Introduction to Wireless Sensor Networks: Networking Aspects

Nancy Panousopoulou
Electrical and Computer Engineer, PhD
Signal Processing Lab, ICS-FORTH
apanouso@ics.forth.gr

Outline

• Part 1: Applications, Standards and Protocols
  • Introduction & Reasoning of Existence
  • Sensing, Processing, and Networking Aspects
  • Standards, Topologies & Protocols

• Part 2: WSN Programming
  • WSN Core and types of nodes
  • Real-time Operating Systems
  • Examples & Hands on Session

- sensor
  (transducer, measuring a physical phenomenon e.g. heat, light, motion, vibration, and sound)

- processing, storage
  (communication with sensor, data acquisition, and preprocessing, buffers handling, etc)

- power unit
  (battery based – limited lifetime!)

- transceiver
  (connection to the outer-world, e.g. other sensor nodes, or data collectors – sinks)
WSN Core

Typical State Machine of a Sensor Node

- **Standby / sleep**: Wake-up(SENSOR) → sensing
  - Wake-up(NET) → Networking
  - Network Ready (DONE) → processing
- **sensing**: Data Ready (DONE) → Sensor Ready (PROCESS)
- **Networking**: Wake-up(NET) → networking
  - Network Ready (COMMUNE) → networking
  - Network Ready (PROCESS) → processing
- **processing**: Data Ready(SEND) → processing
  - Data Ready (SENSE) → sensing
<table>
<thead>
<tr>
<th>Family</th>
<th>TRX</th>
<th>Processor</th>
<th>Memory</th>
<th>On-board Sensors</th>
<th>Expandability</th>
<th>Notes &amp; Application areas</th>
</tr>
</thead>
<tbody>
<tr>
<td>TELOS8</td>
<td><strong>TI <a href="mailto:802.15.4@2.4GHz">802.15.4@2.4GHz</a></strong></td>
<td>Ti msp430-F1 (16-bit)</td>
<td>10KB RAM, 48KB Flash</td>
<td>Temperature, Humidity, Light</td>
<td>10 GIOs, USB programming interface</td>
<td>Open platform. Environmental and health structural monitoring. PoC research projects</td>
</tr>
<tr>
<td>Mica2</td>
<td><strong>TI 802.15.4@868MHz</strong></td>
<td>ATMEL AVR 128L (16-bit)</td>
<td>4KB RAM/48 KB Flash</td>
<td>-</td>
<td>Dedicated environmental sensor board 51-pin expansion, RS232.</td>
<td>One of the oldest platforms. Environmental and health structural monitoring. PoC research projects. Open source software support – Active (?)</td>
</tr>
<tr>
<td>Mica2</td>
<td><strong>TI <a href="mailto:802.15.4@2.4GHz">802.15.4@2.4GHz</a></strong></td>
<td>ATMEL AVR 128 (16-bit)</td>
<td>4KB RAM/48 KB Flash</td>
<td>-</td>
<td>Dedicated environmental sensor board 51-pin expansion.</td>
<td>Environmental and health structural monitoring. PoC research projects. Open source software support – Active (?) Dipole Antenna</td>
</tr>
<tr>
<td>IRIS</td>
<td>ATMEL <strong><a href="mailto:802.15.4@2.4GHz">802.15.4@2.4GHz</a></strong></td>
<td>ATMEL AVR 1281</td>
<td>8KB RAM/48 KB Flash</td>
<td>-</td>
<td>Dedicated environmental sensor board 51-pin expansion.</td>
<td>Environmental and health structural monitoring. PoC research projects. Open source software support – Active. Dipole Antenna</td>
</tr>
<tr>
<td>Shimmer</td>
<td><strong>TI <a href="mailto:802.15.4@2.4GHz">802.15.4@2.4GHz</a></strong></td>
<td>Ti msp430-F1 (16-bit)</td>
<td>10 KB RAM, 48 KB Flash, 2GB μSD</td>
<td>3-axis accelerometer, Tilt &amp; vibration</td>
<td>Expandability for Accelerometers and ECG, EMG, USB mother board.</td>
<td>Research platform with commercial support. Excellent support (open source tools &amp; customized applications). Healthcare and Sports projects (wearable computing) Active and expanding. Rechargeable battery (up to 8 hours in fully functional mode)</td>
</tr>
<tr>
<td>SUNSPOT</td>
<td><strong>TI <a href="mailto:802.15.4@2.4GHz">802.15.4@2.4GHz</a></strong></td>
<td>ATMEL ARM (32-bit)</td>
<td>1 MB RAM, 8 MB Flash</td>
<td>3-axis accelerometer, 3-color light.</td>
<td>USB. 4 GIOs.</td>
<td>Open platform. JVM (very easy to program). Emulator is also available. Fancy platform with demos for audience with no related background. Active. For hobbyists 😊 Built in Li Battery</td>
</tr>
<tr>
<td>Zolertia Z1</td>
<td><strong>TI <a href="mailto:802.15.4@2.4GHz">802.15.4@2.4GHz</a></strong></td>
<td>Ti msp430-F2</td>
<td>8K RAM, 92KB Flash</td>
<td>3-axis accelerometer, temperature</td>
<td>52-pin expansion board. Open source community support &amp; commercial support (excellent Wiki)</td>
<td>All WSN-related. One of the latest platforms. Allows the option for a dipole antenna.</td>
</tr>
<tr>
<td>XM1000</td>
<td><strong>TI <a href="mailto:802.15.4@2.4GHz">802.15.4@2.4GHz</a></strong></td>
<td>Ti msp430-F2</td>
<td>8K RAM, 116 Flash, 1MB External Flash</td>
<td>Temperature, Humidity, Light</td>
<td>10 GIOs, USB programming interface</td>
<td>from a family of open platforms…. SMA connection (dipole antenna)… All WSN-related, perhaps not for healthcare (bulky size and design). Can last up to 3 weeks on low data rate (per minute).</td>
</tr>
</tbody>
</table>

The WSN Core – technologies and platforms...[1-6]
<table>
<thead>
<tr>
<th>Family</th>
<th>TRX</th>
<th>µProcessor</th>
<th>Memory</th>
<th>On-board Sensors</th>
<th>Expandability, Usability &amp; Support</th>
<th>Notes &amp; Application areas</th>
</tr>
</thead>
<tbody>
<tr>
<td>Firefly</td>
<td>AT Mega128RFA1 (SoC)</td>
<td><a href="mailto:802.15.4@2.4GHz">802.15.4@2.4GHz</a> (functional PHY, MAC compatible)</td>
<td>8 KB RAM, 128 KB Flash</td>
<td>-</td>
<td>Dedicated environmental sensor board (inc. audio, barometric pressure, PIR sensor, liquid / relay switch). + GPIOs</td>
<td>Research platform (CMU). Not as popular as other platforms. (?)</td>
</tr>
<tr>
<td>WiS mote</td>
<td>TI <a href="mailto:802.15.4@2.4GHz">802.15.4@2.4GHz</a> (functional PHY, MAC compatible)</td>
<td><a href="mailto:802.15.4@2.4GHz">802.15.4@2.4GHz</a> (functional PHY, MAC compatible)</td>
<td>16 KB RAM, 128 Flash</td>
<td>3-axis accelerometer, temperature, light.</td>
<td>8 Analog, 16 GPIO, mini USB</td>
<td>Optional support for Power-Line Communications and RS-485 (candidate for homes automation and industrial monitoring.) Research, open platform.</td>
</tr>
<tr>
<td>Xbee</td>
<td>Digi 868 / 2.4GHz (SoC)</td>
<td>Needs (mother board)</td>
<td>-</td>
<td>Serial communication (to µController) or host SCB (arduino, raspbery etc)</td>
<td>Provide wireless end-point connectivity to devices -&gt; plug-and-play. AT Commands for accessing the board. OTAP. 802.15.4 on HW</td>
<td></td>
</tr>
<tr>
<td>WaspMote</td>
<td>XBee-15.4. / ZigBee</td>
<td>ATME AVR 1281</td>
<td>8 KB RAM, 128 KB Flash, 2GB µSD</td>
<td>3-axis accelerometer, temperature.</td>
<td>Analog, Digital, USB, I2C</td>
<td>Built in a torrent style – highly customizable w.r.t. the application needs. GPS optional. Commercial product – for commercial and very applied projects. OTAP</td>
</tr>
<tr>
<td>Jennic / NXP</td>
<td>Jennic 2.4GHZ (SoC)</td>
<td>128KB RAM, 128 KB ROM</td>
<td>-</td>
<td>Analog, Digital, ADC, SPI, Digital audio interface, UART</td>
<td>Closed platform. Proprietary protocol stack – ZigBee / 6LoWPAN Pure commercial platform. Plug-and-play...</td>
<td></td>
</tr>
</tbody>
</table>
### WSN Core

#### What we use...

<table>
<thead>
<tr>
<th>Product Name</th>
<th>Extras</th>
<th>Notes:</th>
</tr>
</thead>
<tbody>
<tr>
<td>XM1000</td>
<td>Indoors RF range: ~30 m (without Line-of-Sight).</td>
<td>Not advisable for industrial environments due to antenna. SMA connector / Dipole antenna <strong>is not supported</strong>.</td>
</tr>
<tr>
<td>CM5000-SMA</td>
<td>Similar as XM1000, less powerfull. 5dBi dipole antenna</td>
<td>Advisable for industrial environments, due to antenna option. <strong>Network compatible to XM1000</strong></td>
</tr>
</tbody>
</table>
WSN Core

When selecting motes for your applications...

- One size doesn’t fit them all.

- Support by company and open source community

- Power consumption

- Interoperability, Accessibility and tools (μProcessor toolchains, etc)

- **Antenna design and antenna performance** – standard-compliance &/ implementation is not panacea to RF problems....
WSN Programming

- Motes selection ↔ Programming environment.
- Open source & Research platforms: Linux-alike environments
- Plug-and-play and closed platforms: wide range of tools.

- When programming a mote → programming its μProcessor to:
  - access the peripheral devices (transceiver, leds, sensors etc)
  - handle, store, modify the acquired information.
WSN Programming

Direct μProcessor programming

- Low-level / Embedded C & Assembly
- Hardware specific
- Faster (simplified applications & experienced programmer)
- Not suitable for sophisticated applications & network topologies

Real time Operating Systems

- A level of abstraction between the programmer and the hardware platform
- HW Interoperability of WSN application
- Allows better control on the platform
- Suitable for more complex network topologies
WSN Programming


Hardware Abstraction Layer

- APP
- Transport
- NWK
- MAC

- Sensors
- Memory
- μProcessor
- TRX / PHY (MAC)
- Other (e.g. battery monitor, GIOs, etc)
<table>
<thead>
<tr>
<th>WSN Programming</th>
<th>Contiki</th>
</tr>
</thead>
<tbody>
<tr>
<td>TinyOS</td>
<td>The Open Source OS for the Internet of Things</td>
</tr>
</tbody>
</table>

| First Release     | 1999 | 2005 |
| Supported Platforms (in official distributions) | 17 | 26 |
| Community Support & Forums | Yes | Yes |
| Programming Language | nesC | C |
| Single / Multiple Thread | Single (multithread is optional) | Single (multithread – explicitly defined library) |
| Structure | Component-based | Protothreads |
| Simulator / Emulator | TOSSIM (python) | Cooja / MSPSim Emulator (java) |
| OTAP | Yes | Yes |
| Protocol Stack | (802.15.4) MAC (not fully supported)  
Collection Tree  
6LoWPAN | (802.15.4) MAC (not fully supported)  
Radio Duty Cycle & MAC  
RIME / uIP  
6LoWPAN |
| Great flexibility in generating highly customizable protocol stack | With default distribution: RIME or 6LoWPAN (modifiable) |
| Interfacing with host (Serial Communication) | Specific format (ActiveMessageC) | Flexible  
(but provides tools s.a. SLIP) |
| Documentation* | ★ ★ ★ | ★ |
| Debugging experience* | ★ ★ ★ ★ ★ ★ ★ ★ ★ ★ | ★ ★ |
WSN Programming

- Component-based architecture, implementing one single stack
- Event-based, non-blocking design that allows intra-mote concurrency
- Written in NesC
  - Structured, component-based C-like programming language

Programming Model:

- **Components**: encapsulate state and processing – use or provide interfaces
- **Interfaces** list commands and events
- **Configurations** wire components together
Two components are wired via interfaces.

Components are statically linked to kernel (not reconfigurable after compiling)

The kernel is a chain of components interacting via interfaces

**Interface:**
The set of functions (*events and commands*).

**Commands:** the user component *may* use and the provider component *must* define and implement.

**Events:** the provider component *must* define and *may* implement and the user component *must* implement.
WSN Programming

Sequential flow control while keeping a single stack

Event-based → Invoking processes (non-blocking)

Using protothreads: a programming abstraction that combines events and threads

Single stack and sequential flow control

Posting events or polling
Each process is essentially a protothread

```c
int a_protothread(struct pt *pt) {
    PT_BEGIN(pt);
    /* ... */
    PT_WAIT_UNTIL(pt, condition1);
    /* ... */
    if(something) {
        /* ... */
        PT_WAIT_UNTIL(pt, condition2);
        /* ... */
    }
    PT_END(pt);
}
```
Hello-world in WSN programming.

A Blinking-Led Application

• Program a mote to blink a led every T seconds.
module BlinkC @safe()
{
  uses interface Timer<TMilli> as Timer0;
  uses interface Timer<TMilli> as Timer1;
  uses interface Timer<TMilli> as Timer2;
  uses interface Leds;
  uses interface Boot;
}

implementation
{
  event void Boot.booted()
  {
    call Timer0.startPeriodic( 250 );
    call Timer1.startPeriodic( 500 );
    call Timer2.startPeriodic( 1000 );
  }

  event void Timer0.fired()
  {
    dbg("BlinkC", "Timer 0 fired @ %s\n", sim_time_string());
    call Leds.led0Toggle();
  }

  event void Timer1.fired()
  {
    dbg("BlinkC", "Timer 1 fired @ %s \n", sim_time_string());
    call Leds.led1Toggle();
  }

  event void Timer2.fired()
  {
    dbg("BlinkC", "Timer 2 fired @ %s\n", sim_time_string());
    call Leds.led2Toggle();
  }
}
#include "contiki.h"
#include "dev/leds.h"

#include <stdio.h> /* For printf() */

/* We declare the process */
PROCESS(blink_process, "LED blink process");

/* We require the processes to be started automatically */
AUTOSTART_PROCESSES(&blink_process);

/* Implementation of the process */
PROCESS_THREAD(blink_process, ev, data)
{
    static struct etimer timer;
    PROCESS_BEGIN();
    while (1)
    {
        /* we set the timer from here every time */
        etimer_set(&timer, CLOCK_CONF_SECOND);
        /* and wait until the event we receive is the one we’re
         * waiting for */
        PROCESS_WAIT_EVENT_UNTIL(ev == PROCESS_EVENT_TIMER);
        printf("Blink... (state %0.2X).
", leds_get());
        /* update the LEDs */
        leds_toggle(LEDS_GREEN);
    }
    PROCESS_END();
}

One main.c for each platform: Core & Network processes

process_init();
process_start(&etimer_process, NULL);
ctimer_init();
init_platform();
set_rime_addr();

//-----------------------low level api to phy-----------------------
cc2420_init();
{
    uint8_t longaddr[8];
    uint16_t shortaddr;
    shortaddr = (rimeaddr_node_addr.u8[0] << 8) + rimeaddr_node_addr.u8[1];
    memset(longaddr, 0, sizeof(longaddr));
    rimeaddr_copy((rimeaddr_t *)&longaddr, &rimeaddr_node_addr);
    cc2420_set_pan_addr(IEEE802154_PANID, shortaddr, longaddr);
} cc2420_set_channel(RF_CHANNEL);
memcpy(&uip_lladdr.addr, ds2411_id, sizeof(uip_lladdr.addr));
queuebuf_init();
NETSTACK_RDC.init();
NETSTACK_MAC.init();
WSN Programming

The communication layers in Contiki [29-31]

• The uIP TCP/IP stack
  • Lightweight TCP/IP functionalities for low complexity μControllers
  • A single network interface (IP, ICMP, UDP, TCP)
  • Compliant to RFC but the Application layer is responsible for handling retransmissions (reduce memory requirements)

• The Rime protocol stack
  • A set of communication primitives (keeping pck headers and protocol stacks separated)
  • A pool of NWK protocols for ad-hoc networking
  • Best-effort anonymous broadcast to reliable multihop flooding and tree protocols
WSN Programming

How does Rime work

• Rime is a software trick
• A stack of NWK layers
  • Each layer is associated with a channel
• 2KB memory footprint
• Interoperability and ease in changing the protocol stack
WSN Programming

How does Rime work – Example

• The Collection Tree Protocol (CTP)
  • Tree-based hop-by-hop reliable data collection
  • Large-scale network (e.g. environmental or industrial monitoring)

• Reliable Unicast Bulk
  • Event-driven data transmission of a large data volume
  • Personal health-care
## WSN Programming

<table>
<thead>
<tr>
<th>Layer</th>
<th>Description</th>
<th>Channel</th>
<th>Contribution to Rime Header</th>
</tr>
</thead>
<tbody>
<tr>
<td>Broadcast</td>
<td>Best-effort local area broadcast</td>
<td>129</td>
<td>Sender ID</td>
</tr>
<tr>
<td>Neighbor discovery</td>
<td>Periodic Neighbor Discovery mechanism</td>
<td>2</td>
<td>Receiver ID, Application Channel</td>
</tr>
<tr>
<td>Unicast</td>
<td>Single-hop unicast to an identified single-hop neighbor</td>
<td>146</td>
<td>Receiver ID</td>
</tr>
<tr>
<td>Stubborn unicast</td>
<td>Repeatedly sends a packet until cancelled by upper layer</td>
<td></td>
<td>Receiver ID</td>
</tr>
<tr>
<td>Reliable Unicast</td>
<td>Single-hop reliable unicast (ACKs and retransmissions)</td>
<td>144</td>
<td>Packet Type and Packet ID</td>
</tr>
</tbody>
</table>

[Diagram of WSN Programming]
WSN Programming

- **Cooja**
  - The Contiki emulator for running WSN applications.
  - Very useful for debugging your codes – the same code you test on cooja, the same you upload to your mote.
  - Evaluating the network performance (?) – has very simplifying models for radio propagation:
    - Unit disk model: Edges are instantly configured according to power attenuation w.r.t to distance & success ratio (configurable).
    - Directed graph radio medium: Considers preconfigured edges, without checking the output power.
    - Multipath ray tracer: Simulates reflection and diffraction through homogeneous obstacles (considers that all nodes have the same transmission power).
Outline

• Part 1: Applications, Standards and Protocols
  • Introduction & Reasoning of Existence
  • Sensing, Processing, and Networking Aspects
  • Standards, Topologies & Protocols

• Part 2: WSN Programming
  • WSN Core and types of nodes
  • Real-time Operating Systems
  • Examples & Hands on Session
Hands on Session

What we are going to do...

Contiki
The Open Source OS for the Internet of Things
Hands on Session

What we are going to use...in order to upload code to the motes

• FTDI drivers (for Windows machines only) – USB2Serial

• How the host computer reserves a mote:
  • COM<No> (Windows – Device Manager)
  • /dev/ttyUSB<No> (Linux) [cat /var/log/syslog]
  • Make sure that you have access on device (for programming it)
    chmod 777 /dev/ttyUSB0
  • Serial dump: make TARGET=sky MOTES=/dev/ttyUSB0 login
Hands on Session at Cooja

• Cooja (for emulating the motes behavior)
  Guidelines for running the codes at Cooja:

  1. From your VM / Instant Contiki run the “cooja” application
  2. Follow the instructions given at: http://www.contiki-os.org/start.html (step 3) for creating a new simulation
   Select “sky” as the mote type
  3. The result of the printf is shown at the “Mote Output” view
Hands on Session at Cooja
Hands on Session

Hello World 😊 contiki/examples/hello-world

[Code structure & compile]

```
#include "contiki.h"

#include <stdio.h> /* For printf() */
/*-----------------------------------------------*/
PROCESS(hello_world_process, "Hello world process");  /**Process definition**/
AUTOSTART_PROCESSES(&hello_world_process);         /**Process Start**/
/*-----------------------------------------------*/
PROCESS_THREAD(hello_world_process, ev, data) /**Process implementation**/
{
  PROCESS_BEGIN(); /**Always first**/
  printf("Hello, world\n");  //process core
  PROCESS_END(); /**Always last**/
} /*-----------------------------------------------*/

CONTIKI_PROJECT = hello-world
all: $(CONTIKI_PROJECT)
include $(CONTIKI)/Makefile.include
```
Hands on Session

Hello World ☺ contiki/examples/hello-world
[Code structure & compile]

Program:
1. Open command terminal.
2. cd contiki/examples/hello-world
3. **make** TARGET=<platform*> hello-world.upload (compile and program)

Serial Dump
1. At new tab (File/Open new tab).
2. **make** TARGET=sky MOTES=/dev/ttyUSB0 login

*sky/xm1000*
Hands on Session

Hello World 😊 contiki/examples/hello-world

[How to trigger a process]
• How to wake up from a process

Keep on mind that:

Automatic variables not stored across a blocking wait

When in doubt, use static local variables
Hands on Session

Hello World 😊 contiki/examples/hello

[How to trigger a process]

• Timers

  • Event timer (etimer) : Sends an event when expired

  • Callback timer (ctimer) : Calls a function when expired – used by Rime
Hands on Session

Hello World 😊 contiki/examples/hello-world
[How to trigger a process]

From hello-world.c generate a new application (print-and-blink.c) that:

1. periodically (e.g. per second) prints a message.
2. when the message is printed a led toggles

   #include “leds.h”
   
   leds_toggle(LEDS_RED / LEDS_GREEN / LEDS_YELLOW)

   macro for time: CLOCK_SECOND
/*---------------------------------------------------------------------------*/
PROCESS(print_and_blink_process, "Print and blink process");
AUTOSTART_PROCESSES(&print_and_blink_process);
/*---------------------------------------------------------------------------*/
PROCESS_THREAD(print_and_blink_process, ev, data)
{
    static struct etimer et;

    PROCESS_BEGIN(); /**<Always first**/

    while(1) {

        etimer_set(&et, CLOCK_SECOND);

        PROCESS_WAIT_EVENT_UNTIL(etimer_expired(&et));

        printf("Echo\n");

        leds_toggle(LEDS_GREEN);

    }

    PROCESS_END(); /**<Always last**/
}
Hands on Session

Sensing 😊 contiki/examples/hello-world
[Access a sensor]

• Sensor: supported by contiki (platform/dev/<platform>)

• `const struct sensors_sensor`
  • `@sky: sht11_sensor.value(type) --global`  
    //type = SHT11_SENSOR_TEMP, SHT11_SENSOR_HUMIDITY

  ```
  light_sensor.value(type) --global
  //type = LIGHT_SENSOR_TOTAL_SOLAR, LIGHT_SENSOR_PHOTOSYNTHETIC
  ```

  ```
  battery_sensor.value(type) --global
  //type = 0
  ```

• ACTIVATE / DEACTIVATE (<sensors_sensor>)
Sensing

Hands on Session

Sensing contiki/examples/hello-world
[Access a sensor]

From the print-and-blink, generate a new application (sense-and-blink.c) that:

1. Periodically sample one or more of the on-board sensors
   
   ```
   #include "dev/light-sensor.h" / "dev/sht11-sensor.h" / "dev/battery-sensor.h"
   ```

   ```
   SENSORS_ACTIVATE(<>)
   ```

   [Sample...]

   ```
   SENSORS_DEACTIVATE(<>)
   ```

2. When done prints the sampled value and toggles a led

Command for serial dump: make TARGET=sky MOTES=/dev/ttyUSB0 login
Hands on Session

1 process

```c
struct sensor_datamsg{
    uint16_t temp;
    uint16_t humm;
    uint16_t batt;
};
sensor_datamsg;

PROCESS_THREAD(sense_and_blink_process, ev, data){
    static struct etimer et;
    static struct sensor_datamsg msg;
    PROCESS_BEGIN(); /* Always first */
    SENSORS_ACTIVATE(sht11_sensor);
    SENSORS_ACTIVATE(battery_sensor);
    while (1) {
        etimer_set(&et, CLOCK_SECOND);
        PROCESS_WAIT_EVENT_UNTIL(etimer_expired(&et));
        msg.temp = sht11_sensor.value(SHT11_SENSOR_TEMP);
        msg.humm = sht11_sensor.value(SHT11_SENSOR_HUMIDITY);
        msg.batt = battery_sensor.value(0);
        printf("Sensor raw values: temperature:%d, humidity: %d, battery: %d\n", msg.temp, msg.humm, msg.batt);
        leds_toggle(LEDS_GREEN);
    }
    SENSORS_DEACTIVATE(sht11_sensor);
    SENSORS_DEACTIVATE(battery_sensor);
    PROCESS_END(); /* Always last */
}
```

2 processes

```c
static struct sensor_datamsg msg;
static process_event_t event_data_ready;

PROCESS_THREAD(sense_process, ev, data){
    PROCESS_BEGIN(); /* Always first */
    SENSORS_ACTIVATE(sht11_sensor);
    SENSORS_ACTIVATE(battery_sensor);
    while (1) {
        process_post(&print_and_blink_process, event_data_ready, &msg);
    }
    PROCESS_END(); /* Always last */
}
```

```c
PROCESS_THREAD(print_and_blink_process, ev, data){
    PROCESS_BEGIN(); /* Always first */
    while (1) {
        PROCESS_YIELD_UNTIL(ev==event_data_ready);
        printf("Sensor raw values: temperature:%d, humidity: %d, battery: %d\n", msg.temp, msg.humm, msg.batt);
        leds_toggle(LEDS_GREEN);
    }
    PROCESS_END(); /* Always last */
}
```
Hands on Session

Wireless Sensing 😊 contiki/examples/hello-world
[Access a sensor & trx]

Communication:
• Each type of connection (rime / uIP / 6LoWPAN) defines a structure

• Each type of rime connection defines a struct for the callback function (rx events).
  Callback function has to have a specific definition...

• Each rime-based connection is associated with a predefined channel (>128)
Hands on Session

Wireless Sensing 😊 contiki/examples/hello-world
[Access a sensor & trx]

@ rime:
• packetbuf module for packet buffer management
• Struct rimeaddr_t for rime addressing…
  typedef union {
    unsigned char u8[RIMEADDR_SIZE]; //=2
  } rimeaddr_t;

@ uip:
• uipbuf module for packet buffer management
• Struct ipaddr_t

Unless otherwise specified,
IP= 176.12.RIME_ADDR[0]. RIME_ADDR[1]
The packetbuf module does Rime's buffer management. More...

### Files

<table>
<thead>
<tr>
<th>File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>packetbuf.c</td>
<td>Rime buffer (packetbuf) management.</td>
</tr>
<tr>
<td>packetbuf.h</td>
<td>Header file for the Rime buffer (packetbuf) management.</td>
</tr>
</tbody>
</table>

### Defines

<table>
<thead>
<tr>
<th>Define</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PACKETBUF_SIZE</td>
<td>128</td>
<td>The size of the packetbuf, in bytes.</td>
</tr>
<tr>
<td>PACKETBUF_HDR_SIZE</td>
<td>48</td>
<td>The size of the packetbuf header, in bytes.</td>
</tr>
</tbody>
</table>

### Functions

<table>
<thead>
<tr>
<th>Function Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>packetbuf_clear (void)</td>
<td>Clear and reset the packetbuf.</td>
</tr>
<tr>
<td>packetbuf_clear_hdr (void)</td>
<td>Clear and reset the header of the packetbuf.</td>
</tr>
<tr>
<td>packetbuf_copyfrom (void, const void *)</td>
<td>Copy from external data into the packetbuf.</td>
</tr>
<tr>
<td>packetbuf_compact (void)</td>
<td>Compact the packetbuf.</td>
</tr>
<tr>
<td>packetbuf_copyto_hdr (void, *to)</td>
<td>Copy the header portion of the packetbuf to an external buffer.</td>
</tr>
<tr>
<td>packetbuf_copyto (void *)</td>
<td>Copy the entire packetbuf to an external buffer.</td>
</tr>
<tr>
<td>packetbuf_hdralloc (int size)</td>
<td>Extend the header of the packetbuf, for outbound packets.</td>
</tr>
<tr>
<td>packetbuf_hdrreduce (int size)</td>
<td>Reduce the header in the packetbuf, for incoming packets.</td>
</tr>
<tr>
<td>packetbuf_set_datalen (uint16_t)</td>
<td>Set the length of the data in the packetbuf.</td>
</tr>
<tr>
<td>packetbuf_dataptr (void)</td>
<td>Get a pointer to the data in the packetbuf.</td>
</tr>
<tr>
<td>packetbuf_hdrptr (void)</td>
<td>Get a pointer to the header in the packetbuf, for outbound packets.</td>
</tr>
<tr>
<td>packetbuf_reference (void, *ptr, uint16_t)</td>
<td>Point the packetbuf to external data.</td>
</tr>
<tr>
<td>packetbuf_is_reference (void)</td>
<td>Check if the packetbuf references external data.</td>
</tr>
<tr>
<td>packetbuf_reference_ptr (void, *ptr, uint16_t)</td>
<td>Get a pointer to external data referenced by the packetbuf.</td>
</tr>
<tr>
<td>packetbuf_datalen (void)</td>
<td>Get the length of the data in the packetbuf.</td>
</tr>
<tr>
<td>packetbuf_hdrlen (void)</td>
<td>Get the length of the header in the packetbuf, for outbound packets.</td>
</tr>
<tr>
<td>packetbuf_totlen (void)</td>
<td>Get the total length of the header and data in the packetbuf.</td>
</tr>
</tbody>
</table>
Hands on Session

Wireless Sensing 😊 contiki/examples/hello-world
[Access a sensor & trx]
From the sense-and-tx, generate a new application (sense-and-trx.c) that:
1. Periodically samples from on-board temperature sensor
2. When done broadcast the value
3. Upon the reception of an incoming packet, print its contents and the source node id

#include net/rime.h
static const struct broadcast_callbacks broadcast_call = {broadcast_recv}; -- visible outside process
Defined as: static void broadcast_recv(struct broadcast_conn *c, const rimeaddr_t *from)

static struct broadcast_conn broadcast; -- visible outside process

Inside process:
broadcast_open(&broadcast, 129, &broadcast_call); --connection -- 129: the broadcast rime channel
packetbuf_copyfrom(const void *data, data length); --form tx buffer
broadcast_send(&broadcast); -- send to connection
Hands on Session

```c
PROCESS_THREAD(send_and_blink_process, ev, data) {
  static uint8_t data2send[sizeof(sensor_datamsg)];
  PROCESS_BEGIN(); /**< Always first **/
  broadcast_open(&broadcast, 129, &broadcast_call);
  while (1) {
    PROCESS_YIELD_UNTIL(ev==event_data_ready);
    data2send[0] = msg.temp & 255; // lsb
    data2send[1] = msg.temp >> 8;  // msb
    data2send[2] = msg.humm & 255;
    data2send[3] = msg.humm >> 8;
    data2send[5] = msg.batt >> 8;
    packetbuf_copyfrom(data2send,sizeof(sensor_datamsg));
    broadcast_send(&broadcast);
    // printf("Sensor raw values: temperature:%d, humidity: %d, battery: %d\n", msg.temp, msg.humm, msg.batt);
    leds_toggle(LEDS_GREEN);
  }
  PROCESS_END(); /**< Always last **/
}
```

**Send**

**Receive**

```c
static void broadcast_recv(struct broadcast_conn *c, const rimeaddr_t *from) {
  uint8_t *appdata;
  int i;
  appdata = (uint8_t *)packetbuf_dataptr();
  printf("Data recv:");
  for (i=0;packetbuf_datalen();i++)
    printf("%u ",appdata[i]);

  // this is the id of the sender (as defined in compile time).
  printf("from: %d.%d\n", from->u8[0], from->u8[1]);
  printf("\n");
}
```
References

2. www.advanticsys.com
3. www.shimmersensing.com
4. www.jennic.com
6. www.digi.com/xbee
9. www.tinyos.net
10. www.contiki-os.org