GreenCastalia: An Energy-Harvesting-Enabled Framework for the Castalia Simulator

User Manual

Dora Spenza, David Benedetti

SensesLab
Computer Science Department
Sapienza University of Rome
http://senseslab.di.uniroma1.it/greencastalia

Version 0.1d
Last updated: **August 2015**
Initial version: December 2013

Contents

1 Introduction

GreenCastalia [1] is an extension for the popular Castalia simulator [2] that allows to model and simulate networks of embedded devices with energy-harvesting capabilities.
The main features of GreenCastalia are:

- support for multiple energy sources and multi-source harvesters;
- support for networks of embedded devices with heterogeneous harvesting and storage capabilities;
- support for multi-storage architectures consisting of a combination of disposable batteries, supercapacitors and rechargeable batteries;
- support for non-ideal battery models based on empirical discharge patterns, and of supercapacitor leakage models;
- support for energy prediction models.

2 Installation

GreenCastalia is based on OMNeT++ and is an extension of the Castalia simulator.
We assume the reader to be familiar with the OMNeT++ environment and to have a good understanding of the basic concepts and structure of Castalia. The Castalia User Manual is available from https://github.com/boulis/Castalia, while documentations and tutorials about OMNeT++ can be found at http://www.omnetpp.org/documentation.
GreenCastalia has been developed and tested using Castalia 3.3 and OMNeT++ 4.3.1. Although OMNeT++ is available for Windows systems, Castalia has been designed for GNU/Linux-like systems. For this reason we strongly recommend to used a GNU/Linux-like system to use GreenCastalia. However, installation for Windows systems is possible using the Cygwin environment.
NEW The latest release GreenCastalia v0.1d has also been tested with OMNeT++ 4.6.

The following steps assume you have a working version of Castalia 3.3 installed on your system in a directory named Castalia. Please refer to the excellent Castalia Installation Guide https://github.com/boulis/Castalia to install Castalia and OMNeT++.

To install GreenCastalia, copy the archive in the Castalia root directory:

```
$ cp GreenCastalia-v0.1c.tar.bz2 Castalia/
$ cd Castalia
```

If you download Castalia from github, be sure to copy the GreenCastalia archive in the **inner** Castalia directory, not in the outer one:

```
$ cp GreenCastalia-v0.1c.tar.bz2 Castalia/Castalia
$ cd Castalia/Castalia
```

The correct directory is the one in which the makemake script file is located.

Untar and unzip the archive:

```
$ tar -xjvf GreenCastalia-v0.1c.tar.bz2
```

IMPORTANT! The following two files must be removed from the Castalia source tree before compiling GreenCastalia:

```
$ rm src/node/resourceManager/ResourceManager.h
$ rm src/node/resourceManager/ResourceManager.cc
```

You are now ready to build Castalia with GreenCastalia:

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

3 Overview

In this section, we present an overview of GreenCastalia and describe its integration into Castalia. The overall structure of a node in GreenCastalia is shown in Figure 1.

3.1 Integration into Castalia

Energy management in Castalia is carried on by the *ResourceManager* module, which holds energy-specific parameters, such as the baseline power consumption of the mote and its initial energy budget. This module keeps track of the remaining energy by performing periodic updates of the energy spent by the node over time. Energy updates are also triggered on-demand by Castalia modules that model hardware components whenever their power consumption changes. The amount of remaining energy is computed by the *ResourceManager* by modeling an ideal primary battery that is linearly discharged.
GreenCastalia extends the energy model of Castalia by introducing a new compound module, called *EnergySubsystem*, that completely replaces the *ResourceManager* module for what concerns energy management. To allow easy integration with Castalia, in GreenCastalia the *ResourceManager* is a compound module that includes the new *EnergySubsystem* module (Fig. 1). The new interface for the *ResourceManager* module includes parameters such as the baseline power consumption of the node (in mW), the cutoff voltage (in V), and the frequency of energy updates (in msec).

3.2 The EnergySubsystem module

The *EnergySubsystem* implements the support for energy harvesting at node level, by defining and managing submodules that represent energy harvesting and storage devices. More in details, it is composed by three main submodules:

- *EnergyHarvester*, which models the energy harvesting process and handles the corresponding devices (Section 4.1);
- *EnergyStorage*, which represent an energy storage device, i.e., a supercapacitor, a rechargeable battery or a disposable battery (Section 4.2);
- *EnergyManager*, which implements the control logic for storage utilization and charging (Section 4.3).

Environmental energy sources are modeled by the external *EnergySource* module (Fig. 2), of which multiple instances can be created to simulate multi-source harvesting systems. The current implementation

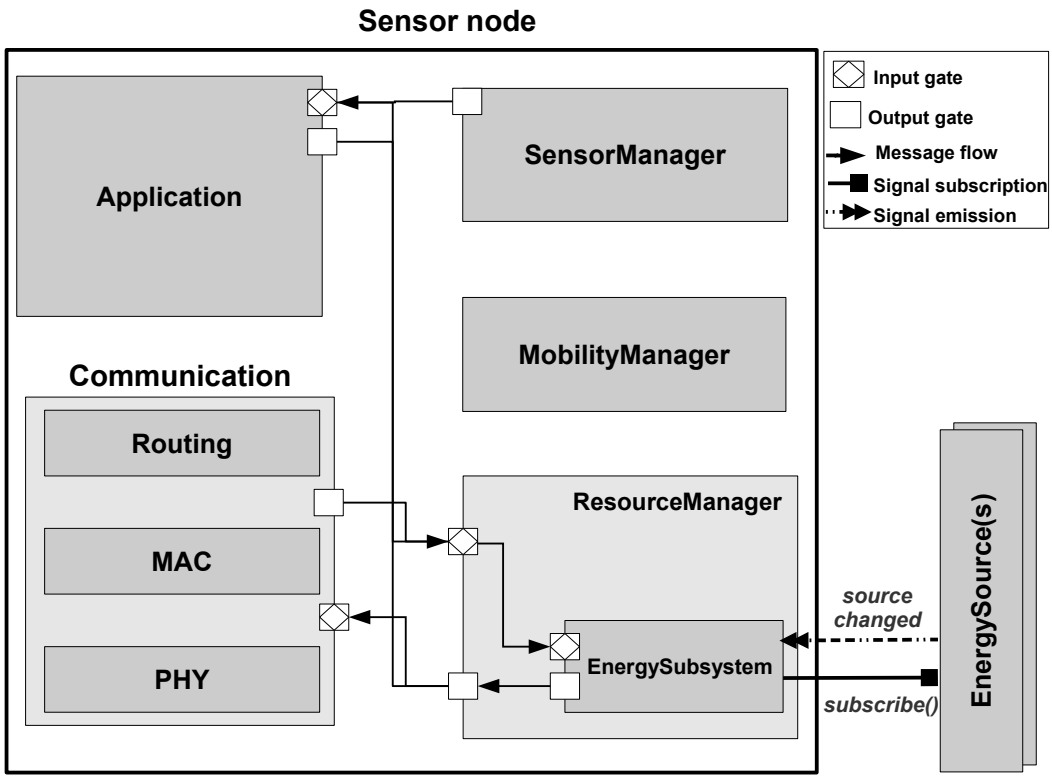

Figure 1: General structure of the SensorNode module in GreenCastalia.

of GreenCastalia provides a generic *TraceEnergySource* module, which allows to feed the simulator with timestamped power traces collected through real-life deployments and measurement studies [3, 4], or with energy availability traces obtained by data repositories [5] or meteorological stations [6]. This module can also be extended to support user-defined models of different energy sources. In GreenCastalia, each harvester is logically connected to one and only one energy source, while multiple harvesters can scavenge power from the same source. This allows to handle heterogeneous EH-WSNs, in which each node may have different harvesting capabilities.

4 Modeling in GreenCastalia

4.1 Energy harvesters

The *EnergyHarvester* module represents a physical harvesting device connected to a node. GreenCastalia currently provides an implementation of two simple models of energy harvesting devices (i.e., a solar cell and a wind-micro turbine), and of a generic harvester that scavenges power with a given efficiency from the energy source it is connected to.

The default interface for an energy harvester module allows to specify the type of the energy source is it connected to and the maximum amount of power it is able to generate. In addition, it is possible to optionally indicate a timestamped file that specifies how the harvesting efficiency of the device varies over time. This allows to model time-varying effects such as moving shadows, temporarily obstructions, changing in harvester orientation, and decreasing efficiency due to dust or aging. Finally, rather than being connected to an energy source, an harvesting device can also read energy availability traces from a

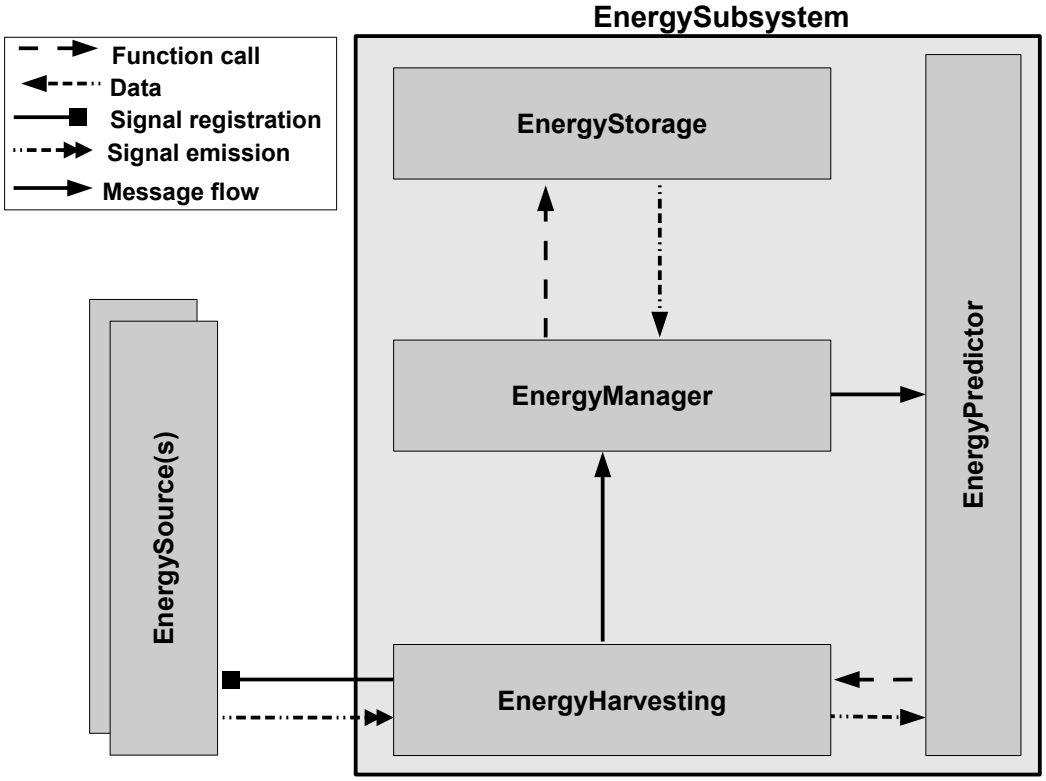

Figure 2: Architecture of the EnergySubsystem module.

file specified by the `harvesterTraceFile` parameter. This is especially useful to feed an harvester with a specific trace collected in a real-life deployment.
The interface from a `SolarCell` module additionally allows to specify the size of the solar cell (in cm^2) and its efficiency (which depends on the type of cell considered). This module is meant to be connected to a `TraceEnergySource` emitting irradiance values in unit W/m^2, or to read irradiance values in the same unit from the file specified by the `harvesterTraceFile` parameter.
The interface from a `WindTurbine` module allows to specify the rotor diameter (in cm), as well as air density and power coefficient.
It is worth noting that the current implementation of solar cell and wind micro turbine models **does not** take the storage level into account when computing the effective amount of energy harvested. The inclusion in GreenCastalia of more realistic harvester models is planned in the next release.

4.2 Energy storage

The *EnergyStorage* module represents a storage device that supplies energy to the node. The default interface for this module includes parameters such as charging and discharging efficiency, maximum rated voltage and initial charge fraction of the storage. GreenCastalia currently provides an implementation of two battery models and of a supercapacitor model:

- *IdealBattery*: a simple model in which the voltage of the battery remains constant over its lifetime and the discharge rate is always proportional to the power drawn from the battery. This is the default battery model provided by Castalia;

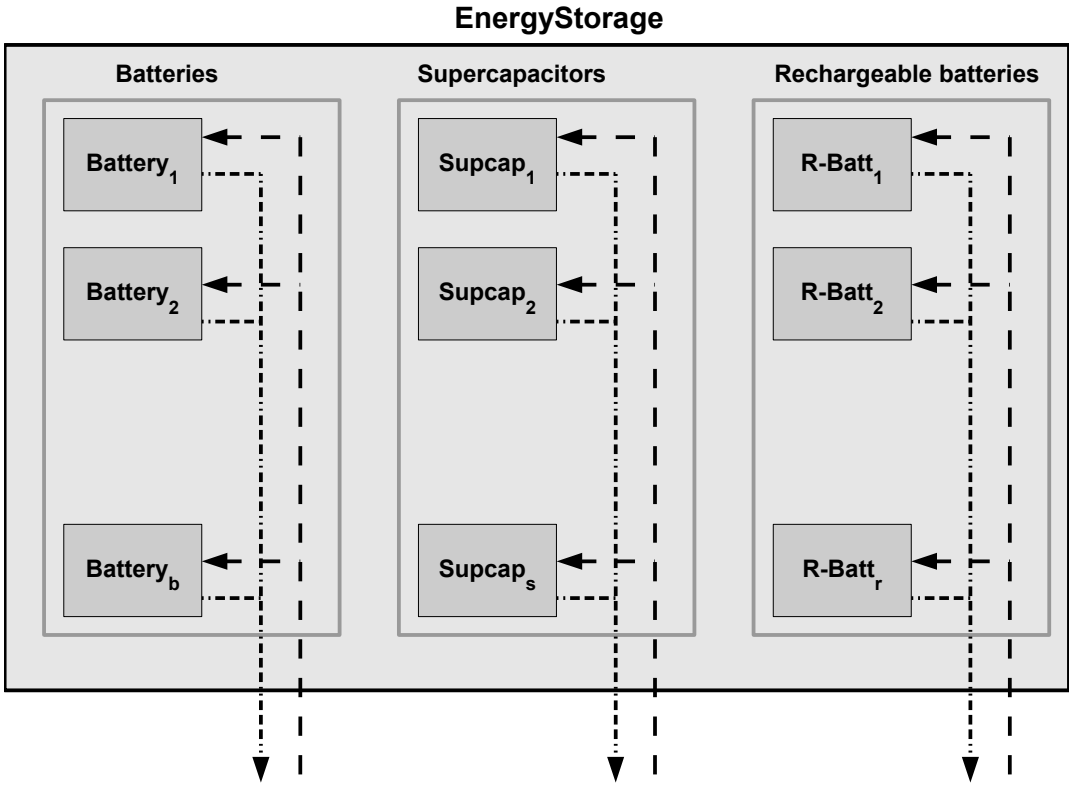

Figure 3: *EnergyStorages* module in GreenCastalia including a combinations of disposable batteries, supercapacitors and rechargeable batteries. Inward and outward arrows represent energy intake and provisioning.

- *EmpiricalBattery*: a model similar to *IdealBattery*, but including support to simulate the voltage of the battery decreasing over time based on its depth of discharge. The estimated voltage of the battery is computed by using a piecewise linear approximation of its empirical discharge pattern:

$$V(t) = \begin{cases} a_1 \cdot C(t) + b_1, & C_{R_1} \leq C(t) < C_{R_2} \\ \vdots & \vdots \\ a_n \cdot C(t) + b_n, & C_{R_n} \leq C(t) < C_{R_{n+1}} \end{cases}$$

  where $C_{R_1}, \ldots, C_{R_{n+1}}$ are the residual energy values in which the slope of the discharge curve changes significantly and $a_1, \ldots, a_n$, $b_1, \ldots, b_n$ are constants representing the coefficients of the line segments used for the approximation. An example of such approach is shown in Figure 4.

- *Supercapacitor*: a simple model in which the voltage of the supercapacitor is estimated based on the formula: $E(t) = \frac{1}{2}C(V(t))^2$, where $E(t)$ is the energy stored by the supercapacitor at time t, C is its rated capacity and $V(t)$ is the voltage across it.

As for *RechargeableBattery*, a simple model accounting for remaining battery cycles is provided, while more accurate models are under investigation.
EnergyStorage modules supply energy to the sensor node through the `discharge()` function called by the *EnergyManager* module. Supercapacitors and rechargeable batteries also implement the `charge()` function that is called by the *EnergyManager* module whenever there is an excess of harvesting energy

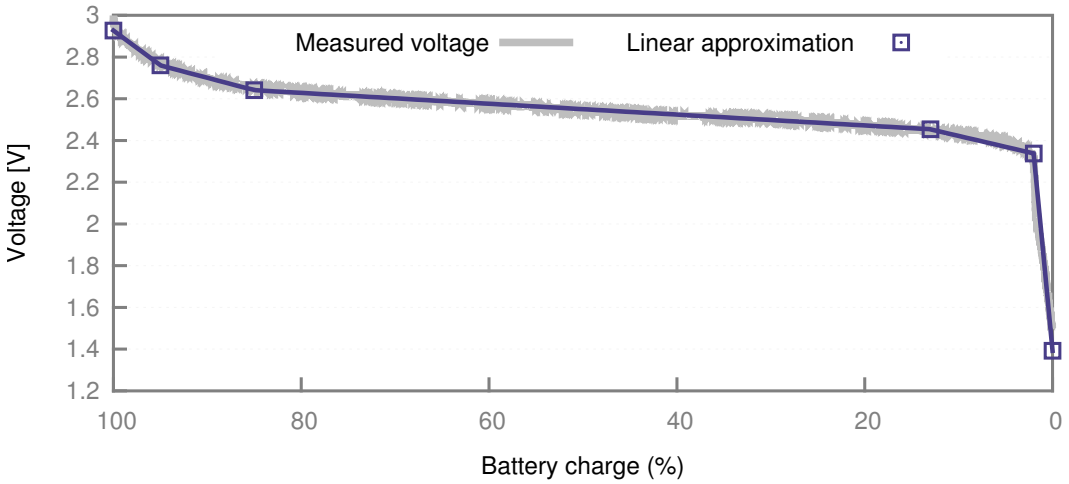

Figure 4: Piecewise linear approximation of battery empirical discharge pattern.

available. Each storage device can also implement a `selfDischarge()` function, which models the leakage and self-discharge effects suffered by charged supercapacitor and batteries. GreenCastalia currently provides a basic supercapacitor leakage model that can be used in simulations, serving as a starting point for future extension:

- *Constant current*: The leakage experienced by a charged supercapacitor is modeled as a constant current [7];

More accurate models, such as those recently proposed in [8, 9], can be integrated in GreenCastalia by extending the *Supercapacitor* module and its corresponding class.

4.3 Energy manager

The *EnergyManager* module is the core of the energy subsystem, having a complete view of the power harvested and drawn over time. It implements the control logic for storage devices utilization and charging, simulating the energy flow from harvesters and storage devices to the load, and from harvesters to storage devices. Moreover, it also keeps track of the energy wasted due to storage devices non-idealities, such as charging and discharging efficiency, self-discharge, and limited capacity, which may cause excess energy from harvesting to be lost. Energy updates are performed through the `energyUpdate` function in response to power consumption changes, which are triggered by modules modeling hardware components when their state changes. Updates are also triggered by *EnergyHarvesters*, which asynchronously notify the *EnergyManager* module of variations in their harvesting power, which can occur because of the dynamics of the energy source or due to temporary variations in the surrounding environment (e.g., moving shadows that impact on the amount of power generated by solar cells). In addition, the *EnergyManager* performs periodic updates with the period specified by the `periodicEnergyCalculationInterval` parameter. When the `energyUpdate` function is called, the *EnergyManager* module computes the net power consumption of the node, taking into account its current power consumption and harvesting rate. If the net power consumption is negative, the excess energy is used to recharge storage devices. Otherwise, storage devices are discharged to supply energy to the node. Devices handled by the *EnergyManager* can include a combinations of supercapacitors, rechargeable batteries and disposable batteries (Fig. 3). By extending the *EnergyManager* module, the specific controller for storage devices utilization and charging

can be modified so as to simulate a variety of architectures, including hybrid and multi-stage storage systems [10]. During energy updates, self-discharge of storage devices is also taken into account. Whenever the node runs out of energy, the *EnergyManager* notifies other modules by sending an `OUT_OF_ENERGY` message. If energy becomes available again, a `NODE_STARTUP` message is sent to simulates a node reset.

4.4 Support for energy predictions

Power-scavenging systems need to deal with the dramatic changes of energy availability over time. In the case of predictable energy sources, such as solar light, energy prediction models are a precious tool to devise smart energy allocation strategies. In fact, by forecasting the expected energy intake in the near future, proactive power management strategies can be implemented to optimize the utilization of the available energy. Acknowledging the importance of energy predictions for the design of harvesting-aware protocols, we included in GreenCastalia an *EnergyPredictor* module. The interface of this module allows to define the prediction horizon, the number of timeslots per day and the frequency with which the energy source is sampled within each timeslot. GreenCastalia currently implements the widely used energy prediction model EWMA [7], based on an exponentially weighted moving-average filter, and includes an advanced expectation model of the amount of harvested energy that can account for temporary environmental conditions [11].
This basic module can be extended to implement more sophisticated predictors [12, 13, 14].
EWMA maintains the history of the energy harvested on past days as an exponential moving average. Predictions are delivered based on the assumption that the energy available at a given time of the day is similar to the energy intake at the same time on the previous days. Time is discretized into N time slots of fixed length (usually 30 minutes each). The number of time slots used by the prediction model is determined based on parameter `slotSize` in `AEWMA.ned`, which specify the duration of each time slots in seconds. The history of the energy harvested on past days is maintained as an exponential moving average, in which the contribution of older data is exponentially decaying.
The EWMA model predicts that in time slot n the amount of energy

$$\mu_n^{(d)} = \alpha \cdot x_n + (1 - \alpha) \cdot \mu_n^{(d-1)}$$

will be available for harvesting, where:

x_n is the amount of energy harvested by the end of the nth slot;

$\mu_n^{(d-1)}$ is the average over the previous $d-1$ days of the energy harvested in their nth slot, and

α is a weighting factor, $0 \leq \alpha \leq 1$, whose value is defined by the `alpha` parameter in `AEWMA.ned`.

EWMA exploits the diurnal solar energy cycle and adapts to seasonal variations. The prediction results quite accurate in presence of scarce weather variability. However, when weather conditions are frequently changing (e.g., a mix of sunny and cloudy days in a row) EWMA does not adapt well to the variations in the solar energy profile.
In order to account for short-term varying weather conditions, the model proposed by Noh and Kang in [15] has also been implemented. Such model introduces a scaling factor φ_n to adjust future energy expectations. This factor is computed as: $\varphi_n = \frac{x_{n-1}}{\mu_{n-1}}$, where x_{n-1} is the amount of energy harvested by the end of slot $n-1$, and μ_{n-1} is the prediction of the amount of energy harvestable during slot $n-1$ according to the EWMA. Thus, φ_n expresses the ratio between the actual harvested energy at time slot n and the energy predicted for the same time slot. This scaling factor is then used to adjust future predictions:

$$\mu_{K\%N} = \left(\varphi_n + \frac{1 - \varphi_n}{N} \cdot (k-1)\right) \mu_{K\%N}, \quad n \leq k < N + i.$$

Since the scaling factor computed for time slot t is only valid temporarily, its influence decreases linearly for time slots that are far away in time.
To use this advanced model in GreenCastalia, the parameter `useAdvancedModel` must be set to true in `AEWMA.ned`.

5 Software organization

GreenCastalia OMNeT++ modules (NED/C++ files) are located in the `src/energySource` and in the `src/node/resourceManager/energySubsystem` sub-folders of Castalia.
The structure of the GreenCastalia source subtree is the following:

```
src/
├── energySource/ ..... Definition of EnergySource modules
│   └── traceEnergySource/ ..... TraceEnergySource module
└── node/
    └── resourceManager/
        └── energySubsystem/
            ├── energyHarvester/ ..... Definition of EnergyHarvester modules (Sec. 4.1)
            │   ├── solarCell/ ..... SolarCell module
            │   ├── windTurbine/ ..... WindTurbine module
            │   └── traceHarvester/ ..... TraceHarvester module
            ├── energyManager/ ..... Definition of EnergyManager modules (Sec. 4.3)
            ├── energyStorage/ ..... Definition of EnergyStorage modules (Sec. 4.2)
            │   ├── supercapacitor/ ..... Supercapacitor module
            │   ├── rechargeableBattery/ ..... RechargeableBattery module
            │   └── battery/ ..... Battery module
            └── energyPrediction/ ..... Definition of EnergyPredictor modules (Sec. 4.4)
                └── aewma/ ..... AEWMA energy prediction module
```

6 Using GreenCastalia

In this section we provide some configuration examples that can be added to the `omnetpp.ini` file of a simulation to make use of GreenCastalia features.

6.1 Multi-storage architectures

GreenCastalia provides support for multi-storage architectures consisting of a combination of disposable batteries, supercapacitors and rechargeable batteries. The following snapshot shows the configuration that should be added to the `omnetpp.ini` file to define a network of nodes each equipped with a supercapacitor, a rechargeable battery and a primary battery.

```
# [EnergyStorage]
SN.node[*].ResourceManager.EnergySubsystem.EnergyStorage.numSupercaps = 1
SN.node[*].ResourceManager.EnergySubsystem.EnergyStorage.numRechBatteries = 1
SN.node[*].ResourceManager.EnergySubsystem.EnergyStorage.numBatteries = 1

# [Supercapacitor] Rated capacity: 100F, Rated voltage: 2.7V
SN.node[*].ResourceManager.EnergySubsystem.EnergyStorage.Supercapacitors[0].maxVoltage = 2.7
SN.node[*].ResourceManager.EnergySubsystem.EnergyStorage.Supercapacitors[0].capacitance = 100
```

```
# [Rechargeable battery] 2 x AA 1.2V 2450mAh
SN.node[*].ResourceManager.EnergySubsystem.EnergyStorage.RechBatteries[0].maxVoltage = 2.4
SN.node[*].ResourceManager.EnergySubsystem.EnergyStorage.RechBatteries[0].mAmpereHour = 2450

# [Battery] 2 x AA 1.5V 1800mAh
SN.node[*].ResourceManager.EnergySubsystem.EnergyStorage.Batteries[0].maxVoltage = 3
SN.node[*].ResourceManager.EnergySubsystem.EnergyStorage.Batteries[0].mAmpereHour = 1800
```

Note that is possible to define networks of nodes with heterogeneous storage capabilities. In the following example, all nodes are battery-powered, except for node 0 that is powered by a supercapacitor and a rechargeable battery.

```
# All nodes are battery-powered except from node 0
# [EnergyStorage]
SN.node[1..].ResourceManager.EnergySubsystem.EnergyStorage.numSupercaps = 0
SN.node[1..].ResourceManager.EnergySubsystem.EnergyStorage.numRechBatteries = 0
SN.node[1..].ResourceManager.EnergySubsystem.EnergyStorage.numBatteries = 1

SN.node[1..].ResourceManager.EnergySubsystem.EnergyStorage.Batteries[0].maxVoltage = 3
SN.node[1..].ResourceManager.EnergySubsystem.EnergyStorage.Batteries[0].mAmpereHour = 1800

# Node 0 is powered by a supercapacitor and a rechargeable battery
# [EnergyStorage]
SN.node[0].ResourceManager.EnergySubsystem.EnergyStorage.numSupercaps = 1
SN.node[0].ResourceManager.EnergySubsystem.EnergyStorage.numRechBatteries = 1
SN.node[0].ResourceManager.EnergySubsystem.EnergyStorage.numBatteries = 0

SN.node[0].ResourceManager.EnergySubsystem.EnergyStorage.Supercapacitors[0].maxVoltage = 2.7
SN.node[0].ResourceManager.EnergySubsystem.EnergyStorage.Supercapacitors[0].capacitance = 100

SN.node[0].ResourceManager.EnergySubsystem.EnergyStorage.RechBatteries[0].maxVoltage = 2.4
SN.node[0].ResourceManager.EnergySubsystem.EnergyStorage.RechBatteries[0].mAmpereHour = 2450
```

By default, storage devices are assumed to be ideal, i.e., having a round-trip efficiency of 100%. The charging and discharging efficiency of each device can be specified by using the `chargingEfficiency` and `dischargingEfficiency` parameters. The following snapshot of the `omnetpp.ini` file shows how to configure a supercapacitor with 95% efficiency.

```
SN.node[*].ResourceManager.EnergySubsystem.EnergyStorage.Supercapacitors[0].chargingEfficiency
    = 0.95
SN.node[*].ResourceManager.EnergySubsystem.EnergyStorage.Supercapacitors[0].
    dischargingEfficiency = 0.95
```

The fraction of charge of each device at the beginning of the simulation can be defined as follows:

```
# At the beginning of the simulation the supercapacitor battery is 20% charged
SN.node[*].ResourceManager.EnergySubsystem.EnergyStorage.Supercapacitors[0].
    fractionInitialCharge = 0.2

# At the beginning of the simulation the rechargeable battery is 75% charged
SN.node[*].ResourceManager.EnergySubsystem.EnergyStorage.RechBatteries[0].fractionInitialCharge
     = 0.75
```

If the fraction of charge at the beginning of the simulation is not specified, it is assumed to be equal to 1. Unless otherwise specified, GreenCastalia uses by default the ideal battery model when a battery is instantiated. The following configuration snapshot shows how to define a battery whose estimated voltage

is computed by using a piecewise linear approximation of its empirical discharge pattern:

```
SN.node[*].ResourceManager.EnergySubsystem.EnergyStorage.Batteries[0].batteryModel = "empirical
    "
SN.node[*].ResourceManager.EnergySubsystem.EnergyStorage.Batteries[0].empiricalDischargeFile =
    "path/to/empiricalDischargeFile"
```

The discharge pattern is specified by the `empiricalDischargeFile` parameter, which gives the filename of a specially formatted input file. The input file has lines of the following format:

C_{R_1} a_1 b_1
C_{R_2} a_2 b_2
...
C_{R_n} a_n b_n

where $C_{R_1}, C_{R_2}, \ldots C_{R_{n+1}}$ are the percentage charge level of the battery where the slope of the discharge curve changes significantly, and $a_1, \ldots, a_n$, $b_1, \ldots, b_n$ are constants representing the coefficients of the line segments used for the approximation.
This means that when the charge level of the battery, $C(t)$, is between C_{R_1} and C_{R_2}, its estimated voltage is computed as $a_1 \cdot C(t) + b_1$, when it is between C_{R_2} and C_{R_3} its estimated voltage is computed as $a_2 \cdot C(t) + b_2$, and so on. Please note that charge levels $C_{R_1}, C_{R_2}, \ldots C_{R_{n+1}}$ must be sorted in increasing order.
An example of such file is included in the GreenCastalia distribution:
`Simulation/Parameters/EnergySubsystem/constantDischarge.dat`.

6.2 Energy sources and energy harvesters

The following configuration snapshot shows how to instantiate two `TraceEnergySource` modules, i.e., a solar energy source reading irradiance values from file *NREL-1year.irradiance* and a wind energy source reading wind speed values from file *NREL-1year.speed*. Such traces of solar and wind energy availability were obtained from the National Renewable Energy Laboratory at Oak Ridge, Tennessee [6].

```
# Define a solar energy source
# Energy sources
SN.numEnergySources = 2

SN.energySource[0].description = "Solar"
SN.energySource[0].traceFile = "../Parameters/EnergySource/SolarTraces/NREL-1year.irradiance"

SN.energySource[1].description = "Wind"
SN.energySource[1].traceFile = "../Parameters/EnergySource/WindTraces/NREL-1year.speed"
```

In order to be able to harvest power from an energy source, sensor nodes must be equipped with at least one harvesting device:

```
# Node 0 harvests energy from solar light
SN.node[0].ResourceManager.EnergySubsystem.EnergyHarvesting.numEnergyHarvesters = 1
SN.node[0].ResourceManager.EnergySubsystem.EnergyHarvesting.Harvesters[0].typename = "SolarCell
    "

# Node 1 harvests energy through a wind micro turbine
SN.node[1].ResourceManager.EnergySubsystem.EnergyHarvesting.numEnergyHarvesters = 1
SN.node[1].ResourceManager.EnergySubsystem.EnergyHarvesting.Harvesters[0].typename = "
    WindTurbine"
```

6.3 Energy predictions

An energy predictor can be instantiated by assigning a valid type name to the `PredictorType` parameter of the energy subsystem module. The following configuration snapshot shows how to define a network of nodes using the EWMA energy predictor with alpha parameter set to 0.7.

```
SN.node[*].ResourceManager.EnergySubsystem.PredictorType = "AEWMA"
SN.node[*].ResourceManager.EnergySubsystem.EnergyPrediction.alpha = 0.7
```

The following code snapshot shows how to access energy prediction within an application:

```
#include "VirtualEnergyPredictor.h"

/* Obtain a pointer to the energy predictor module */
VirtualEnergyPredictor* predictorModule = check_and_cast<VirtualEnergyPredictor*>
(getParentModule()->getSubmodule("ResourceManager")->getSubmodule("EnergySubsystem")->
    getSubmodule("EnergyPrediction"));

/* Get the harvesting power predicted at some time predTime */
simtime_t predTime = ...;
double predHarvPwr = predictorModule->getPrediction( predTime );
```

7 Frequently Asked Questions

7.1 Get the current energy level from the routing module

The following code snapshot shows how to get information about the currently energy level from the routing module:

```
#include "VirtualEnergyManager.h"

/* Obtain a pointer to the energy manager module */
VirtualEnergyManager* engyMgr =
check_and_cast<VirtualEnergyManager*>(getParentModule()->getParentModule()->getSubmodule("
    ResourceManager")->getSubmodule("EnergySubsystem")->getSubmodule("EnergyManager"));

double currentEnergyRatio = engyMgr->getCurrentEnergyRatio();
```

References

[1] D. Benedetti, C. Petrioli, and D. Spenza, "Greencastalia: An energy-harvesting-enabled framework for the castalia simulator," in *Proceedings of the 1st International Workshop on Energy Neutral Sensing Systems*, ser. ACM ENSSys 2013. New York, NY, USA: ACM, 2013, pp. 7:1–7:6. [Online]. Available: http://doi.acm.org/10.1145/2534208.2534215

[2] A. Boulis, "Castalia: Revealing Pitfalls in Designing Distributed Algorithms in WSN," in *Proceedings of SenSys 2007*, New York, NY, USA, 2007, pp. 407–408.

[3] A. Cammarano, D. Spenza, and C. Petrioli, "Energy-harvesting WSNs for structural health monitoring of underground train tunnels," in *Proceedings of IEEE INFOCOM WKSHPS 2013*, April 2013, pp. 75–76.

[4] M. Gorlatova, M. Zapas, E. Xu, M. Bahlke, I. J. Kymissis, and G. Zussman, "CRAWDAD Data Set Columbia/Enhants," April 2011. [Online]. Available: http://crawdad.cs.dartmouth.edu/columbia/enhants

[5] "EH Network Data Repository." [Online]. Available: http://eh-network.org/data

[6] "NREL: Measurement and Instrumentation Data Center," 2011. [Online]. Available: http://www.nrel.gov/midc

[7] A. Kansal, J. Hsu, S. Zahedi, and M. Srivastava, "Power Management in Energy Harvesting Sensor Networks," *ACM Transactions in Embedded Computing Systems*, vol. 6, no. 4, p. Article 32, September 2007.

[8] A. Kailas, M.-A. Ingram, and D. Brunelli, "A Simple Energy Model for the Harvesting and Leakage in a Supercapacitor," in *Proceedings of IEEE ICC 2012*, Ottawa, Canada, June 2012, pp. 6278–6282.

[9] G. Merrett and A. Weddell, "Supercapacitor Leakage in Energy-Harvesting Sensor Nodes: Fact or Fiction?" in *Proceedings of EnHaNSS 2012*, Antwerp, Belgium, June 2012.

[10] A. Nasiri, S. Zabalawi, and G. Mandic, "Indoor Power Harvesting Using Photovoltaic Cells for Low-Power Applications," *IEEE Transactions on Industrial Electronics*, vol. 56, no. 11, pp. 4502–4509, 2009.

[11] D. K. Noh and K. Kang, "Balanced energy allocation scheme for a solar-powered sensor system and its effects on network-wide performance," *J. Comput. Syst. Sci.*, vol. 77, no. 5, pp. 917–932, Sep. 2011.

[12] J. Piorno, C. Bergonzini, D. Atienza, and T. Rosing, "Prediction and Management in Energy Harvested Wireless Sensor Nodes," in *Proceedings of Wireless VITAE 2009*, Aalborg, Denmark, May 2009, pp. 6–10.

[13] A. Cammarano, C. Petrioli, and D. Spenza, "Pro-Energy: A novel energy prediction model for solar and wind energy-harvesting wireless sensor networks," in *Proceedings of IEEE MASS 2012*, Las Vegas, Nevada, Oct 2012, pp. 75–83.

[14] A. Cammarano, C. Petrioli, and D. Spenza, "Improving Energy Predictions in EH-WSNs with Pro-Energy-VLT," in *Proceedings of ACM SenSys 2013, Poster Session.* New York, NY, USA: ACM, 2013, pp. 41:1–41:2.

[15] D. Noh and K. Kang, "Balanced Energy Allocation Scheme for a Solar-Powered Sensor System and its Effects on Network-Wide Performance," *Journal of Computer and System Sciences*, vol. 77, no. 5, pp. 917–932, September 2011.