Internet Of Things Lab

Francesca Righetti
francesca.righetti@ing.unipi.it

IoT and Contiki-NG

IoT

Outline

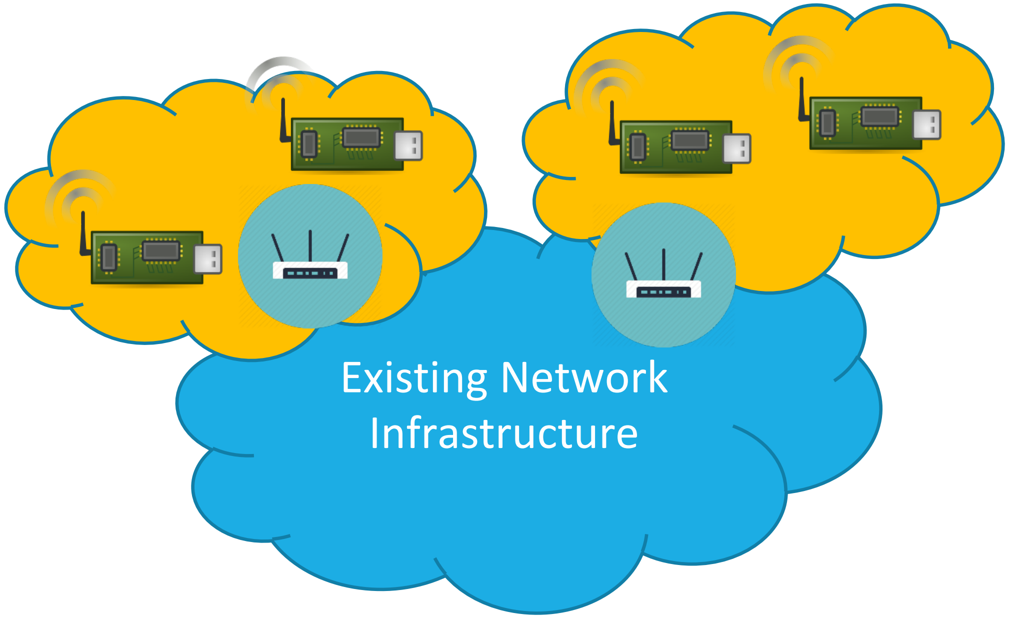
- Introduction
- Contiki-NG OS. Cooja, the network simulator
- Contiki-NG basic programming (buttons, leds, serial lines, timers)
- Layer 2 communication with IEEE 802.15.4
- IoT and IPv6 - integration into existing networks
- IPv6 Border Router
- CoAP basic operations in Contiki-NG (server)

IoT

francesca.righetti@ing.unipi.it

Internet of Things

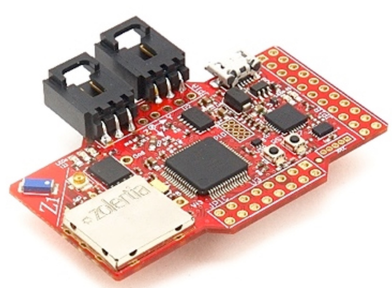
- Nowadays almost every device has connectivity capabilities
  - Networks of sensors/actuators
- New ways to develop applications for :
  - Smart home
  - Smart building
  - Smart factory
  - Smart city

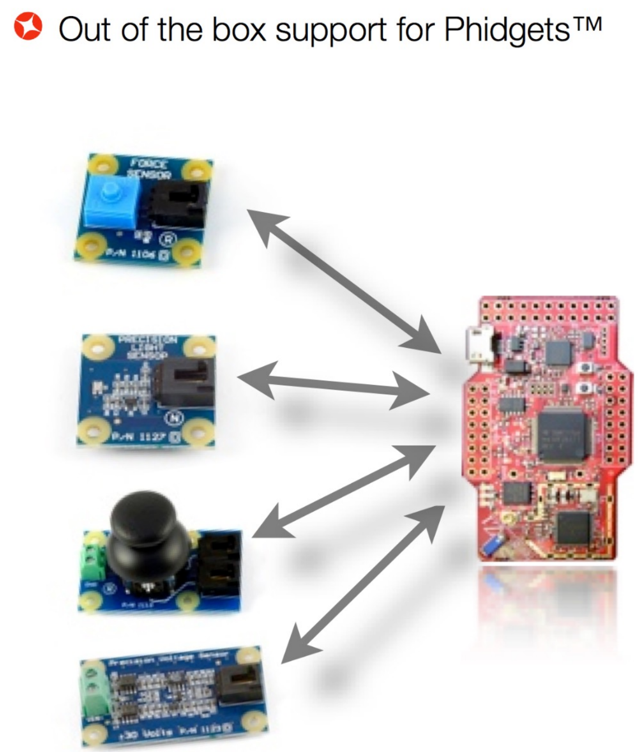


IoT

Internet of Things

- Sensors may be constrained devices
  - Computational power
  - Memory
  - Transmission range
  - Battery life





- Ad-Hoc Operating Systems (OSs) are required

francesca.righetti@ing.unipi.it

Internet of Things – Operating System

- The OS is the **interface** between the hardware and the programmer. It manages:
  - **Drivers**
    - Sensors
    - Radio transceiver
  - **Processes**
  - **Network stacks**
  - **Power management**

- **Examples** of OS: [Contiki-NG](), FreeRTOS, Riot, Zephyr

IoT

francesca.righetti@ing.unipi.it

Internet of Things – Contiki-NG

- Contiki-NG: open-source OS for resource-constrained networked embedded systems

  - Programmed in C
  - Support for several platforms (Launchpad, Zolertia, etc.)
  - Support for many CPU
  - Event driven kernel
    - **Protothreads**

- In Contiki-NG traditional Processes cannot be adopted, as they are demanding in terms of resources

francesca.righetti@ing.unipi.it

Internet of Things – Contiki-NG

- **Event driven kernel only uses events**
  - Difficult to program
  - No sequential flow of control
  - Low overhead

VS

- **Threads**
  - **Easy to program**
  - **Sequential flow of control**
  - **High overhead (each thread has its own stack)**

francesca.righetti@ing.unipi.it

Contiki-NG Prototreads

IoT

francesca.righetti@ing.unipi.it

Contiki-NG - Protothreads

- The Contiki-NG repository is structured as follows:
  - **os**: Contiki-NG system source code (e.g., systems primitives such as processes and timers, all libraries and services and the networking stack).
  - **arch**: all platform-specific code (e.g., CPU, device and platform drivers). A list of supported platforms can be found under arch/platforms.
  - **examples**: Contains ready-to-use example projects.
  - **tools**: Contains tools not to be included in a Contiki-NG firmware, but that runs on a computer. Includes flashing tools, the Cooja simulator.
  - **tests**: Contains all continuous integration tests. These run in Travis for every pull request and merge, to ensure non-regression.

IoT

Contiki-NG - Protothreads

- In Contiki-NG, **Processes = Protothreads**
  - Processes are built on top of a lightweight threading library called Protothreads

    «Dunkels, O. Schmidt, T. Voigt, Thiemo, A. Muneeb. Protothreads: Simplifying Event-driven Programming of Memory-constrained Embedded Systems»

```
int a_protothread(struct pt *pt) {
  PT_BEGIN(pt);

  PT_WAIT_UNTIL(pt, condition1);

  if(something) {

    PT_WAIT_UNTIL(pt, condition2);

  }
  PT_END(pt);
}
```

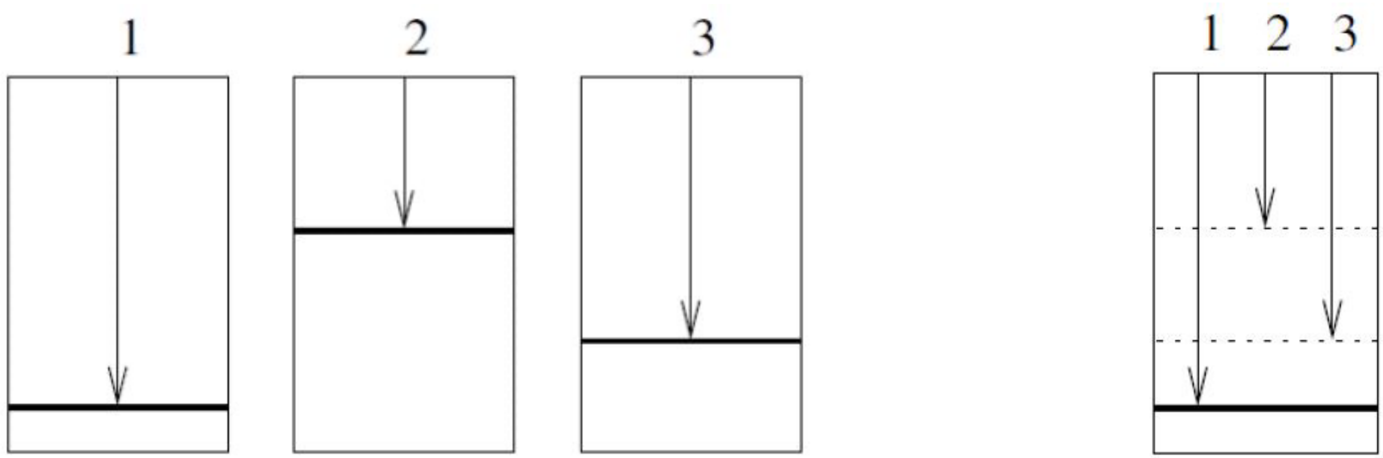
- Prototthreads have:
  - **Event management support**
  - Sequential flow of instructions that can be interrupted to wait for <u>events</u> or <u>conditions</u>

IoT

francesca.righetti@ing.unipi.it

Contiki-NG - Protothreads

- Protothreads share the same stack in memory
- Reduced memory overhead
- Only one protothread can be in execution at each time



Threads **Protothreads**

IoT

Contiki-NG - Protothreads

- The code of each sensor is composed by one or more processes (Protothreads)
- The code of the thread is called PROCESS_THREAD
- Each PROCESS_THREAD contains the code of a single protothread invoked from the process scheduler and it is declared as follows:

```
PROCESS_THREAD(hello_world_process, ev, data)
  {
      PROCESS_BEGIN();
      printf("Hello, world\n");
      PROCESS_END();
  }
```

- A process thread must start with PROCESS_BEGIN(); and terminate with PROCESS_END();

IoT

francesca.righetti@ing.unipi.it

Contiki-NG - Protothreads

- Instructions to manage events or conditions inside the PROCESS_THREAD:

```
PROCESS_WAIT_EVENT();
//Wait for an event.
PROCESS_WAIT_EVENT_UNTIL(c);
//Wait for an event, but with a condition c.
PROCESS_YIELD();
//Wait for any event, equivalent to PROCESS_WAIT_EVENT();.
PROCESS_YIELD_UNTIL(c);
//Wait for any event, but with a condition c.
PROCESS_WAIT_UNTIL(c);
//Wait for a given condition c, may not yield the process
(this MACRO should be used with care).
PROCESS_PAUSE();
//Temporarily yield the process.
PROCESS_EXIT();
//Exit the current running process.
```


francesca.righetti@ing.unipi.it

Contiki-NG - Protothreads

- A process is declared at the top of a source file with the **PROCESS()** macro that takes 2 arguments:
  - The variable that identifies the process (hello_world_process)
  - The name of the process («Hello world process»)

```
PROCESS(hello_world_process, "Hello world process");
```

- The process thread is declared with the macro **PROCESS_THREAD()** that takes 3 arguments:
  - The variable that identifies the processe specified in the PROCESS() call (hello_world_process).
  - Ev, the value of an incoming event.
  - Data, an optional pointer to an event argument object

```
PROCESS_THREAD(hello_world_process, ev, data)
```

IoT

Contiki-NG - Protothreads

- Every Contiki-NG code to run on devices must have a **main process thread** that runs **automatically** as the device **boots up**.
- To make a process **starting automatically** the following autostart declaration must be added

```
PROCESS(example_process, "Example process");
AUTOSTART_PROCESSES(&example_process);
```

IoT

Contiki-NG - Protothreads

- **Protothreads are stackless**: they all share the same global stack of execution. The values local variables are not preserved in protothreads across yields:

```
int i = 1;
PROCESS_YIELD();
printf("i=%d\n", i); // <- prints garbage
```

- The traditional way to deal with this limitation is to declare all protothread-local variables as static:

```
static int i = 1;
PROCESS_YIELD();
printf("i=%d\n", i); // <- prints 1
```

IoT

Contiki-NG Process/Event Scheduler

francesca.righetti@ing.unipi.it

Contiki-NG – Process/Event Scheduler

- The process scheduler dynamically update a scheduling plan and **invoke processes when it is their time to run**
- **The Contiki kernel is event-driven**:
  - Process invocations are done in response to events being posted to processes.
  - Events can be posted by processes or by the Contiki-NG kernel.
- To be invoked for running, processes need to be in the **kernel's list of active processes**.
- To be part of this list, processes need to be started. A process can be started:
  - By another process, through the **process_start()** function
  - Automatically, through the **AUTOSTART_PROCESSES()**

IoT

francesca.righetti@ing.unipi.it

Contiki-NG – Process/Event Scheduler

- Process A starts Process B by invoking **process_start()**. It takes two parameters:

```
process_start(&example_process, NULL);
```

  - The **address of the process** that has to be started
  - The auxiliary data to be sent to the receiving process

- Process B is started. The kernel sends the **PROCESS_EVENT_INIT** event. This event causes Process B to start.

- Process B can be **removed** from the kernel's list of active processes:

  - **Exit by itself,** B invokes the **PROCESS_EXIT()** or **PROCESS_END()**
  - **Exit killed by another process,** Process A calls **process_exit(&ProcessB)** with the address of the process to be killed

IoT

Contiki-NG – Process/Event Scheduler

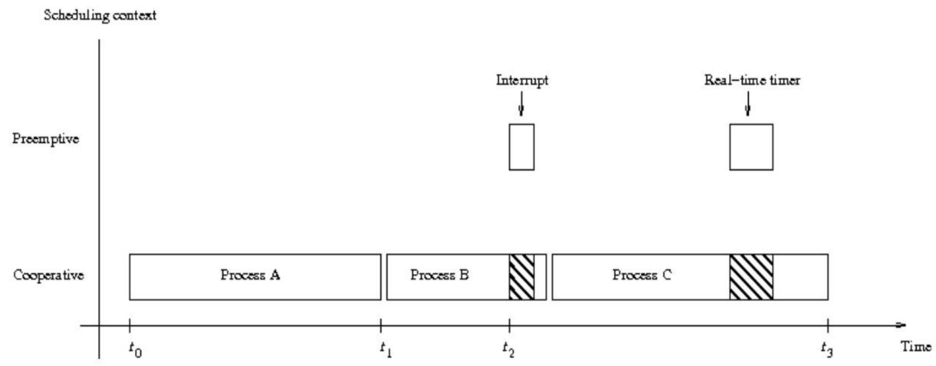
- When Process B exits
  - The kernel sends **PROCESS_EVENT_EXITED** to all the other active processes to inform of Process B exiting.
  - The other processes can thus free up any resource allocations made by Process B.
- If Process B is killed by another process, also Process B receives a **PROCESS_EVENT_EXIT**.
  - This informs Process B that it is about to be killed.
  - Process B can take the opportunity to free up any resource allocations it has made

IoT

francesca.righetti@ing.unipi.it

Contiki-NG – Process/Event Scheduler

- Contiki-NG code runs in either :
  - **Cooperative context**: runs sequentially with respect to other cooperative code
    - Processes perform chunks of work before waiting for an event
    - Cooperative code must run to completion before other cooperatively code can run.
  - **Preemptive context**: temporarily stops the cooperative code
    - Interrupts and real-time timers run in the preemptive context
    - When preemptive code stops the cooperative code, the cooperative code will be resumed when the preemptive code has completed.



IoT

francesca.righetti@ing.unipi.it

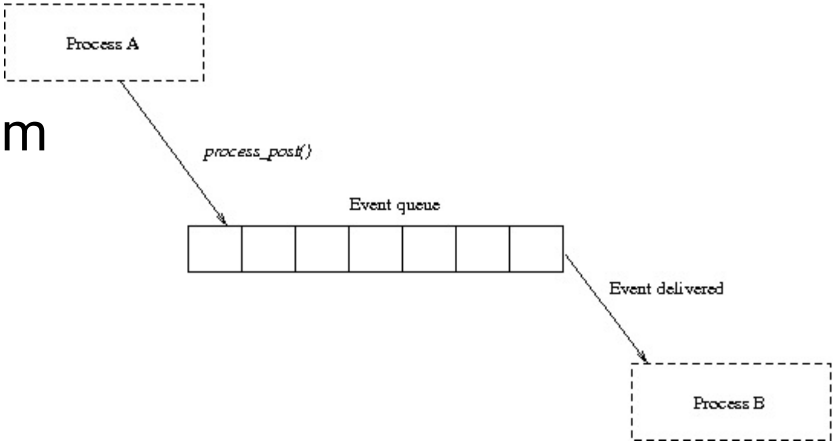
Contiki-NG – Process interactions

- Processes can interact only by exchanging **events,** in two ways:
  - **Asynchronously**
  - **Synchronously**

- Process A **asynchronously** posts an event to Process B, or to all the other active processes.

- After posting the event, **Process A can keep on running and does not block** until the receiving process(es) has processed the event.

The kernel delivers the events from the **event queue** to the receiving processes by invoking them **at some later time**.



Contiki-NG – Process interactions

- Events are **asynchronously** posted with the **process_post()** function. It takes three arguments:

```
process_post(&example_process, PROCESS_EVENT_MSG, msg);
```

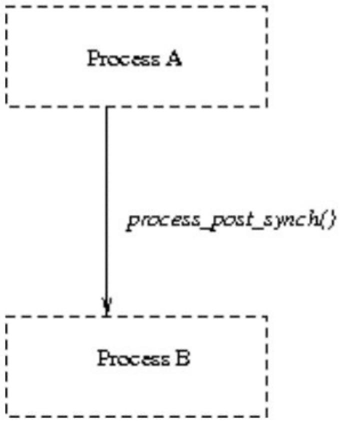
  - &example_process -> The address of the process to which the event should be posted, or PROCESS_BROADCAST if the event should be posted to all processes
  - PROCESS_EVENT_MSG -> The event to be posted
  - msg -> The auxiliary data to be sent with the event

IoT

francesca.righetti@ing.unipi.it

Contiki-NG – Process interactions

- Events posted synchronously, are immediately delivered to the receiving process. There is no event queue.
- Process A synchronously posts an event to Process B (events can be synchronously posted only to a specific receiving process)



- When posting an eve ... sly, the receiving process is directly invoked. The process that posted the event is blocked until the receiving process has finished processing the event.

Contiki-NG – Process interactions

- Events are **synchronously** posted by invoking **process_post_synch()**. It takes the same three parameters as the process_post():

```
process_post_synch(&example_process, PROCESS_EVENT_MSG, msg);
```

  - &example_process -> The address of the process to which the event should be posted
  - PROCESS_EVENT_MSG -> The event to be posted
  - msg -> The auxiliary data to be sent with the event

IoT

francesca.righetti@ing.unipi.it

Contiki-NG – Process interactions

- A poll request is a special type of event (**PROCESS_EVENT_POLL**).

```
process_poll(&example_process);
```

- A process is polled by calling **process_poll().**

- It takes the address of the receiving function process as argument.

- Calling this function on a process causes the process to be **scheduled as quickly as possible**.

- Typically, process_poll() is called by an interrupt handler in order to “wake up” the interested process as soon as possible.

IoT

Contiki-NG – Process interactions

- System defined events that can be passed to the receiving process as second argument of process_post() or process_post_synch().

Event	ID	Description
PROCESS_EVENT_NONE	0x80	No event.
PROCESS_EVENT_INIT	0x81	Delivered to a process when it is started.
PROCESS_EVENT_POLL	0x82	Delivered to a process being polled.
PROCESS_EVENT_EXIT	0x83	Delivered to an exiting a process.
PROCESS_EVENT_SERVICE_REMOVED	0x84	Unused.
PROCESS_EVENT_CONTINUE	0x85	Delivered to a paused process when resuming execution.
PROCESS_EVENT_MSG	0x86	Delivered to a process upon a sensor event.
PROCESS_EVENT_EXITED	0x87	Delivered to all processes about an exited process.
PROCESS_EVENT_TIMER	0x88	Delivered to a process when one of its timers expired.
PROCESS_EVENT_COM	0x89	Unused.
PROCESS_EVENT_MAX	0x8a	The maximum number of the system-defined events.

francesca.righetti@ing.unipi.it

Contiki-NG – Process interactions

- In addition to the system defined events, custom events can be defined.
  - Declare the variable associated to the custom event

    ```
    static process_event_t evento_di_test;
    ```

  - Allocate the identifier associated to the event

    ```
    evento_di_test = process_alloc_event();
    ```

  - The event can be posted and received

    ```
    process_post(&process, evento_di_test, &data);
    PROCESS_WAIT_EVENT_UNTIL(ev == evento_di_test );
    ```

IoT

Contiki-NG – Process example

```
PROCESS_THREAD(key_sampling, ev, data)
{
  PROCESS_BEGIN();
  static struct etimer et;
  static int previous_key_value = 0;
  static char debounce_check = 0;
  int current_key_value;

  etimer_set(&et, CLOCK_SECOND / 50);
  while(1) {
    PROCESS_WAIT_EVENT_UNTIL((ev == PROCESS_EVENT_TIMER) || (ev == PROCESS_EVENT_MSG));
    if(ev == PROCESS_EVENT_TIMER) {
      /* Handle sensor reading.   */
      PRINTF("Key sample\n");
      current_key_value = get_key_value();
      if(debounce_check != 0) {
        /* Check if key remained constant */
        if(previous_key_value == current_key_value) {
          sensors_changed(&button_sensor);
          key_value = current_key_value;
          debounce_check = 0;
        } else {
          /* Bouncing */
          previous_key_value = current_key_value;
        }
      } else
      /* Check for new key change */
      if(current_key_value != previous_key_value) {
        previous_key_value = current_key_value;
        debounce_check = 1;
      }
      etimer_reset(&et);
    } else {
      /* ev == PROCESS_EVENT_MSG */
      if(*(buttons_status_t *)data == BUTTONS_STATUS_NOT_ACTIVE) {
        /* Stop sampling */
        etimer_stop(&et);
      } else if((*(buttons_status_t *)data == BUTTONS_STATUS_ACTIVE)) {
        /* restart sampling */
        etimer_restart(&et);
      }
    }
  }
  PROCESS_END();
}
```

- Example of a Process that handles different events

Do something as the timer has expired

Do something as a message has been received

IoT

System update

francesca.righetti@ing.unipi.it

Contiki-NG – System updates

- Update the docker image of Contiki-NG
  - Open a terminal and type:
    - docker pull contiker/contiki-ng

- Update the source code of Contiki-NG
  - Open a terminal and type:
    - cd
    - sudo rm -r ~/contiki-ng
    - git clone https://github.com/contiki-ng/contiki-ng.git
    - cd contiki-ng
    - git submodule update --init --recursive

IIoT

Contiki-NG – Cooja

- To run Cooja (it will run only **inside** the container):
  - Every time you start the container, a new one will be created. In order to use always the same container and avoid to lose history:
    - Check the Container ID, if the container is running with the command
      - docker ps
      - docker ps –a (if the container is not running)
    - If there are many containers, you can delete them, to avoid confusion:
      - docker container prune

```
osboxes@osboxes:~$ docker ps -a
CONTAINER ID   IMAGE                COMMAND                  CREATED          STATUS                     PORTS     NAMES
0e6b06861621   contiker/contiki-ng  "/bin/sh -c 'bash --…"   34 hours ago     Exited (0) 2 minutes ago             charming_lederberg
osboxes@osboxes:~$
```

    - After the first execution of the container, in order to start always the same one:
      - docker start –ai CONTAINER ID

Contiki-NG – More shells, same container

- To run another shell with the container you are using
    - Check the Container ID, the container is running for sure
        - docker ps
        - docker exec –it CONTAINER ID /bin/bash

francesca.righetti@ing.unipi.it

Contiki-NG
Cooja

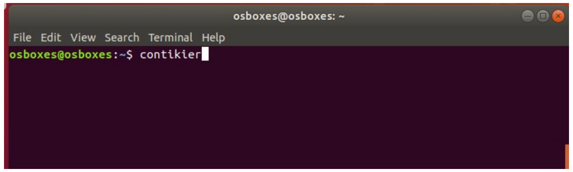
IoT

francesca.righetti@ing.unipi.it

Contiki-NG – Cooja

- Cooja is a **network simulation environment**
  - The hardware of many different motes is emulated
  - It allows to test the behavior of motes in large and small networks
  - Wireless connection among motes is simulated

- To run Cooja (it will run **only inside** the container):
  - Run the container:
    - Type the command: contikier
  - cd tools/cooja
  - Start Cooja with the following command:
    - **ant run**

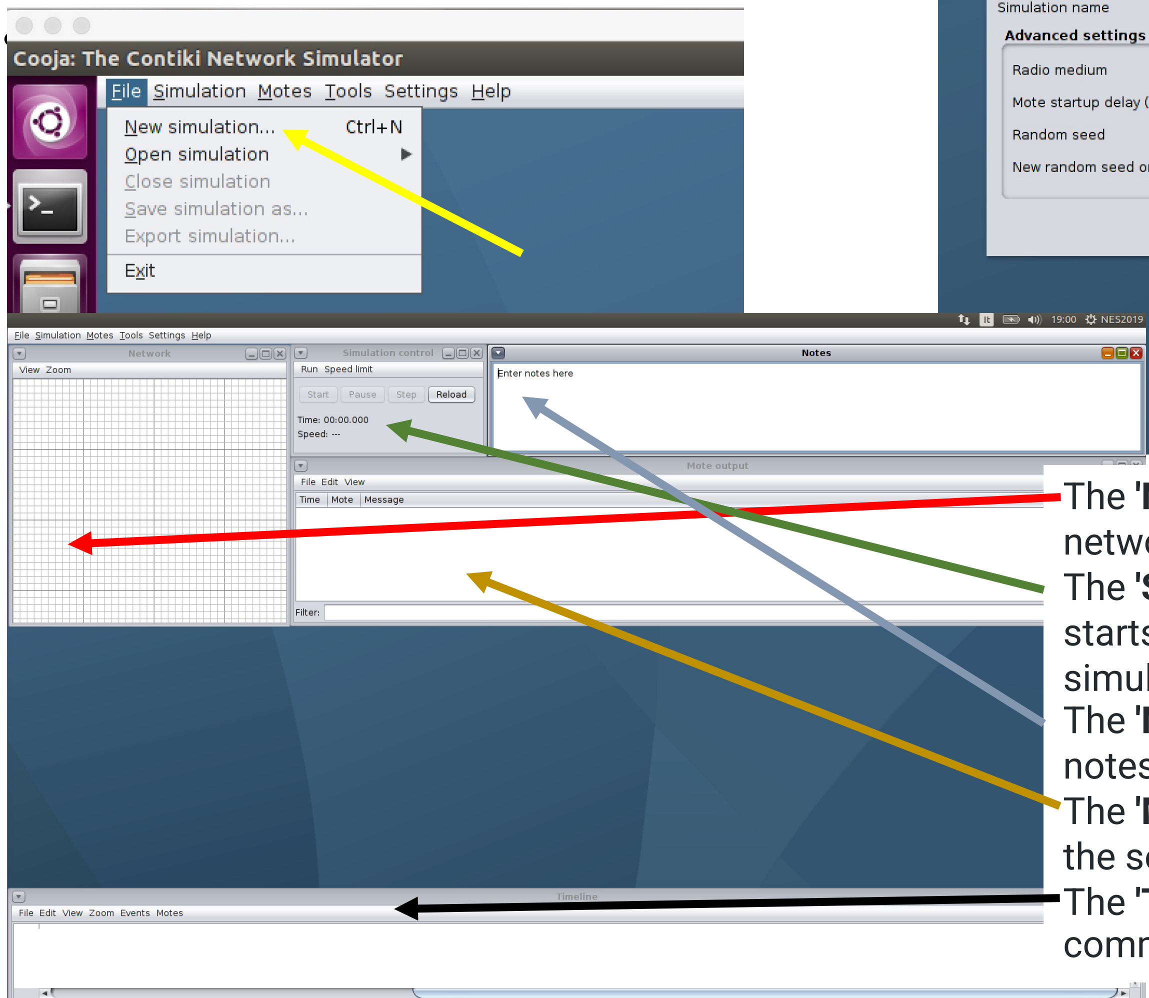


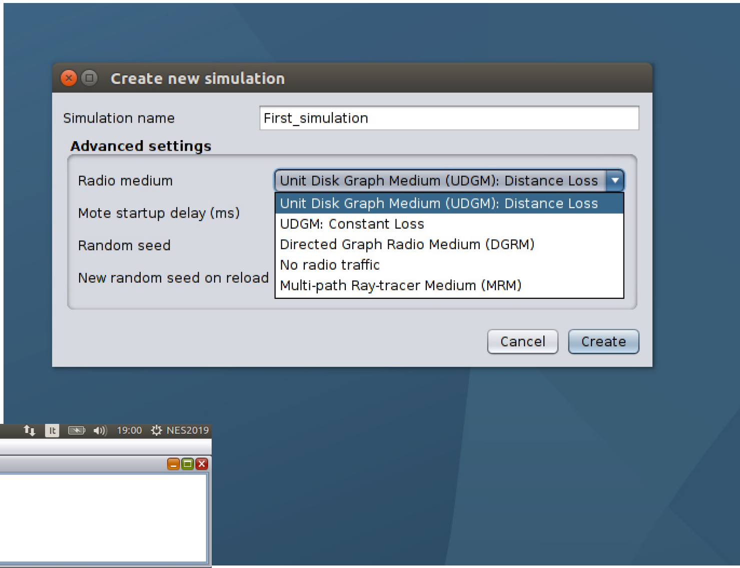
- NB: execute the contikier command only for the very first time to launch the container, afterwards use the “docker start..” command. More details on the next slides.

francesca.righetti@ing.unipi.it

Contiki-NG – Cooja

- How to create a new simulation



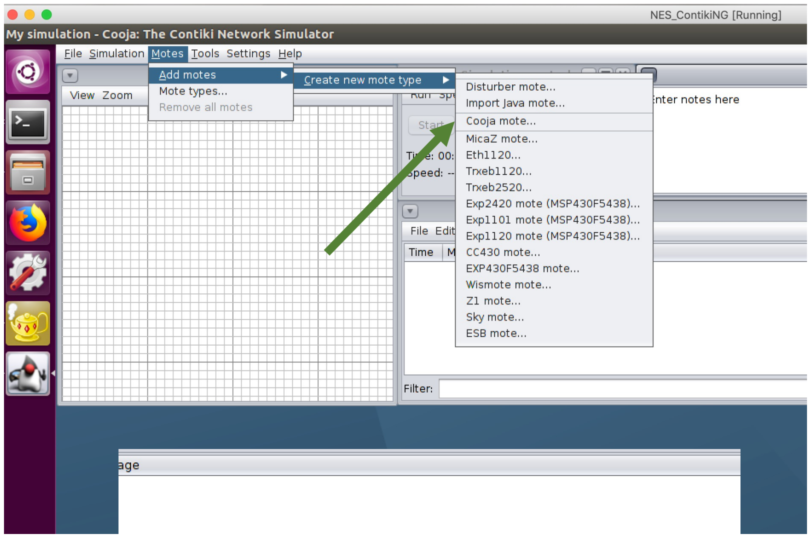


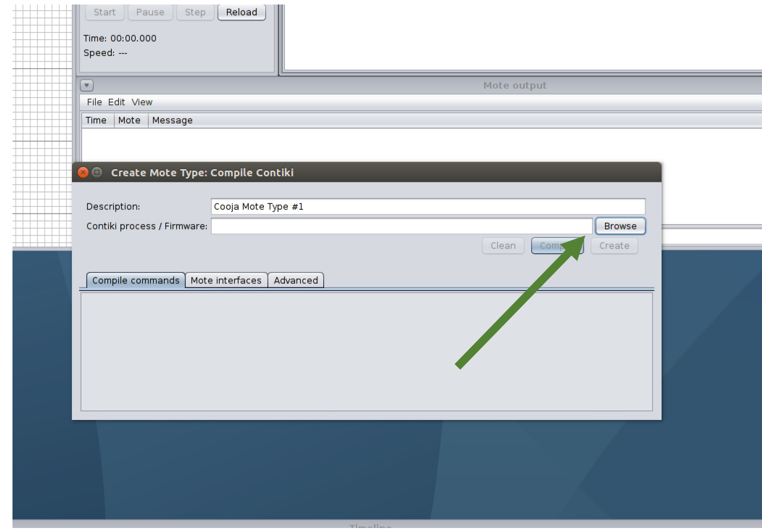
The '**Network**' panel shows the network topology.
The '**Simulation control**' panel starts/pauses/reloads simulations.
The '**Notes**' panel is for your own notes.
The '**Mote output**' panel shows the serial line output of nodes.
The '**Timeline**' visualizes network communication.

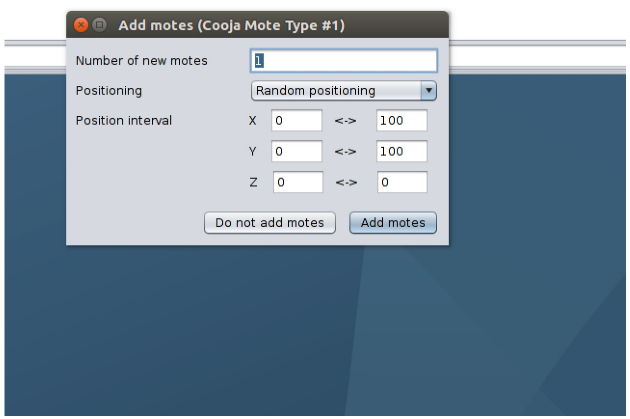
IoT

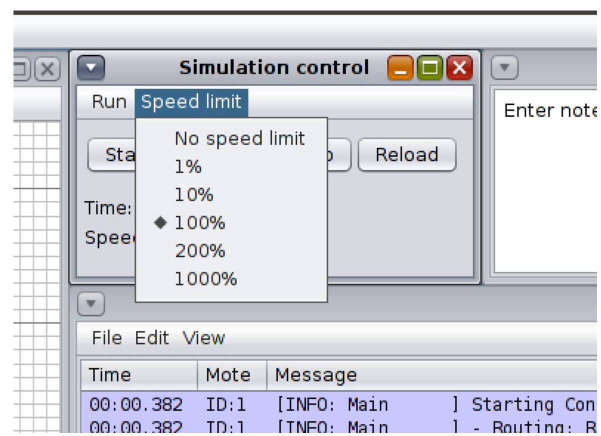
francesca.righetti@ing.unipi.it

Contiki-NG – Cooja









IoT

Contiki-NG – Cooja, the .csc file

- Save the simulation in the current directory.
- You will obtain a .csc file that describes the simulation environment settings.
- It can be modified in order to chang simulation scenario
- Add more nodes
- Modify the communication range… etc etc

```xml
<?xml version="1.0" encoding="UTF-8"?>
<simconf>
  <project EXPORT="discard">[APPS_DIR]/mrm</project>
  <project EXPORT="discard">[APPS_DIR]/mspsim</project>
  <project EXPORT="discard">[APPS_DIR]/avrora</project>
  <project EXPORT="discard">[APPS_DIR]/serial_socket</project>
  <project EXPORT="discard">[APPS_DIR]/powertracker</project>
  <simulation>
    <title>My simulation</title>
    <speedlimit>1.0</speedlimit>
    <randomseed>123456</randomseed>
    <motedelay_us>1000000</motedelay_us>
    <radiomedium>
      org.contikios.cooja.radiomediums.UDGM
      <transmitting_range>50.0</transmitting_range>
      <interference_range>100.0</interference_range>
      <success_ratio_tx>1.0</success_ratio_tx>
      <success_ratio_rx>1.0</success_ratio_rx>
    </radiomedium>
    <events>
      <logoutput>40000</logoutput>
    </events>
    <motetype>
      org.contikios.cooja.contikimote.ContikiMoteType
      <identifier>mtype91</identifier>
      <description>Cooja Mote Type #1</description>
      <source>[CONTIKI_DIR]/examples/hello-world/hello-world.c</source>
      <commands>make hello-world.cooja TARGET=cooja</commands>
      <moteinterface>org.contikios.cooja.interfaces.Position</moteinterface>
      <moteinterface>org.contikios.cooja.interfaces.Battery</moteinterface>
      <moteinterface>org.contikios.cooja.contikimote.interfaces.ContikiVib</moteinterface>
      <moteinterface>org.contikios.cooja.contikimote.interfaces.ContikiMoteID</moteinterface>
      <moteinterface>org.contikios.cooja.contikimote.interfaces.ContikiRS232</moteinterface>
      <moteinterface>org.contikios.cooja.contikimote.interfaces.ContikiBeeper</moteinterface>
      <moteinterface>org.contikios.cooja.interfaces.RimeAddress</moteinterface>
      <moteinterface>org.contikios.cooja.contikimote.interfaces.ContikiIPAddress</moteinterface>
      <moteinterface>org.contikios.cooja.contikimote.interfaces.ContikiRadio</moteinterface>
      <moteinterface>org.contikios.cooja.contikimote.interfaces.ContikiButton</moteinterface>
      <moteinterface>org.contikios.cooja.contikimote.interfaces.ContikiPIR</moteinterface>
      <moteinterface>org.contikios.cooja.contikimote.interfaces.ContikiClock</moteinterface>
      <moteinterface>org.contikios.cooja.contikimote.interfaces.ContikiLED</moteinterface>
      <moteinterface>org.contikios.cooja.contikimote.interfaces.ContikiCFS</moteinterface>
      <moteinterface>org.contikios.cooja.contikimote.interfaces.ContikiEEPROM</moteinterface>
      <moteinterface>org.contikios.cooja.interfaces.Mote2MoteRelations</moteinterface>
      <moteinterface>org.contikios.cooja.interfaces.MoteAttributes</moteinterface>
      <symbols>false</symbols>
    </motetype>
    <mote>
      <interface_config>
        org.contikios.cooja.interfaces.Position
        <x>65.45084971874738</x>
        <y>97.55282581475768</y>
        <z>0.0</z>
      </interface_config>
      <interface_config>
        org.contikios.cooja.contikimote.interfaces.ContikiMoteID
        <id>1</id>
      </interface_config>
      <interface_config>
        org.contikios.cooja.contikimote.interfaces.ContikiRadio
        <bitrate>250.0</bitrate>
```

IoT

francesca.righetti@ing.unipi.it

Contiki-NG – Hello World

```
#include "contiki.h"
#include <stdio.h>
/* Declare the process */
PROCESS(hello_world_process, "Hello world");
/* Make the process start when the module is loaded */
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 */
}
```

IoT

Contiki-NG – Hello World - Makefile

- The project includes a Makefile that specifies how to produce the binary code:

```
CONTIKI_PROJECT = hello-world
all: $(CONTIKI_PROJECT)

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

IoT

Contiki-NG – Hello World – project-conf.h

- An additional configuration file is usually included to override operating system default configurations

  - Add it to the Makefile

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

  - Example: change nodes' queue size

```
#undef QUEUEBUF_CONF_NUM
#define QUEUEBUF_CONF_NUM       32
```

IoT

francesca.righetti@ing.unipi.it

Contiki-NG – Excercise

- Go to the directory of the hello-world example (cd contiki-ng/examples/hello-world)

- Add the "project-conf.h" file
  - Use the header/footer of the figure
  - Add the macro to change queue size

project-conf.h

```c
#ifndef __PROJECT_CONF_H__
#define __PROJECT_CONF_H__





#endif
```

- Add the project-conf.h to the Makefile

- Open Cooja (1: contikier, 2: cd tools/cooja)

- Run Cooja

francesca.righetti@ing.unipi.it