Simulators for IoT Systems

Georgia Koutsandria

Internet of Things A.A. 18-19
Prof. Chiara Petrioli
Dept. of Computer Science
Sapienza University of Rome

PART I

Simulators for IoT Systems

What is a Simulator?

- A tool/software that realistically imitates/models the behavior of IoT systems.

- Different types of simulators; Most commonly used:
    - Trace-Driven Simulators
    - Discrete-Event Simulators

Why do we use Simulators?

- The most common approach to delelop and test new protocols/applications.

- Evulate the performance of new solutions.

- Consider a large-scale IoT network:
  - Low cost
  - Easy(?) to implement
  - Practical

Simulators for IoT Systems

- Several simulators exist:
  - ns-3/ns-2
  - OMNeT
  - Castalia
  - GreenCastalia
  - SUNSET
  - COOJA
  - Avrora
  - …

GreenCastalia: An energy harvesting-enabled simulator for IoT

What is GreenCastalia?

- An extension of the Castalia simulator.
- Allows to model and simulate networks of IoT devices, i.e., embedded devices, with energy harvesting capabilities.
- Castalia: Am OMNeT++ based simulator for WSNs, BANs, and networks of low-power embedded devices.
  - A realistic framework for fisrt order validation.
  - Not platform(device) specific.
  - Highly parametric.

How to install GC

- You will first need to install OMNET++
  - OMNET++ (recommended version 4.6): https://omnetpp.org
  - Castalia: https://github.com/boulis/Castalia

- Complete instructions:
  - http://senseslab.di.uniroma1.it/greencastaliav01d

How to install GC

- We strongly recommend that you use a Unix-based machine.

- Alternative Option: Download the VM (available link on twiki) with the GC simulator already installed on it (pwd: iot2018)
  - You will first need to install the VirtualBox software
    - https://www.virtualbox.org/wiki/Downloads

GreenCastalia

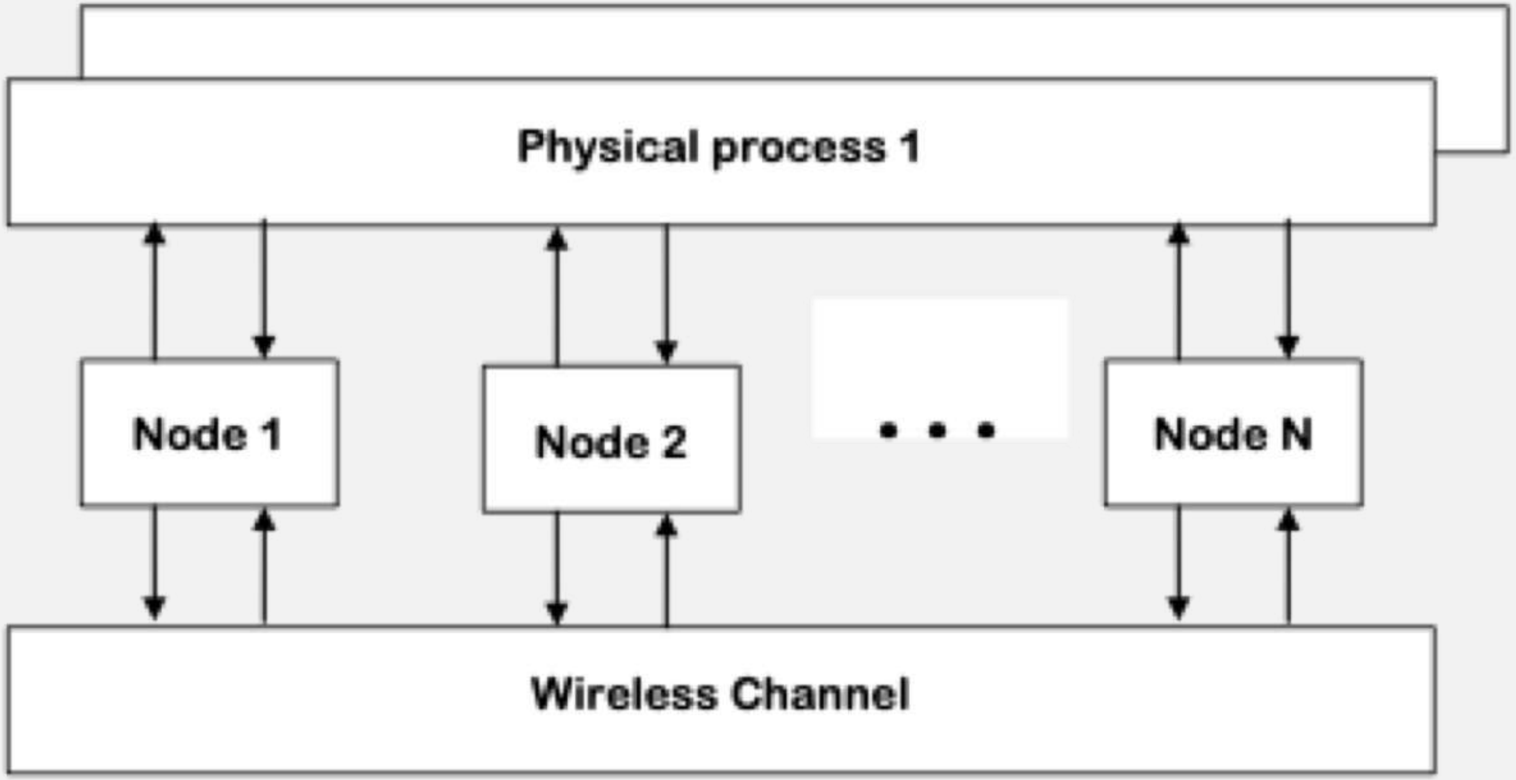
GreenCastalia: Main features

- Inherited by the Castaila simulator:
  - Channel model based on empirically measured data.
  - Radio model based on real traces for low-power communication.
  - Sensing modelling provisions.
  - MAC and routing protocols available.
  - Designed for adaption and expansion.

GreenCastalia: Main features

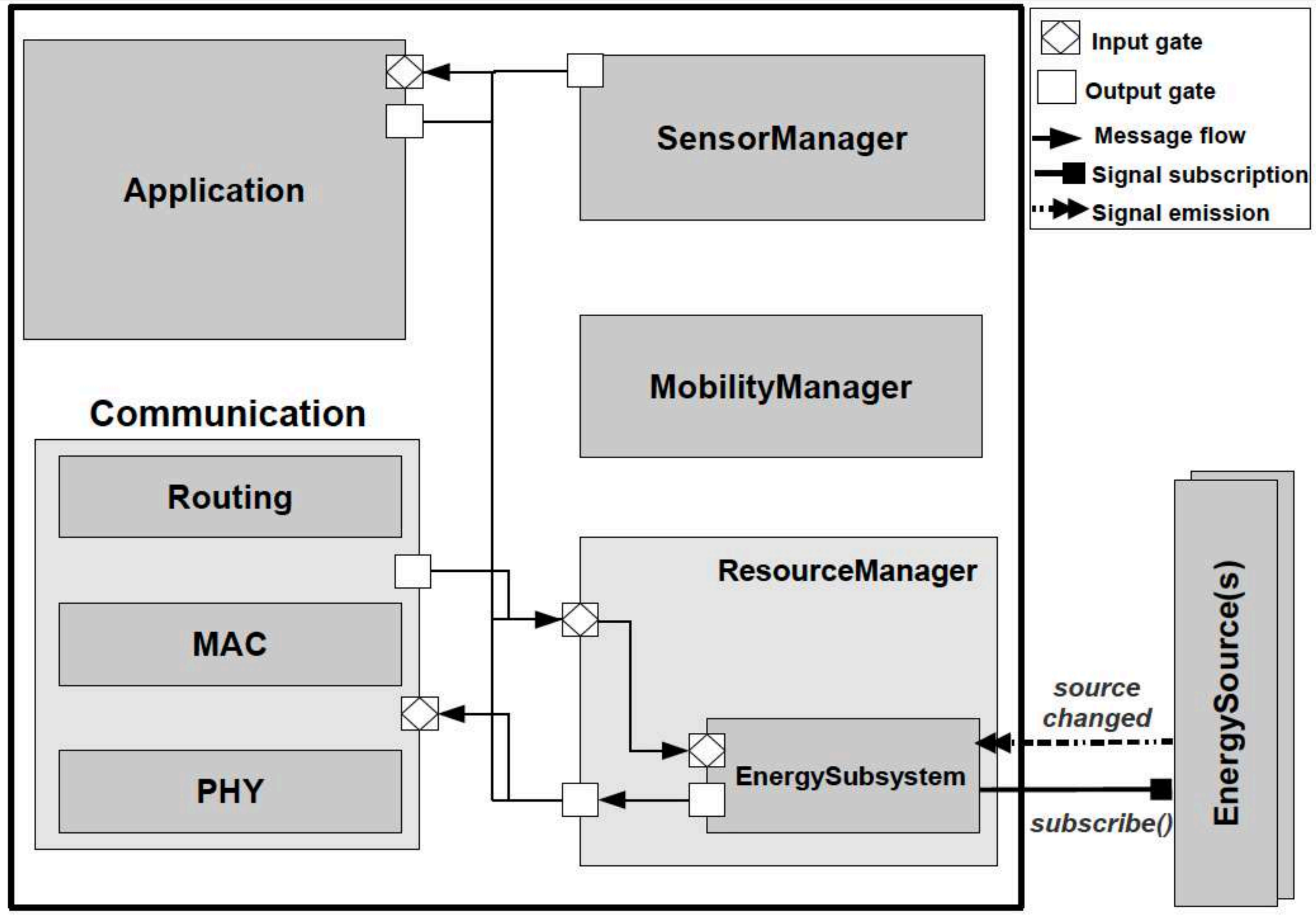
- GC-specific:
  - Multiple energy sources and multi-source harvesters.
  - Networks of embedded devices with heterogeneous harvesting and storage capabilities.
  - Multi-storage architectures (batteries, supercaps, rechargeable batteries).
  - Non-ideal battery models based on empirical discharge patterns, and supercaps leakage models.
  - Energy prediction models.

GreenCastalia Structure



GreenCastalia

SensorNode module



GreenCastalia

GC Organization

- Each module or submodule has its corrsponding directory.

- All reside in the directory `~/Castalia/src/`

- E.g.: Module node resides in the directory:
  `~/Castalia/src/node/`
  Module communication resides in the directory:
  `~/Castalia/src/node/communication/`
  Submodule routing resides in the directory:
  `~/Castalia/src/node/communication/routing`

GreenCastalia

GC Organization

- In the GC directory there is a folder named *Simulations*

  `~/Castalia/Simulations/`

- This folder includes:
  - Existing simulation examples with their simulation configuration files.
  - A subfolder named *Parameters*
    - Includes specially fromatted files with parameters that define the basic operational properties of specific modules (MAC, Radio, WirelessChannel, SensorDevice, PhysicalProcess).

Building GreenCastalia

- (Re)Build GC by using the following commands at the top-most GC directory `~/Castalia/`

```
make clean
./makemake
make
```

- After the creation of new files or any modifications in existing ones, rebuild GC using the same commands.

Using GreenCastalia

- Files with the suffice «.ned» contain NED language code
  - Define a module’s name and interfaces (gates in/out)
  - Define parameters

- Module directories always contain a «.ned» file defining them

- Simple modules include C++ code (.cc and .h files) defining their behavior

- Composite modules, e.g., node, include subdirectories to define the submodules.

Simulation Configuration File

- All simulation examples/tests reside in the directory `~/Castalia/Simulations`
- Configuration file typically named `omnetpp.ini`
    - Assigns values to parameters; Defines the simulation scenario.
    - The following file should be always included in the configuration file
        - `include ../Parameters/Castalia.ini`
        - It containes basic parameter assignment.
    - Defines the simulation time
    - Parameters always start with `SN` (sensor network: the top-most composite module)

GreenCastalia

Simulation Configuration File

```
[General]

include ../Parameters/Castalia.ini

sim-time-limit = 100s

SN.field_x = 200 #meters
SN.field_y = 200 #meters
```

Simulation Configuration File

- Defining the area of deployment using the parameter
  `SN.deployment`

- Several options:
  - uniform: random uniform distribution
  - NxM: nodes are placed in a NxM grid area
  - NxMxK: 3D dimension; nodes are placed in a NxMxK grid area
  - randomized_NxM: nodes are randomly places to NxM grid
  - Randomized_NxMxK: nodes are randomly places to NxMxK grid
  - center: nodes are placed in the center of the deployment area

Simulation Configuration File

- The sensor network compound module (`SN`) contains many Node sub-modules.

- Sub-modules are addressed in the form of an array.

- Assigning values to multiple nodes:
  - `[*]`: all indexes
  - `[3..5]:` indexes 3,4,5
  - `[..4]:` indexes 1, 2, 3, 4
  - `[5..]:` indexes 5 till last one

Running a simulation

- How to use the Castalia input script
  - `../../bin/Castalia -h`
- Available configurations
  - `../../bin/Castalia`
- Run a simulation using a specific configuration

  ```
  ../../bin/Castalia -c General
  ```
- Two files created in the directory
  1. `YYMMDD-HHMMSS.txt:` Output file which includes results.
  2. `Castalia-Trace.txt:` Contains traces of all events requested.

The `CastaliaResults` script

- Directory:
  `~/Castalia/bin/CastaliaResults/`
- `CastaliaResults`
  - Full list of Castalia output files with information about the configurations and the creation date.
  - Number of repetitions is indicated in the parenthesis.
  - `CastaliaResults -i YYMMDD-HHMMSS.txt`
    - Parses the given file and finds out what output was recorded by the different modules.

The `CastaliaResults` script

- Use the -s switch to select among outputs, e.g., packets;
  Results are the average of all modules and indices.

```
../../bin/CastaliaResults -i YYMMDD-HHMMSS.txt -s packets
```

- Get the sum of all nodes

```
../../bin/CastaliaResults -i YYMMDD-HHMMSS.txt -s packets -sum
```

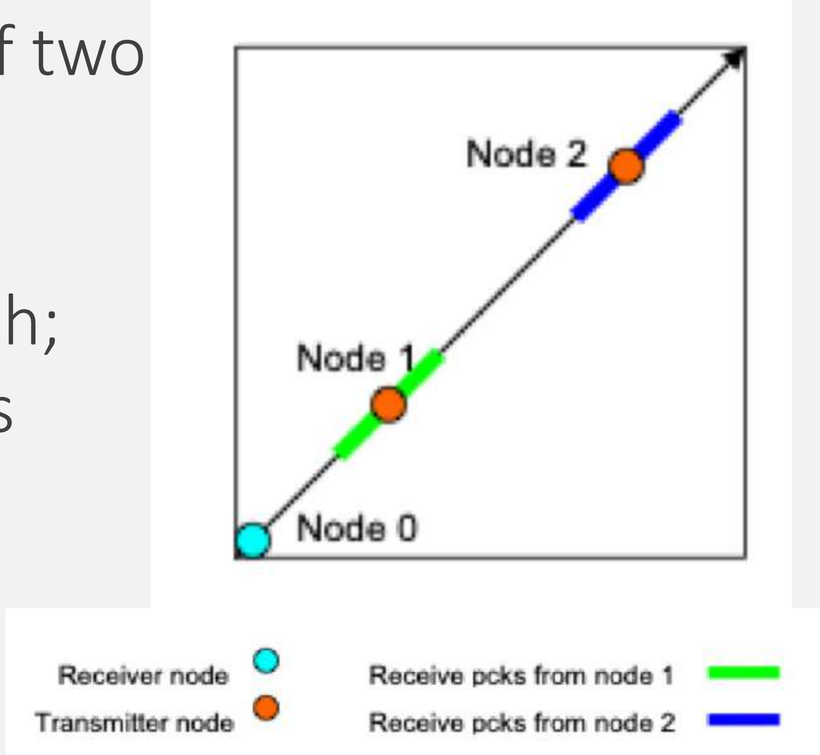
- Get per node results

```
../../bin/CastaliaResults -i YYMMDD-HHMMSS.txt -s packets -n
```

Simulation: An Example

- Go to `~/Castalia/Simulations/radioTest`

- Scenario: General (Tests reception)
  - A receiver (node 0) moves through the area of two transmitters (nodes 1 and 2).
  - No interference between transmitters.
  - Receiver moves in a straight line back and forth;
  - The receiver should receive packets when it is close to each of the two transmitters.



GreenCastalia

Simulation: An Example

- Type the following commands:

```
1. rm 1*.txt
2. rm Castalia-Trace.txt
```

- Run a simulation using the default configuration

```
../../bin/Castalia -c General
```

- Two files created in the directory
  1. `YYMMDD-HHMMSS.txt:` Output file which includes results.
  2. `Castalia-Trace.txt:` Contains traces of all events requested.

Questions?

Internet of Things A.A. 18-19