Implementing the Internet of Things with Contiki

Laboratory within the postgraduate course “Topics in Distributed Computing” (MO809) delivered by the University of Campinas

Lecture 1

$whoami

Carlo Puliafito

Where – Department of Information Engineering, University of Pisa, Italy

Email - carlo.puliafito@ing.unipi.it

Teaching material – will be gradually uploaded on Google Classroom

Outline of the course

- **WSANs, the Contiki OS, and the Cooja simulator**
- The uIPv6 stack and first hands-on using UDP
- Working with RPL (IPv6 Routing Protocol for Low-power and Lossy Networks)
- How to implement a WSAN Gateway
- Implementing solutions based on CoAP (Constrained Application Protocol)

Wireless Sensor and Actuator Networks (WSANs)

A **WSAN** is a network of wireless nodes, called ***motes***, which embed:

- **Sensors** → perceive the environment
- **Actuators** → control the environment
- **Radio transceiver** → exchange data
- **Microcontroller** → perform computation
- **Memory** → store data
- Energy source (e.g., **battery**, energy harvesting)

Motes are resource-constrained

SkyMote

Zolertia Z1

Examples of Operating Systems (OS) for motes:

- **Contiki**
- TinyOS
- LiteOS

Download the IoT Workshop VM

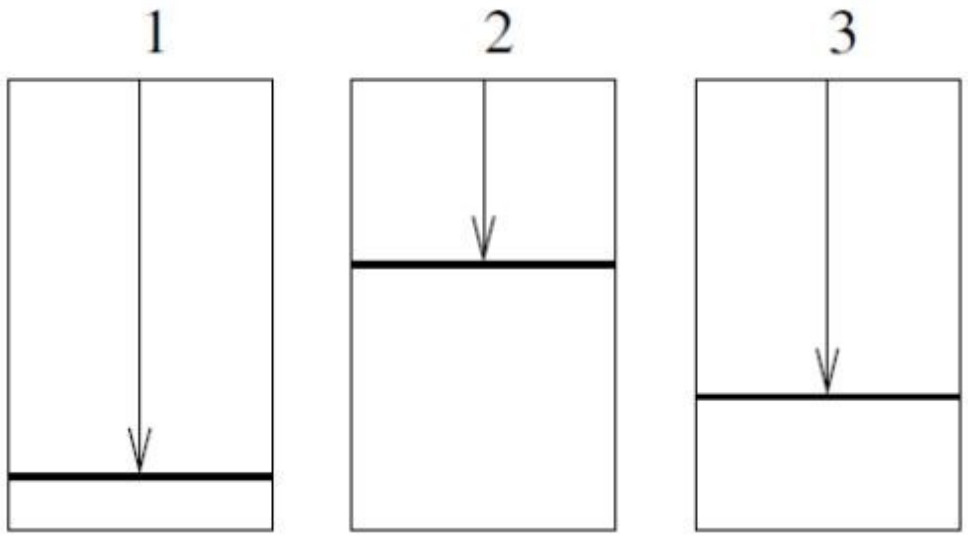
Download **IoT Workshop VM**, a complete Contiki development environment that includes:

- Contiki OS and all the development tools
- The Cooja simulation/emulation environment
- Code examples

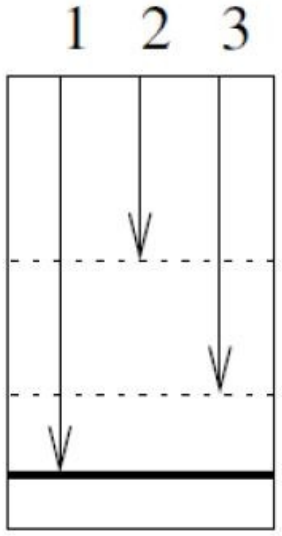
- It is highly suggested to use **VMware** as hypervisor
- To log into the VM, the password is **user**
- Change the keyboard settings into the language of your preference

Protothreads

Contiki firmwares are implemented as one or more protothreads. **Protothreads** are lightweight versions of threads – they all share the same stack in memory.



Threads



Protothreads

Protothread implementation in Contiki: The “Hello, world” example

```
#include “contiki.h”
#include <stdio.h>
/* Declare the process */
PROCESS(hello_world_process, “Hello world”);
/* Make the process start when Contiki finishes booting */
AUTOSTART_PROCESSES(&hello_world_process);

/* Define the process code */
PROCESS_THREAD(hello_world_process, ev, data) {
   PROCESS_BEGIN(); /* Must always come first */

   printf(“Hello, world!\n”); /*code goes here*/

   PROCESS_END(); /* Must always come last */
}
```

Contiki is event-driven

- A protothread is **executed only when it receives an event** (e.g., a timer expired, a button pressed)
- A protothread **runs until** it reaches the **PROCESS_END()** statement **or until it waits again for a new event**:

```
/* Waits for a generic event */
PROCESS_WAIT_EVENT();
/* Waits for a specific event */
PROCESS_WAIT_EVENT_UNTIL(condition c);
```

Remember – Local variables must be declared as **static** to keep their values across **WAITs**

Etimer

Sends an event when it expires. How to work with etimers?

```
#include “sys/etimer.h”                  // Includes the header file for etimers

static struct etimer et;                 // Declares an etimer

etimer_set(&et, CLOCK_SECOND*4); // Sets the etimer to e.g., 4 seconds

etimer_expired(&et);                     // Checks if etimer has expired

etimer_reset(&et);                       // Resets the etimer … periodic behavior

etimer_restart(&et);                     // Restarts the etimer … allows time drift
```

An example with etimers

```
#include “contiki.h”
#include “sys/etimer.h” /* Include the header file */

PROCESS(example_process, “Example”);
AUTOSTART_PROCESSES(&example_process);

PROCESS_THREAD(example_process, ev, data) {
  static struct etimer et; /* Declare the etimer */
  PROCESS_BEGIN();
  etimer_set(&et, CLOCK_SECOND); /* set et to 1 second */

  while(1){
    PROCESS_WAIT_EVENT_UNTIL(etimer_expired(&et)); /* wait for et expiration */
    etimer_reset(&et); /* reset et to trigger again in 1 second */
  }
  PROCESS_END();
}
```

LEDs

How to work with LEDs?

```
#include “dev/leds.h”        // Includes the header file for LEDs

LEDS_BLUE … LEDS_RED … LEDS_GREEN … LEDS_ALL // LEDs identifiers

leds_on(LEDS_GREEN);         // Switches the green LED on

leds_off(LEDS_BLUE);         // Switches the blue LED off

leds_toggle(LEDS_RED);       // Toggles (i.e., changes the state) the red LED
```

Temperature sensor

The temperature sensor for the **SkyMote** is **sht11**. That for the **Zolertia Z1** is **tmp102**. How to work with them?

```
#include “dev/sht11/sht11-sensor.h” // Include the header files for sensors
#include “dev/tmp102.h”

SENSORS_ACTIVATE(sht11_sensor);      // Activate sensors … usually between
SENSORS_ACTIVATE(tmp102);               PROCESS_BEGIN() and while(1)

int temp = (sht11_sensor.value(SHT11_SENSOR_TEMP)/10-396)/10;
int temp = tmp102.value(TMP102_READ)/100;                  // Sense temperature
```

User button

The user button is **considered as a sensor** in Contiki.

How to work with the user button?

```
#include “dev/button-sensor.h”   // Includes the header file for the button

SENSORS_ACTIVATE(button_sensor); // Activates the button … usually between
                                 PROCESS_BEGIN() and while(1)

PROCESS_WAIT_EVENT_UNTIL(ev == sensors_event && data == &button_sensor);
                                // Waits until the user presses the button
```

Distinction among platforms in the same code

- Sometimes, few lines of code may differ across platforms (e.g., the temperature sensor code).
- Should we write N source codes that only differ for those few lines?
- What if we want to run a source code on any platform?

```
…
#if CONTIKI_TARGET_Z1
#include “dev/tmp102.h”
#else
#include “dev/sht11/sht11-sensor.h”
#endif
…
#if CONTIKI_TARGET_Z1
SENSORS_ACTIVATE(tmp102);
#else
SENSORS_ACTIVATE(sht11_sensor);
#endif
```

Random library

How to use the Contiki random library?

```
#include “lib/random.h” // Includes the header file

random_rand(); // Generates a random value between 0 and 65535
```

Makefile

The project must include a Makefile that **specifies how to produce the binary code.**

It must be exactly called “Makefile”.

It must be located inside the project directory.

```
CONTIKI_PROJECT = source_code_name

all: $(CONTIKI_PROJECT)

CONTIKI = /home/user/contiki

include $(CONTIKI)/Makefile.include
```

project-conf.h

The project-conf.h is often used to **override the Contiki OS default configurations**.
It must be located inside the project directory.

Add to Makefile:

```
CFLAGS += -DPROJECT_CONF_H=\"project-conf.h\“
```

For example, change the Radio Duty Cycling:

```
#ifndef PROJECT_CONF_H_
#define PROJECT_CONF_H_

#undef NETSTACK_CONF_RDC
#define NETSTACK_CONF_RDC nullrdc_driver

#endif
```

Cooja

- Cooja is a Java-based simulator/emulator for Contiki motes
- The hardware of motes is emulated
- Wireless connection among motes is simulated

To launch Cooja, go to "**`contiki/tools/cooja`**"
and run the command "**`ant run`**"

Brief tutorial on Cooja at this link.

Exercise 1

What to do: write a program that loops indefinitely and waits for an event. If the button has been pressed, toggles LEDs and prints out “Button pressed”. If, instead, the timer has expired, toggles LEDs and prints out “Timer expired”. Run the program in Cooja.

Solution: exercise1.c

Exercise 2

What to do: write a program that:

- with a period of 5 seconds, toggles the green LED and gives the user 0.5 seconds to press the button;
- every time the user succeeds, his score is increased by 1;
- in both cases (either the user succeeds or not), the program prints out “Your score is *M* out of *N*”.

Run the program in Cooja.

Solution: exercise2.c