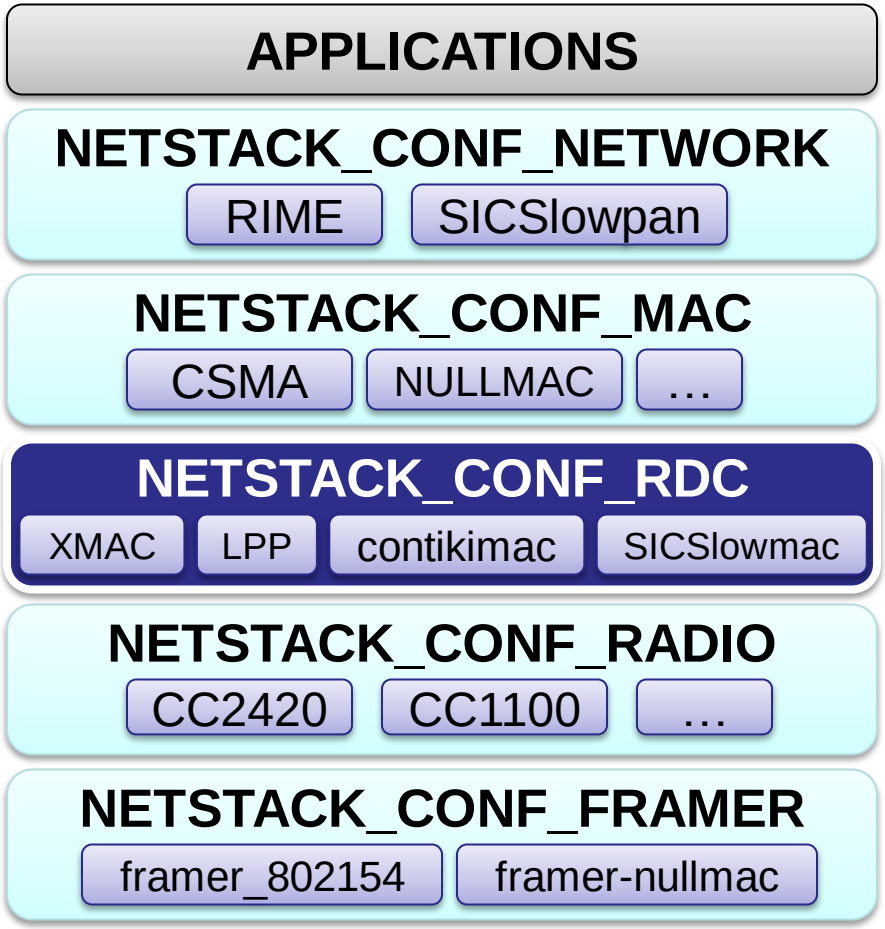
The Contiki Communication Stack

- **Stack of configurable layers**
  - Radio driver protocol
    - NETSTACK_CONF_RADIO
  - **NETSTACK_CONF_RADIO**
    - cc2420_driver
    - cc1100_driver
    - ...



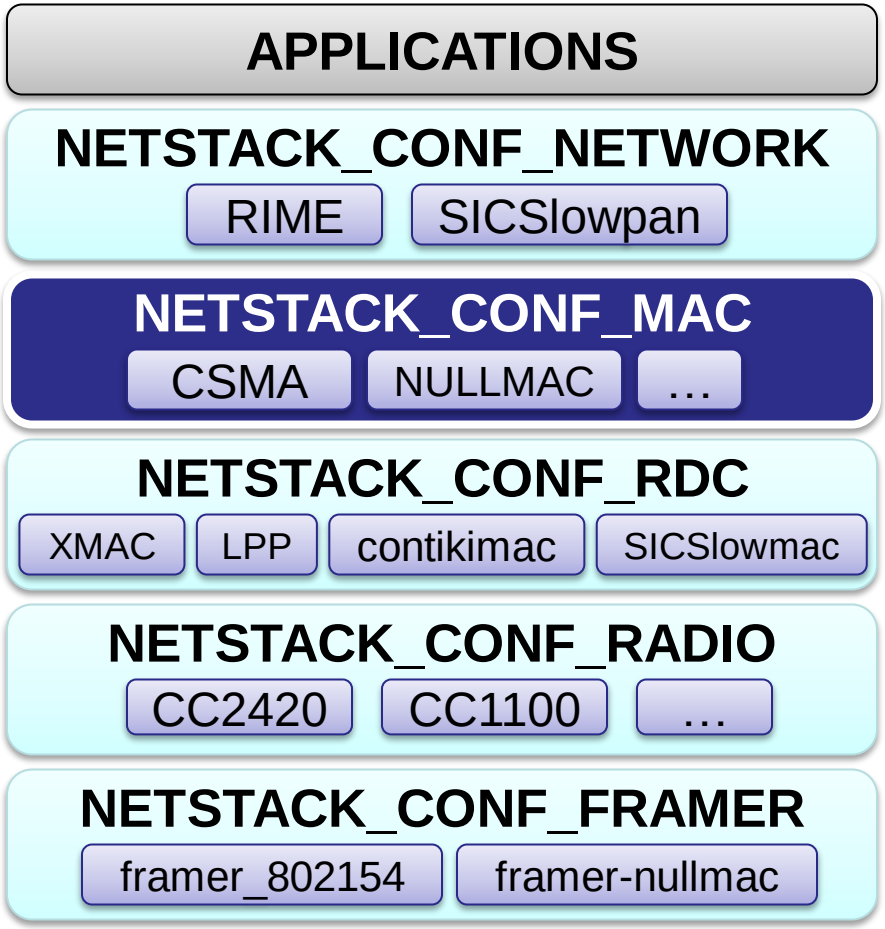
The Contiki Communication Stack

- **Stack of configurable layers**
  - Radio Duty-Cycling protocol
    - NETSTACK_CONF_RDC
  - **NETSTACK_CONF_RDC**
    - contikimac_driver
    - sicslowmac_driver: no MAC layer
    - lpp_driver: Low-Power Probing
    - xmac_driver: X-MAC



The Contiki Communication Stack

- **Stack of configurable layers**
  - MAC protocol
    - NETSTACK_CONF_MAC
  - **NETSTACK_CONF_MAC**
    - csma_driver: CSMA (CCA + queue)
    - nullmac: ALOHA



The Contiki Communication Stack

- **Several MAC protocols available in Contiki**
  - SICSlowMAC
    - No power saving features
    - Radio on all the time
  - ContikiMAC
    - Power-efficient MAC
  - X-MAC
    - Sender sending stobes
  - LPP (Low-power probing)
    - Receiver sending probes

How to change MAC layer in Contiki

Change the configuration parameters in:
contiki-2.x\platform\sky\contiki-conf.h
#define NETSTACK_CONF_* [for non-IPv6!!]**

Example: sicslowmac and nullmac as RDC and MAC to have no CCA and radio always on

```
/* Network setup for non-IPv6 (rime). */
#define NETSTACK_CONF_NETWORK rime_driver
#define NETSTACK_CONF_MAC nullmac_driver
#define NETSTACK_CONF_RDC sicslowmac_driver
#define NETSTACK_CONF_RADIO   cc2420_driver
#define NETSTACK_CONF_FRAMER  framer_802154
```

The Contiki Communication Stack

- **Several MAC protocols available in Contiki**

home ▸ Boano ▸ contiki-2.x ▸ platform ▸ z1 ▸ Search z1

Burn New folder

Name	Date modified	Type	Size
apps	11/02/2011 14:06	File folder	
dev	11/02/2011 14:06	File folder	
cfs-coffee-arch.h	10/02/2011 18:08	H File	4 KB
checkpoint-arch.c	10/02/2011 18:08	C File	10 KB
contiki-conf.h	10/02/2011 18:08	H File	7 KB
contiki-z1-main.c		C File	13 KB
Makefile.z1			
msp430.c			
node-id.c	10/02/		
node-id.h	10/02/		
platform-conf.h	10/02/		

Type: H File
Size: 6,00 KB
Date modified: 10/02/

contiki-conf.h

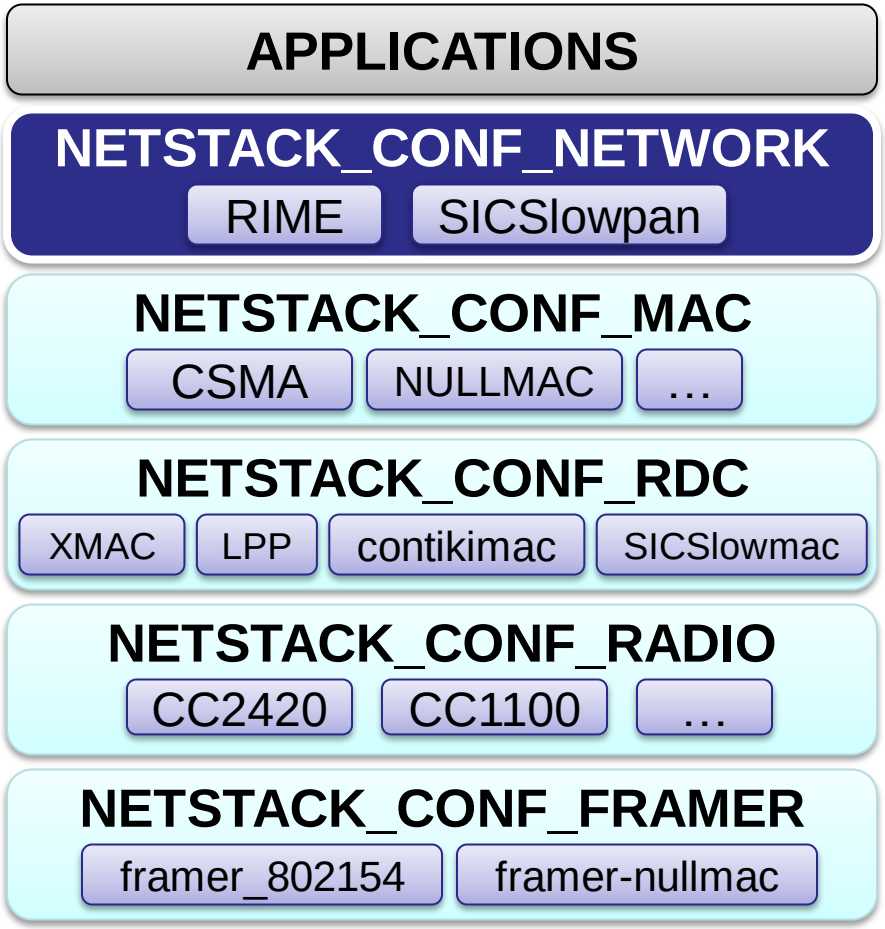
```
/* Network setup for non-IPv6 (rime). */

#define NETSTACK_CONF_NETWORK rime_driver
#define NETSTACK_CONF_MAC     csma_driver
#define NETSTACK_CONF_RDC     nullrdc_driver //contikimac_driver
#define NETSTACK_CONF_FRAMER  framer_802154

#define CC2420_CONF_AUTOACK              0
#define MAC_CONF_CHANNEL_CHECK_RATE      8
```

The Contiki Communication Stack

- **Stack of configurable layers**
  - Network stack
    - NETSTACK_CONF_NETWORK
  - **NETSTACK_CONF_NETWORK**
    - SICSlowpan: uIPv6
    - RIME: sensornet stack



The Contiki Communication Stack

- **The Rime stack**
  - Simple, lightweight, layered communication stack for wireless sensor networks
  - Much thinner layers than traditional architectures
  - Designed to simplify the implementation complexity of communication protocols

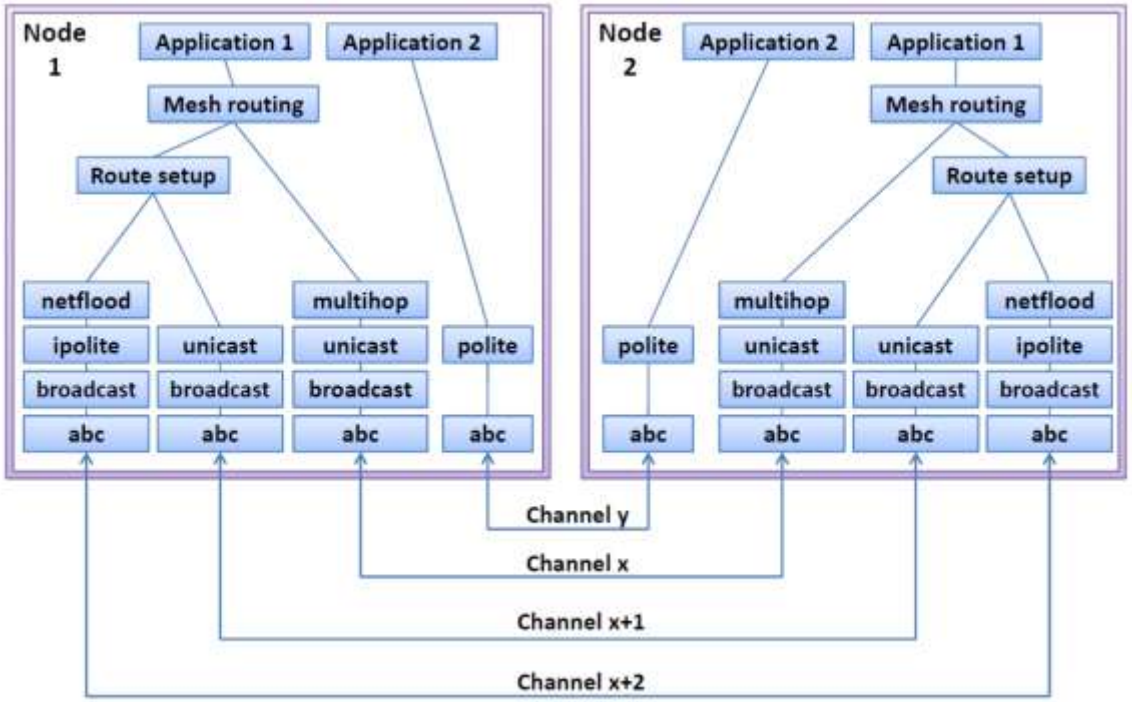
Rime frost: composed of many thin layers of ice

The Contiki Communication Stack

➢ The Rime Stack

- Programming model based on
  - Connections
  - Channels
  - Callbacks

"Sockets" for sensor networks



The Contiki Communication Stack

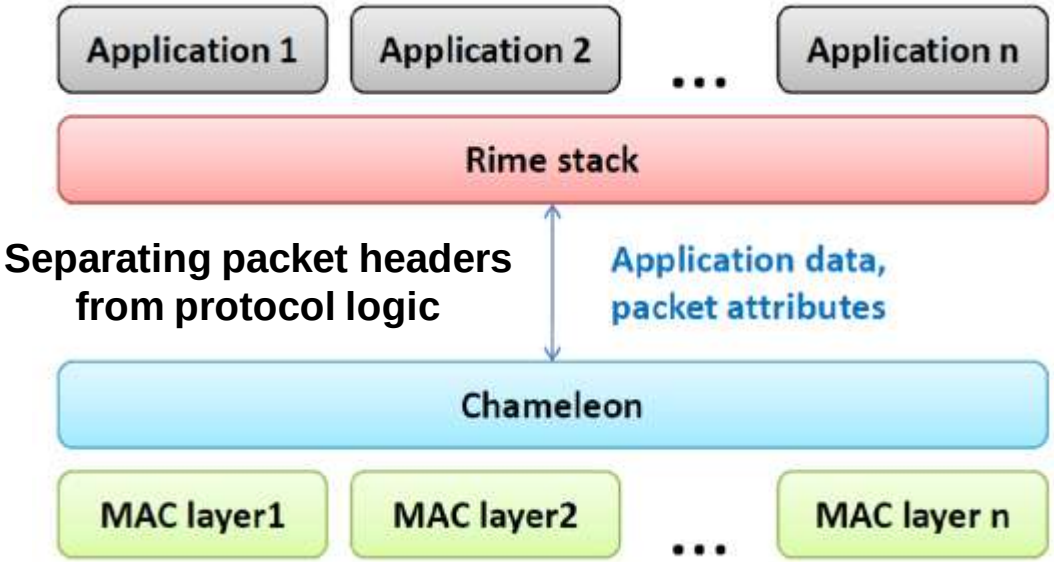
➢ The Rime Stack

- Connections
  - Must be opened before use
  - Arguments: module structure, channel, callbacks
- Channels identify all communications in Rime
  - 16-bit channel (different from the 802.15.4 physical channel)
  - Something like a port in IP
  - Communicating nodes must agree on what modules to use
  - Channels numbers < 128 are reserved by the system
- Callbacks
  - Called whenever a packet arrives, times out, or an error was raised

The Contiki Communication Stack

- **The Rime Stack**
  - Primitives from single-hop broadcast to reliable multi-hop flooding
    - Primitives built in terms of each other

- **Chameleon modules**
  - Separating the header construction and parsing from the rest
  - Bit-level headaches are removed from Rime



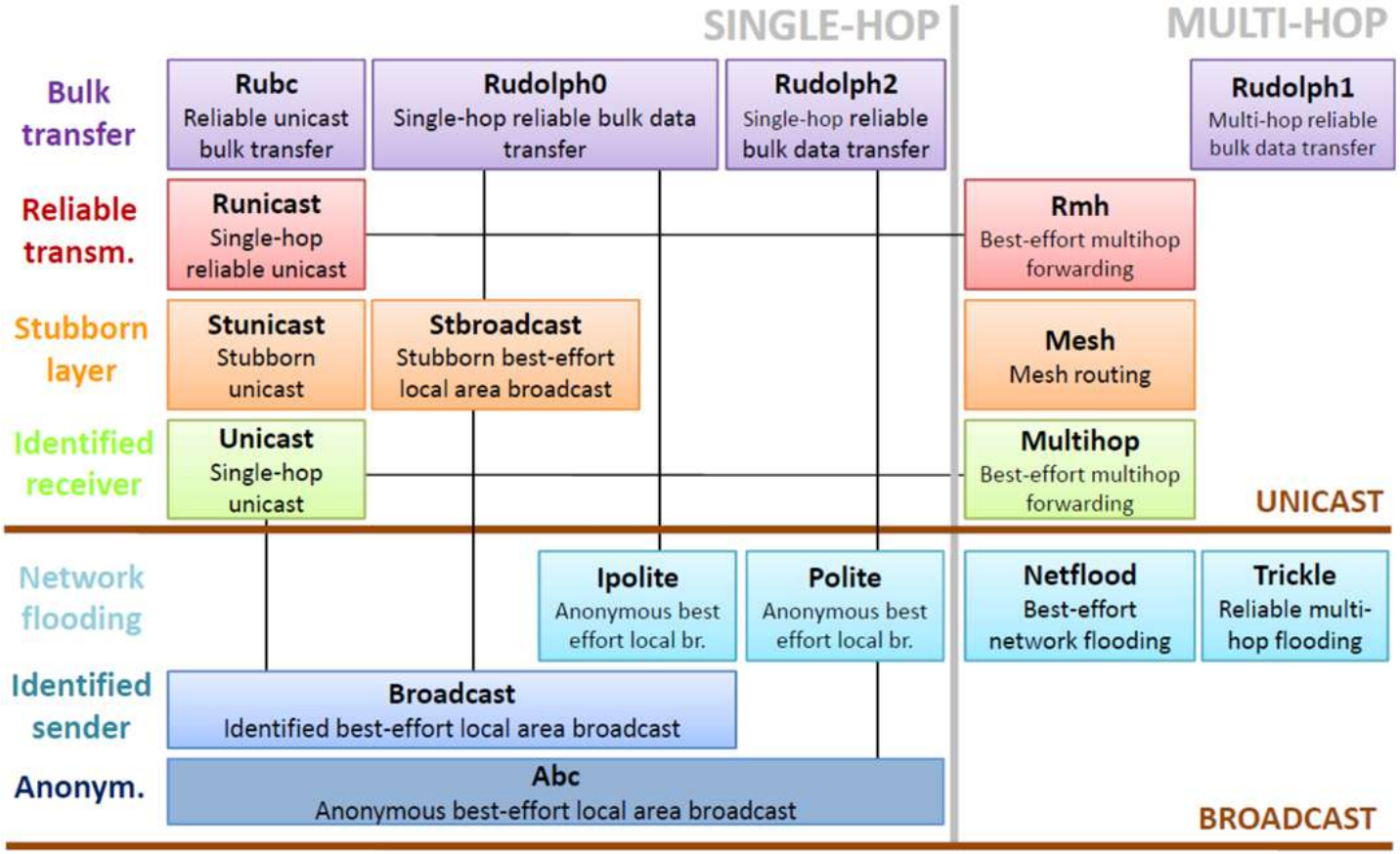
The Contiki Communication Stack

➢ The Rime Stack

Implements ACK and sequencing

Resends messages until the upper layer requests it

Each Rime layer is thin and simple!



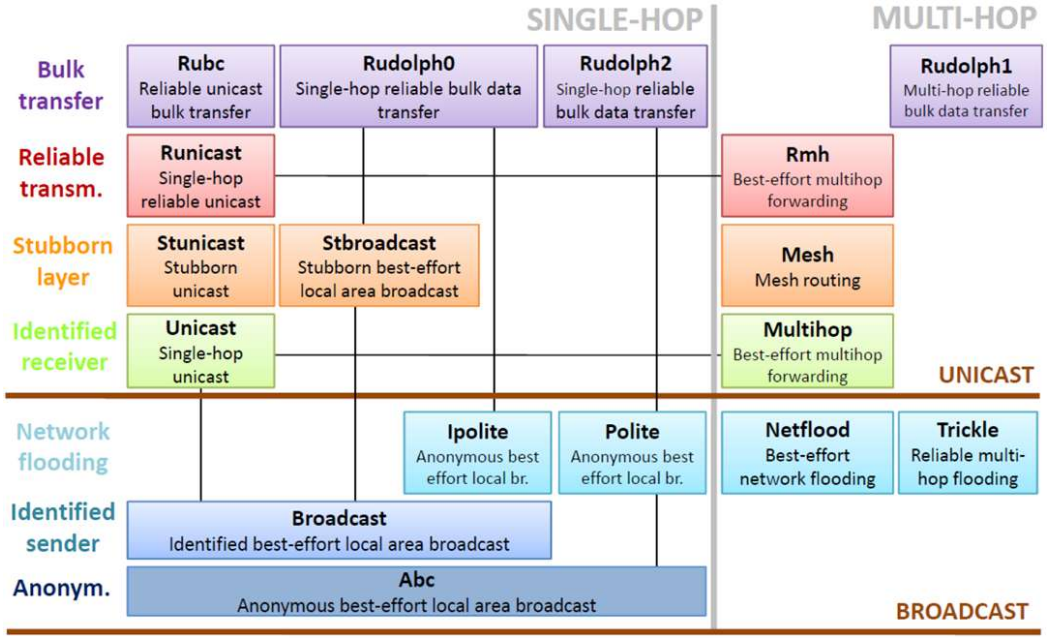
The Contiki Communication Stack

- **The Rime Stack**
  - Step by step
    - Decide the type of connection
    - Open a connection (define a channel and a callback)
    - Set the Exit handler
    - Write the callback function
  - Two simple examples
    - Sending and receiving a broadcast message
    - Sending and receiving a unicast message structure

The Contiki Communication Stack

➢ The Rime Stack

- Sending and receiving a broadcast message
  - Type of connection: broadcast



Check: contiki\core\rime\abc.h

The Contiki Communication Stack

➢ The Rime Stack

- Sending and receiving a broadcast message

```
#include "contiki.h"
#include "net/rime.h"
#include "dev/cc2420.h"
#include <stdio.h>
...
```

Include Rime and the Radio (cc2420)

Check: contiki\core\rime\abc.h

The Contiki Communication Stack

➢ The Rime Stack

- Sending and receiving a broadcast message

```
...
PROCESS(example_abc_process, "ABC example");
AUTOSTART_PROCESSES(&example_abc_process);
static voidabc_recv(struct abc_conn *c){
        printf("abc message received '%s'\n", (char *)packetbuf_dataptr());
}
static const struct abc_callbacks abc_call = {abc_recv};
static struct abc_conn abc;
PROCESS_THREAD(example_abc_process, ev, data)
{
  PROCESS_BEGIN();
  ...
```

Define a callback
(Must be done even if you are only planning to be a sender!)

Check: contiki\core\rime\abc.h

The Contiki Communication Stack

➢ The Rime Stack

- Sending and receiving a broadcast message

```
PROCESS_THREAD(example_abc_process, ev, data)
{
  PROCESS_BEGIN();
  abc_open(&abc, 128, &abc_call);
  while(1) {
      ...
  }
  PROCESS_END();
}
```

Open the broadcast connection
(Specify type, channel and callback)

- Type of connection: abc (anonymous broadcast)
- Logical Rime channel: 128
- Callback function: abc_call

Check: contiki\core\rime\abc.h

The Contiki Communication Stack

➢ The Rime Stack

- Sending and receiving a broadcast message

```
PROCESS_THREAD(example_abc_process, ev, data)
{
  PROCESS_EXITHANDLER(abc_close(&abc);)
  PROCESS_BEGIN();
  cc2420_set_txpower(POWER);
  cc2420_set_channel(CHANNEL);
  abc_open(&abc, 128, &abc_call);
  while(1) {
     …
  }
  PROCESS_END();
}
```

Physical channel of the radio

Rime logical channel

Check: contiki\core\rime\abc.h

The Contiki Communication Stack

➢ The Rime Stack

- Sending and receiving a broadcast message

```
PROCESS_THREAD(example_abc_process, ev, data)
{
  PROCESS_EXITHANDLER(abc_close(&abc);)
  PROCESS_BEGIN();
  cc2420_set_txpower(POWER);
  cc2420_set_channel(CHANNEL);
  abc_open(&abc, 128, &abc_call);
  while(1) {
      …
  }
  PROCESS_END();
}
```

Set the Exit Handler
(Cleanup)

Check: contiki\core\rime\abc.h

The Contiki Communication Stack

➢ The Rime Stack

- Sending and receiving a broadcast message

```
PROCESS_THREAD(example_abc_process, ev, data)
{
  PROCESS_EXITHANDLER(abc_close(&abc);)
  PROCESS_BEGIN();
  static struct etimer et;
  cc2420_set_txpower(POWER);
  cc2420_set_channel(CHANNEL);
  abc_open(&abc, 128, &abc_call);
  while(1) {
      etimer_set(&et, CLOCK_SECOND);
      PROCESS_WAIT_EVENT_UNTIL(etimer_expired(&et));
      …
  }
  PROCESS_END();
}
```

Sending a message every second

Check: contiki\core\rime\abc.h

The Contiki Communication Stack

➢ The Rime Stack

- Sending and receiving a broadcast message

```
PROCESS_THREAD(example_abc_process, ev, data)
{
  PROCESS_EXITHANDLER(abc_close(&abc);)
  PROCESS_BEGIN();
  static struct etimer et;
  abc_open(&abc, 128, &abc_call);
  while(1) {
      etimer_set(&et, CLOCK_SECOND);
      PROCESS_WAIT_EVENT_UNTIL(etimer_expired(&et));
      packetbuf_copyfrom("Hello", 6);
      abc_send(&abc);
      printf("abc message sent\n");
  }
  PROCESS_END();
}
```

Actually sending the packet
(String)

```
char s[30];
sprintf(s,"String with number %d\n", number);
packetbuf_copyfrom(s, 30);
```

Check: contiki\core\rime\abc.h

The Contiki Communication Stack

- **The Rime Stack**
  - Example

contiki
examples
rime
example-abc.c

example-abc.c | example-unicast.c

```c
#include "contiki.h"
#include "net/rime.h"
#include "random.h"
#include "dev/button-sensor.h"
#include "dev/leds.h"
#include <stdio.h>

PROCESS(example_abc_process, "ABC example");
AUTOSTART_PROCESSES(&example_abc_process);

static void abc_recv(struct abc_conn *c) {
  printf("abc message received '%s'\n", (char *)packetbuf_dataptr());
}
static const struct abc_callbacks abc_call = {abc_recv};
static struct abc_conn abc;

PROCESS_THREAD(example_abc_process, ev, data)
{
  PROCESS_EXITHANDLER(abc_close(&abc);)
  PROCESS_BEGIN();

  static struct etimer et;
  cc2420_set_txpower(31);
  cc2420_set_channel(26);
  abc_open(&abc, 128, &abc_call);

  while(1) {
    etimer_set(&et, CLOCK_SECOND * 2 + random_rand() % (CLOCK_SECOND * 2));
    PROCESS_WAIT_EVENT_UNTIL(etimer_expired(&et));

    packetbuf_copyfrom("Hello", 6);
    abc_send(&abc);
    printf("abc message sent\n");
  }

  PROCESS_END();
}
```

The Contiki Communication Stack

- **The Rime Stack**
  - What about addressing a specific device?
    - I want to talk with sensor node X

- **Node ID**
  - Knowing its own node id
  - Important to have a single program for all nodes
    - Specify in the same code different behaviors depending on the node
    - You might want to differentiate between sender and receiver
    - Either you write two different programs (sender.c and receiver.c)
    - Or you write a single program and differentiate on the node id

The Contiki Communication Stack

Node ID

- Example

```
...
#include "node-id.h"
...
printf("Node id: %d\n",node_id);
...
if(node_id == XXX) {
   ...
}
else {
   ...
}
...
```

```
Boano@PC-Boano ~
$ ./contiki-2.x/tools/sky/serialdump.exe -t -b115200 /dev/com15
connecting to /dev/com15 (115200) [OK]
1269442461'525|Rime started with address 153.37 channel 26
1269442461'532|MAC 99:25:00:00:00:00:00:00 Contiki 2.4 started.
1269442461'565|nullmac sicslowmac, channel check rate 128 Hz
1269442461'584|Starting 'Hello process'
```

Check: contiki\platform\sky\node-id.h

The Contiki Communication Stack

➢ Burning the node-id

- make z1-motelist

```
user@instant-contiki: ~/contiki-2.x/examples/z1
File  Edit  View  Terminal  Help
user@instant-contiki:~/contiki-2.x/examples/z1$ make z1-motelist
../../tools/z1/motelist-z1
Reference  Device           Description
---------- ---------------- ------------------------------------------
Z1RC0336   /dev/ttyUSB0     Silicon Labs Zolertia Z1
user@instant-contiki:~/contiki-2.x/examples/z1$
```

- Take the reference ID and type:

make clean

make burn-nodeid.upload nodeid=XXX nodemac=XXX && make login

```
user@instant-contiki: ~/contiki-2.x/examples/z1
File  Edit  View  Terminal  Help
user@instant-contiki:~/contiki-2.x/examples/z1$ make z1-motelist
../../tools/z1/motelist-z1
Reference  Device           Description
---------- ---------------- ------------------------------------------
Z1RC0336   /dev/ttyUSB0     Silicon Labs Zolertia Z1
user@instant-contiki:~/contiki-2.x/examples/z1$ make burn-nodeid.upload nodeid=336 no
demac=336 && make login
```

The Contiki Communication Stack

➢ Unicast communication

- Example: message structure

```
#include "node-id.h"
#include "random.h"
#include "cc2420.h"
#include "net/rime.h"

// Define the address of Sender and Receiver
#define SENDER 215
#define RECEIVER 216
```

ADDRESSES of the sensor nodes:
SENDER: 215.0
RECEIVER: 216.0

The Contiki Communication Stack

➢ Unicast communication

- Example: message structure

```
#include "node-id.h"
#include "random.h"
#include "cc2420.h"
#include "net/rime.h"

// Define the address of Sender and Receiver
#define SENDER 215
#define RECEIVER 216

// Structure of the message to be sent
struct message{
        int id;
        int field1;
};
...
```

Defining the message structure

The Contiki Communication Stack

➢ Unicast communication

- Example: unicast callback (function when a packet is received)

```
// Unicast callback
static struct unicast_conn uc;
static void recv_uc(struct unicast_conn *c, const rimeaddr_t *from)
{
 // Retrieving data from the buffer
 struct message msg;
 memcpy(&msg, packetbuf_dataptr(), sizeof(msg));
 printf("Received message from %d:%d: id %d, and value %d\n", from->u8[0], from->u8[1], msg.id, msg.field1);
}
static const struct unicast_callbacks unicast_callbacks = {recv_uc};

...
```

Address of the sender:
from->u8[0] and from->u8[1]
Example: node id 215.0: u8[0] = 215, u8[1] = 0

The Contiki Communication Stack

➢ Unicast communication

- Example: unicast transmission (function to send a message)

```
// Sending a message
void sending packet(int id, int value, int destination) {

 struct message mesg;
 mesg.id = id;
 mesg.field1 = value;

 ...

}
```

Creating a message and fill in the structure

The Contiki Communication Stack

➢ Unicast communication

- Example: unicast transmission

```
// Sending a message
void sending packet(int id, int value, int destination) {

 struct message mesg;
 mesg.id = id;
 mesg.field1 = value;

 packetbuf_clear();
 packetbuf_copyfrom(&mesg, sizeof(struct message));

 ...

}
```

Copying the message to the buffer

The Contiki Communication Stack

➢ Unicast communication

- Example: unicast transmission

```
// Sending a message
void sending packet(int id, int value, int destination) {

 struct message mesg;
 mesg.id = id;
 mesg.field1 = value;

 packetbuf_clear();
 packetbuf_copyfrom(&mesg, sizeof(struct message));

 rimeaddr_t addr;
 addr.u8[0] = destination;
 addr.u8[1] = 0;

 ...
}
```

Setting the destination address

The Contiki Communication Stack

➢ Unicast communication

- Example: unicast transmission

```
// Sending a message
void sending packet(int id, int value, int destination) {

 struct message mesg;
 mesg.id = id;
 mesg.field1 = value;

 packetbuf_clear();
 packetbuf_copyfrom(&mesg, sizeof(struct message));

 rimeaddr_t addr;
 addr.u8[0] = destination;
 addr.u8[1] = 0;

 unicast_send(&uc, &addr);
}
```

Sending the packet

The Contiki Communication Stack

➢ Unicast communication

- Example: unicast transmission

```
...
PROCESS_EXITHANDLER(unicast_close(&uc);)
PROCESS_BEGIN();
unicast_open(&uc, 200, &unicast_callbacks);
while(1) {
        ...
}
...
```

Exit handler

Opening the unicast connection

The Contiki Communication Stack

➢ Unicast communication

- Example: unicast transmission

```
...
PROCESS_EXITHANDLER(unicast_close(&uc);)
PROCESS_BEGIN();
unicast_open(&uc, 200, &unicast_callbacks);
while(1) {
        if(node_id == SENDER){
                ...
        }
        else{
                ...
        }
}
...
```

Checking the node id

I am the sender

I am the receiver

The Contiki Communication Stack

➢ Unicast communication

- Example: unicast transmission

```
...
PROCESS_EXITHANDLER(unicast_close(&uc);)
PROCESS_BEGIN();
unicast_open(&uc, 200, &unicast_callbacks);
while(1) {
        if(node_id == SENDER){
                etimer_set(&et, CLOCK_SECOND);
                PROCESS_WAIT_EVENT_UNTIL(etimer_expired(&et));
                int new_value = random_rand() % 100;
                printf("Sending packet with ID %d and value %d\n",counter,new_value);
                sending_packet(counter++, new_value, RECEIVER);
        }
        else{
                ...
        }
}
```

The sender sends messages periodically once every second

The Contiki Communication Stack

➢ Unicast communication

- Example: unicast transmission

```
...
PROCESS_EXITHANDLER(unicast_close(&uc);)
PROCESS_BEGIN();
unicast_open(&uc, 200, &unicast_callbacks);
while(1) {
        if(node_id == SENDER){
                etimer_set(&et, CLOCK_SECOND);
                PROCESS_WAIT_EVENT_UNTIL(etimer_expired(&et));
                int new_value = random_rand() % 100;
                printf("Sending packet with ID %d and value %d\n",counter,new_value);
                sending_packet(counter++, new_value, RECEIVER);
        }
        else{
                PROCESS_WAIT_EVENT();
        }
}
```

The receiver just waits for events!

The Contiki Communication Stack

➢ Unicast communication

- Example: output of the program

```
Boano@PC-Boano ~
$ ./contiki-2.x/tools/sky/serialdump.exe -t -b115200 /dev/com10
connecting to /dev/com10 (115200) [OK]
1271674553'977|Rime started with address 215.0
1271674553'983|MAC d7:00:00:00:00:00:00:00 Contiki 2.4 started. Node id is set to 215.
1271674554'007|nullmac sicslowmac, channel check rate 128 Hz, radio channel 26
1271674554'024|Starting 'Unicast example'
1271674554'992|Sending packet with ID 0 and value 49
1271674555'996|Sending packet with ID 1 and value 54
1271674556'991|Sending packet with ID 2 and value 31
```

Node 215.0
SENDER

```
Boano@PC-Boano ~
$ ./contiki-2.x/tools/sky/serialdump.exe -t -b115200 /dev/com11
connecting to /dev/com11 (115200) [OK]
1271674554'773|Rime started with address 216.0
1271674554'780|MAC d8:00:00:00:00:00:00:00 Contiki 2.4 started. Node id is set to 216.
1271674554'821|nullmac sicslowmac, channel check rate 128 Hz, radio channel 26
1271674554'840|Starting 'Unicast example'
1271674554'847|Waiting for packets
1271674554'996|Received a message from host (215:0): id 0, and value 49
1271674555'995|Received a message from host (215:0): id 1, and value 54
1271674557'006|Received a message from host (215:0): id 2, and value 31
```

Node 216.0
RECEIVER

Afternoon Lab Session

- **Remember: steps needed to create your application**

1. Create a folder
2. Create C file and makefile (or paste existing ones)
3. Edit the makefile and set the names accordingly
   - TARGET plaftorm
   - Project name as the name of the C file
   - Upload
4. Edit the C file
5. Compile and upload!
6. Connect to the serial port with make z1-reset && make login

Afternoon Lab Session

- **Open a terminal and go to the home folder**
- **mv contiki-2.x/ contiki-2.old**
- **sudo apt-get install git-core**
- **git clone git://github.com/enricmcalvo/contiki-2.x**
  - Update Contiki folder from Enric (bugfixes, examples…)
- **From contiki-2.x do git pull to update the repo**
  - To update the folder anytime

Afternoon Lab Session

- **sudo apt-get install openjdk-6-jdk**
  - To install Java

Afternoon Lab Session

- **Exercise 1**
  - Run the rssi-scanner
  - contiki/examples/z1/rssi_scanner
  - make upload
  - make viewrssi
  - make viewrssi3d

Afternoon Lab Session

- **Exercise 2**
  - Whenever the user button is pressed…
  - Send a packet containing your name
  - Unicast packet to host 200.0
  - Unicast connection on channel 133

Afternoon Lab Session

- **Exercise 3**
  - git pull
  - contiki-2.x\examples\z1\example2
  - make example-unicast-temp.upload
  - In this way, whenever the user button is pressed you send me the temperature…

Afternoon Lab Session

Exercise 4 - Competition

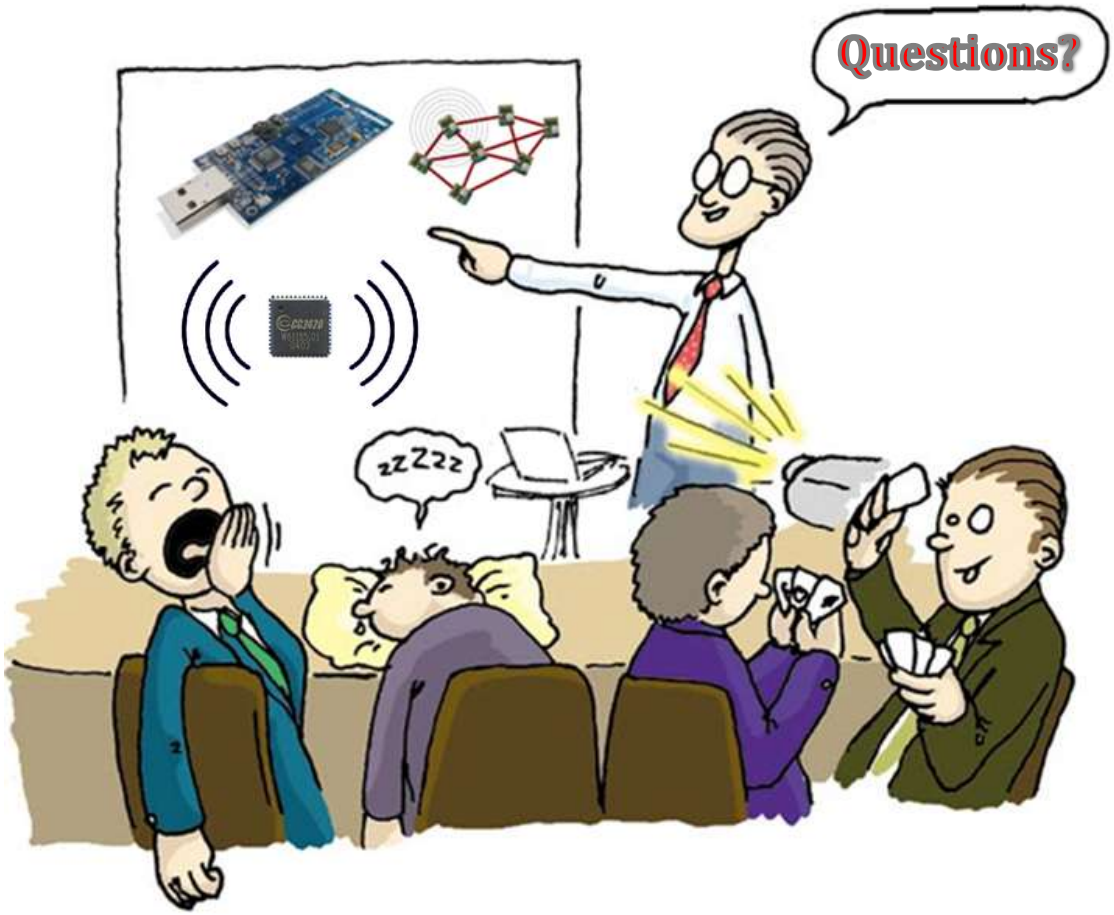
- I am transmitting non-anonymous broadcast packets (rime channel 197) in one of the 16 IEEE 802.15.4 channel in the 2.4 GHz frequency
- Your task:
  1. Find in which PHY channel I am transmitting
  2. Find what I am transmitting (which string)
  3. Be the first group to solve the task (prize!)

References

Contiki and Rime

- Rime: a lightweight layered communication stack for sensor networks
  http://www.sics.se/~adam/dunkels07rime.pdf
- Official website
  http://www.sics.se/contiki

CC2420

- Datasheet
  http://www-mtl.mit.edu/Courses/6.111/labkit/datasheets/CC2420.pdf

Questions & Answers