Simple Network Simulator (sim2net) Documentation

Release

Michal Kalewski

June 14, 2014

Contents

Simple Network Simulator (**sim2net**) is a discrete event simulator of *mobile ad hoc networks* (MANETs) implemented in Python (version 2.7). The simulator allows us to simulate networks of a given number of nodes that move according to the selected mobility model, run custom applications, and communicate only by sending application messages through wireless links.

CHAPTER 1

Installation

There are two possibilities to install the **sim2net** simulator: with the use of the `pip` installation tool, or from the source code obtained from GitHub.

1.1 1. Using the `pip` installation tool

```
$ sudo pip install sim2net
```

1.2 2. Manually from the source code

Step 1. Clone the project:

```
$ git clone git@github.com:mkalewski/sim2net.git sim2net
$ cd sim2net
```

Step 2. Run install:

```
$ sudo python setup.py install
```

CHAPTER 2

“Hello World” example

```
$ sim2net -i .
$ sim2net ./configuration.py ./application.py
```

CHAPTER 3

Contents

3.1 Command-line interface

This package provides a command-line interface for the **sim2net** simulator, which allows users to initialize and start simulations.

3.1.1 Synopsis

`sim2net` – a console script to initialize and start simulations:

```
sim2net [-h | -d | -v | -i DIRECTORY] CONFIGURATION APPLICATION

positional arguments:
  CONFIGURATION         simulation configuration file
  APPLICATION           simulation application file

optional arguments:
  -h, --help            show this help message and exit
  -d, --description     show description message and exit
  -i DIRECTORY, --initialize DIRECTORY
                        write configuration and application files to given
                        directory
  -v, --version         show version message and exit
```

3.1.2 Description

To start a simulation with the **sim2net** simulator, two files are necessary: a configuration file (with the simulator settings) and an application file that is run by every node in the simulated network (the application must implement the `sim2net.application.Application` abstract class). The easiest way to obtain both files is to execute the `sim2net` command with the `-i` option, eg.:

```
sim2net -i .
```

After that, two files are created in the given directory: `configuration.py` and `application.py`. Both files may be edited – for more information about configuration parameters see *Packages* section, and for more information about application implementation see the `sim2net.application.Application` abstract class.

Next, to start the simulation, the `sim2net` command should be executed with both files as arguments, eg.:

```
sim2net ./configuration.py ./application.py
```

See also:

Packages, sim2net.application.Application

3.2 sim2net.application.Application abstract class

class sim2net.application.**Application**

Bases: object

failure(*time*, *shared*)

finalize(*shared*)

initialize(*node_id*, *shared*)

main(*time*, *communication*, *neighbors*, *shared*)

3.2.1 Default configuration

test.

3.3 Packages

Simple Network Simulator (**sim2net**) – a discrete-event simulation of mobile ad hoc networks (MANETs).

3.3.1 Package sim2net

This package provides modules for the **sim2net** simulator.

The sim2net.simulator.Sim2Net class is the main entry point for conducting simulations, and the sim2net.application.Application abstract class defines the interface for simulation applications.

Package modules:

- Module sim2net._version
- Module sim2net._time
- Module sim2net._channel
- Module sim2net._network
- Module sim2net.simulator

Module sim2net._version

This package provides version information for the project.

The project's version number has the following form: *X.Y.Z*, where:

- *X* – is a major version number,
- *Y* – is a minor version number,

- *Z* – is a maintenance version number.

Each number is increased by one at a time. When one of the numbers is increased, the less significant numbers are reset to zero in the following way:

- if there are backwards incompatible changes then the major number is incremented and the minor and maintenance numbers are reset to zero;
- if there are new features (additions) implemented then the minor number is incremented and the maintenance number is reset to zero;
- if there are only implementation detail changes or bug fixes then the maintenance number is incremented (and there are no resets).

`sim2net._version.`**`get_version`**`()`
Returns the current version number as a string.

`sim2net._version.`**`project_information`**`()`
Returns the project information in the form of its name, short name, and the current version number as a string.

Module `sim2net._time`

Supplies time-related functionality for simulations.

In this module the following terminology is used:

***Simulation step*, s:** takes successive discrete values stating from 0 before each simulation iteration.

***Simulation time*, t_s:** keeps track of the current time for the system being simulated; it advances to the next value in accordance with a given *simulation frequency* before each simulation iteration.

***Simulation frequency*, f_s:** a constant that describes the relationship between the *simulation step* and the *simulation time* in the following manner: $t_s = \frac{s}{f_s}$.

***Simulation period*, T_s:** a constant such that: $T_s = \frac{1}{f_s}$.

class `sim2net._time.`**`Time`**
Bases: `object`

This class provides time abstractions for simulations.

Class `Time` keeps track of simulation steps and time in accordance with a given simulation frequency value.

The class must be set up by calling the `setup()` method.

`setup`(*simulation_frequency=1*)
Initializes time abstractions for simulations.

***Parameters*:**

- **simulation_frequency** (*int*): a value of the simulation frequency (greater than 0).

***Raises*:**

- **ValueError**: raised when a given value of the simulation frequency is less or equal to 0.

Examples:

```
>>> clock = Time()
>>> clock.setup()
>>> clock.tick()
(0, 0.0)
>>> clock.tick()
(1, 1.0)
```

```
>>> clock.tick()
(2, 2.0)
>>> clock.simulation_period
1.0

>>> clock = Time()
>>> clock.setup(4)
>>> clock.tick()
(0, 0.0)
>>> clock.tick()
(1, 0.25)
>>> clock.tick()
(2, 0.5)
>>> clock.tick()
(3, 0.75)
>>> clock.tick()
(4, 1.0)
>>> clock.simulation_period
0.25
```

simulation_frequency
(*Property*) The simulation frequency of type *int*.

simulation_period
(*Property*) The simulation period of type *float*.

simulation_step
(*Property*) The current simulation step value of type *int*.

simulation_time
(*Property*) The current simulation time value of type *float*.

tick()
Advances the simulation step and time values.

***Returns*:** A tuple of two values: the current simulation step (*int*) and the current simulation time (*float*).

Note: The first call to this method will always returns `(0, 0.0)`.

Module sim2net._channel

Provides an implementation of bidirectional communication channels for nodes in the simulated network.

The channels transmit *packets* that transport application *messages* between *neighboring* nodes. Each packet has its own identifier that is unique under the same sender, and can be received only by these nodes that are neighbors of the sender for the duration of the packet transmission according to the wireless signal propagation model used (see: sim2net.propagation). Potential packet losses are determined on the basis of the given model (see: sim2net.packet_loss), and transmission time of each packet is uniformly randomized in range $(0, t_{max}]$, where t_{max} is the given maximum transmission time in the *simulation time* units (see: sim2net._time).

class sim2net._channel.**Channel**(*time*, *packet_loss*, *node_id*, *maximum_transmission_time*)
Bases: sim2net._channel._Output, sim2net._channel._Input

This class implements bidirectional communication channels for each node in the simulated network.

The class has no members and inherits all its methods from two classes: _Input and _Output.

Application message passing is implemented here as follows. First, a message is sent locally by the _Output.send_message() method. Then, it is transmitted in a packet to neighboring nodes by the

_Output.transmit_packets() method. If the transmission is successful, the packet leaves the output channel by calling the _Output.deliver_packet() method and will be transferred to receiving nodes by calling the _Input.capture_packet() methods. Finally, the message can be received by the application by calling the _Input.receive_message() method.

***Parameters*:**

- **time**: a simulation time object of the sim2net._time.Time class;
- **packet_loss**: an object representing the packet loss model (see sim2net.packet_loss);
- **node_id** (*int*): an identifier of the node;
- **maximum_transmission_time** (*float*): maximum message transmission time between neighboring nodes in the *simulation time* units (see: sim2net._time).

***Raises*:**

- **ValueError**: raised when the given value of the *time* or *packet_loss* parameter is *None*; or when the given value of the *node_id* or *maximum_transmission_time* parameter is less than zero.

class sim2net._channel.**_Input**(*node_id*)
Bases: object

This class implements input channels for nodes in the simulated network.

Parameters: - **node_id** (*int*): an identifier of the node for which the

input channel is created.

capture_packet(*packet*)
Captures packets transmitted by neighboring nodes.

***Parameters*:**

- **packet** (*tuple*): a packet to capture represented by a tuple that contains the packet's identifier and transported application message, which is also a tuple containing an identifier of the sender and the message.

receive_message()
Returns a received application message.

***Returns*:** *None* value if there is no message at the current simulation step, or a *tuple* that contains an identifier of the sender and the received application message.

class sim2net._channel.**_Output**(*time*, *packet_loss*, *node_id*, *maximum_transmission_time*)
Bases: object

This class implements output channels for nodes in the simulated network.

Note: Methods transmit_packets() and deliver_packet() are responsible for the transmission and delivery of packages, so it is presumed that these methods are called at each step of the simulation.

Parameters: - **time**: a simulation time object of the

sim2net._time.Time class;

•**packet_loss**: an object representing a packet loss model to use (see sim2net.packet_loss);

•**node_id** (*int*): an identifier of the node for which the output channel is created;

•**maximum_transmission_time** (*float*): maximum message transmission time between neighboring nodes in the *simulation time* units (see: sim2net._time).

_Output__get_transmission_neighbors(*packet_id*, *transmission_time*, *neighbors*)
Returns a list of neighboring nodes at the beginning of packet transmission.

***Parameters*:**

- **packet_id** (*int*): an identifier of the transmitted packet;
- **transmission_time** (*float*): scheduled start time of the transmission;
- **neighbors** (*list*): a list of identifiers of all neighboring nodes of the sender at the current simulation step.

***Returns*:** (*list*) a list of identifiers of neighboring nodes of the sender for the given packet transmission or *None* value if the transmission time has not yet begun.

deliver_packet()
Delivers packets to neighboring nodes.

***Returns*:** *None* value if there is no packet to deliver at the current simulation step, or a *tuple* that contains the packet to deliver. In such a case, the tuple has the following data:

- an identifier of the packet to deliver of type *int*;
- a *tuple* that contains an identifier of the sender of type *int* and the transported application message;
- a list of identifiers of nodes which receive the packet.

Hint:

•It is possible that at one simulation step there will be multiple packets to deliver, so this method should be called as long until it returns *None* value.

•This method requires the use of complementary method `_Input.capture_packet()` of input channels of all nodes receiving the packet.

---

send_message(*message*, *neighbors*)
Sends an application message.

***Parameters*:**

- **message**: the application message to send of any type;
- **neighbors** (*list*): a list of identifiers of all neighboring nodes of the sender at the current simulation step.

transmit_packets(*neighbors*)
Transmits packets to neighboring nodes.

***Parameters*:**

- **neighbors** (*list*): a list of identifiers of all neighboring nodes of the sender at the current simulation step according to the wireless signal propagation model used (see: `sim2net.propagation`).

Module `sim2net._network`

This module provides an implementation of the mobile ad hoc network that is to be simulated.

The network is composed of the given number of nodes running the provided simulation application. The main method of this module, the `sim2net._network.Network.step()` method, is called at each simulation step and it advances the simulation by computing node failures, new positions of the nodes, performing direct communication between neighboring nodes, and executing the simulation application at each node.

Additionally, the sim2net._network._Communication class is implemented, which serves as a communication interface for the simulated nodes.

class sim2net._network.**Network**(*environment*)
Bases: object

This class implements the mobile ad hoc network that is to be simulated.

Parameters: - **environment**: a dictionary that contains objects, which form

the network environment for simulations (see sim2net._network.Network.__ENVIRONMENT for the objects list).

_Network__application()
Executes the simulation application at each operative node at the current simulation step.

See also:

sim2net.application

_Network__communication()
Performs packets propagation in the network at the current simulation step.

See also:

sim2net._channel.Channel

_Network__failure()
Computes node failures at the current simulation step.

See also:

sim2net.failure

_Network__move()
Calculates new positions of the simulated nodes at the current simulation step.

See also:

sim2net.mobility

_Network__neighborhood()
Calculates neighboring nodes at the current simulation step.

See also:

sim2net.propagation

communication_receive(*node_id*)
Receives an application message for the given node.

***Parameters*:**

- **node_id** (*int*): an identifier of the receiver.

***Returns*:** None value if there is no message at the current simulation step for the receiver, or a tuple that contains an identifier of the sender and the received application message.

See also:

sim2net._network._Communication

communication_send(*node_id*, *message*)
Sends an application message.

***Parameters*:**

- **node_id** (*int*): an identifier of the sender;

- **message**: the application message to send of any type.

Warning: This method uses the `copy.deepcopy()` function, and hence may be slow.

See also:

`sim2net._network._Communication`

finalize()
Calls the `sim2net.application.Application.finalize()` finalization method at each node after all simulation steps.

step()
Advances the simulation by one simulation step. This method is called as many times as there is simulation steps by the `sim2net.simulator.Sim2Net.run()` method.

This method calls: `sim2net._network.Network._Network__failure()`, `sim2net._network.Network._Network__move()`, `sim2net._network.Network._Network__neighb` `sim2net._network.Network._Network__communication()`, `sim2net._network.Network._Network__application()`, and `sim2net._time.Time.tick()` methods.

class sim2net._network.**_Communication**(*node_id*, *send_message*, *receive_message*)
Bases: object

This class implements a communication interface for the simulated nodes providing two methods for sending and receiving application messages.

Parameters: - **node_id** (*int*): an identifier of the node; - **send_message**: a sending method in the

`sim2net._network.Network` class;

•**receive_message**: a receiving method in the `sim2net._network.Network` class.

receive()
Returns None value if there is no message at the current simulation step, or a tuple that contains an identifier of the sender and the received application message.

send(*message*)
Sends an application message.

***Parameters*:**

- **message**: the application message to send of any type.

Module sim2net.simulator

This module provides an interface to the simulator for the `sim2net.cli` command-line tool and its main entry point for conducting simulations.

class sim2net.simulator.**Sim2Net**(*configuration*, *application_file*)
Bases: object

This class is the main entry point for conducting simulations.

Based on the given simulation configuration and application file, the class initializes and runs the simulation.

_Sim2Net__get_application_class(*application_file*)

_Sim2Net__get_arguments(*name*, *configuration*)

_Sim2Net__get_element(*name*, *configuration*, *environment*, *number=None*)

_Sim2Net__get_value(*name*, *configuration*)

_Sim2Net__report_error(*element*, *name*)

run()

3.3.2 Package sim2net.area

This package provides a collection of simulation area classes.

Area expresses a simulation surface by its shape and extent in the two-dimensional space with the origin in (0, 0).

Package modules:

- Module sim2net.area._area
- Module sim2net.area.rectangle
- Module sim2net.area.square

See also:

sim2net.placement

Module sim2net.area._area

Contains an abstract class that should be implemented by all simulation area classes.

class sim2net.area._area.**Area**(*name*)

Bases: object

This class is an abstract class that should be implemented by all simulation area classes.

Parameters: - **name** (*str*): a name of the implemented simulation area.

ORIGIN = (0.0, 0.0)

The origin for simulation areas.

get_area()

Creates a dictionary that stores information about the simulation area.

***Returns*:** A dictionary containing the simulation area information.

***Raises*:**

- **NotImplementedError**: this method is an abstract method.

height

(*Property*) A height of the simulation area of type *float*.

***Raises*:**

- **NotImplementedError**: this property is an abstract property.

logger

(*Property*) A logger object of the logging.Logger class with an appropriate channel name.

See also:

sim2net.utility.logger

width
(*Property*) A width of the simulation area of type *float*.

***Raises*:**

- **NotImplementedError**: this property is an abstract property.

within(*horizontal_coordinate*, *vertical_coordinate*)
Tests whether the given coordinates are within the simulation area.

***Parameters*:**

- **horizontal_coordinate** (*float*): a horizontal (x-axis) coordinate;
- **vertical_coordinate** (*float*): a vertical (y-axis) coordinate.

***Returns*:** (*bool*) *True* if the given coordinates are within the simulation area, or *False* otherwise.

***Raises*:**

- **NotImplementedError**: this method is an abstract method.

Module sim2net.area.rectangle

Provides an implementation of a rectangular simulation area in the two-dimensional space.

class sim2net.area.rectangle.**Rectangle**(*width*, *height*)
Bases: sim2net.area._area.Area

This class implements a rectangular simulation area of the given size in the two-dimensional space with the origin in (0, 0).

***Parameters*:**

- **width** (*float*): a width of the rectangular simulation area (along the horizontal x-axis),
- **height** (*float*): a height of the rectangular simulation area (along the vertical y-axis).

***Raises*:**

- **ValueError**: raised when a given value of either **width** or **height** parameter is equal to or less than 0.

get_area()
Creates a dictionary that stores information about the simulation area.

***Returns*:** A dictionary that stores information about the simulation area; it has the following fields:

- 'area name': a name of the simulation area of type *str*,
- 'width': a width of the simulation area of type *float*,
- 'height': a height of the simulation area of type *float*.

height
(*Property*) A height of the simulation area of type *float*.

width
(*Property*) A width of the simulation area of type *float*.

within(*horizontal_coordinate*, *vertical_coordinate*)
Tests whether the given coordinates are within the simulation area.

***Parameters*:**

- **horizontal_coordinate** (*float*): a horizontal (x-axis) coordinate;
- **vertical_coordinate** (*float*): a vertical (y-axis) coordinate.

***Returns*:** (*bool*) *True* if the given coordinates are within the rectangular simulation area, or *False* otherwise.

Module `sim2net.area.square`

Provides an implementation of a square simulation area in the two-dimensional space.

class `sim2net.area.square.`**`Square`**(*side*)

Bases: `sim2net.area.rectangle.Rectangle`

This class implements a square simulation area of the given size in the two-dimensional space with the origin in (0, 0).

***Parameters*:**

- **side** (*float*): a side length of the square simulation area.

Note: In this case, the `sim2net.area.rectangle.Rectangle()` method is called with the **width** and **height** parameters set to the value of the given **side** argument.

`get_area`()

Creates a dictionary that stores information about the simulation area.

***Returns*:** A dictionary that stores information about the simulation area; it has the following fields:

- 'area name': a name of the simulation area of type *str*,
- 'side': a side length of the square simulation area of type *float*.

3.3.3 Package `sim2net.failure`

This package provides a collection of process failure models.

A *process failure* occurs whenever the process does not behave according to its algorithm, and here the term *process* means the *application* running on one of the nodes in the simulated network. To simulate such behaviors, process failure models are used, and they differ in the nature and scope of faults. Possible process failures may include ([CGR11]): **crashes** (where a process at some time may simply stop to execute any steps and never recovers); **omissions** (where a process does not send or receive messages that it is supposed to send or receive according to its algorithm); **crashes with recoveries** (where a process *crashes* and never recovers or it keeps infinitely often crashing and recovering); **eavesdropping** (where a process leaks information obtained in its algorithm to an outside entity); and **arbitrary** (where a process may deviate in any conceivable way from its algorithm).

Package modules:

- Module `sim2net.failure._failure`
- Module `sim2net.failure.crash`

Module `sim2net.failure._failure`

Contains an abstract class that should be implemented by all process failure model classes.

class `sim2net.failure._failure.`**`Failure`**(*name*)

Bases: `object`

This class is an abstract class that should be implemented by all process failure model classes.

Parameters: - **name** (*str*): a name of the implemented process failure

model.

logger
(*Property*) A logger object of the `logging.Logger` class with an appropriate channel name.

See also:

`sim2net.utility.logger`

node_failure (*failures*)
Gives *in place* information about nodes which processes have failed according to the implemented process failure model.

***Parameters*:**

- **failures** (*list*): a list of boolean values of the size equal to the total number of nodes in the simulated network; *True* value in position i indicates that the process on node number i has failed.

random_generator
(*Property*) An object representing the `sim2net.utility.randomness._Randomness` pseudo-random number generator.

Module `sim2net.failure.crash`

This module provides an implementation of the crash model.

In the crash model ([CGR11]), processes at some time may simply stop to execute any steps, and if this is the case, the faulty processes never recover. In this implementation, a failure for each process is determined randomly with the use of the given *crash probability* that indicates the probability that a process will crash during the total simulation time. By the method used, times at which processes crash will be distributed uniformly in the total simulation time. There is also a possibility to setup a *transient period* (at the beginning of the simulation), during which process failures do not occur, and the total number of faulty processes can also be limited to a given value.

class `sim2net.failure.crash.`**Crash**(*time*, *nodes_number*, *crash_probability*, *maximum_crash_number*, *total_simulation_steps*, *transient_steps=0*)
Bases: `sim2net.failure._failure.Failure`

This class implements the process crash model.

Note: It is presumed that the `node_failure()` method is called at each step of the simulation.

***Parameters*:**

- **time**: a simulation time object of the `sim2net._time.Time` class;
- **nodes_number** (*int*): the total number of nodes in the simulated network;
- **crash_probability** (*float*): the probability that a single process will crash during the total simulation time;
- **maximum_crash_number** (*int*): the maximum number of faulty processes;
- **total_simulation_steps** (*int*): the total number of simulation steps;
- **transient_steps** (*int*): a number of steps at the beginning of the simulation during which no crashes occur (default: *0*).

***Raises*:**

- **ValueError**: raised when the given value of the *time* object is *None*; or when the given number of nodes is less than or equal to zero; or when the given crash probability is less than zero or grater than one; or when the given value of the maximum number of faulty processes or the given value of the total simulation steps is less than zero; or when the number of steps in the transient period is less than zero or greater than the given value of the total simulation steps.

_Crash__crashes(*nodes_number*, *crash_probability*, *maximum_crash_number*, *total_simulation_steps*, *transient_steps*)

Determines faulty processes and their times of crash with the use of the given *crash probability*. There is also a possibility to setup a transient period (at the beginning of the simulation), during which process failures do not occur, and the total number of faulty processes can also be limited to a given value.

***Parameters*:**

- **nodes_number** (*int*): the total number of nodes in the simulated network;
- **crash_probability** (*float*): the probability that a single process will crash during the total simulation time;
- **maximum_crash_number** (*int*): the maximum number of faulty processes;
- **total_simulation_steps** (*int*): the total number of simulation steps;
- **transient_steps** (*int*): a number of steps at the beginning of the simulation during which no crashes occur (default: *0*).

***Returns*:** A *list* of *tuples*; each tuple contains an identifier of the node with faulty process and its time of crash (in simulation steps). The list is sorted in ascending order by crash times.

node_failure(*failures*)

Gives *in place* information about nodes which processes have failed according to the crash model.

***Parameters*:**

- **failures** (*list*): a list of boolean values of the size equal to the total number of nodes in the simulated network; *True* value in position i indicates that the process on node number i has failed.

***Returns*“** A *list* of nodes which processes failed at the current simulation step.

Examples:

In order to avoid any process failures use this class with the *crash_probability* and/or *maximum_crash_number* parameters set to *0*, as in the examples below.

```
>>> clock = Time()
>>> clock.setup()
>>> crash = Crash(clock, 4, 0.0, 0, 2)
>>> failures = [False, False, False, False]
>>> clock.tick()
(0, 0.0)
>>> crash.node_failure(failures)
[]
>>> print failures
[False, False, False, False]
>>> clock.tick()
(1, 1.0)
>>> crash.node_failure(failures)
[]
>>> print failures
[False, False, False, False]
```

```
>>> clock = Time()
>>> clock.setup()
>>> crash = Crash(clock, 4, 1.0, 0, 2)
>>> failures = [False, False, False, False]
>>> clock.tick()
(0, 0.0)
>>> crash.node_failure(failures)
[]
>>> print failures
[False, False, False, False]
>>> clock.tick()
(1, 1.0)
>>> crash.node_failure(failures)
[]
>>> print failures
[False, False, False, False]
```

3.3.4 Package sim2net.mobility

This package provides a collection of mobility model classes.

Mobility models ([LNR04], [CBD02]) are designed to describe the movement pattern of mobile nodes, and how their location, velocity and acceleration change over time. Since mobility patterns may play a significant role in determining the protocol performance, it is desirable for mobility models to emulate the movement pattern of targeted real life applications in a reasonable way.

The literature categorises mobility models as being either *entity* or *group models*. Entity models are used as a tool to model the behaviour of individual mobile nodes, treated as autonomous, independent entities. On the other hand, the key assumption behind the group models is that individual nodes influence each other's movement to some degree. Therefore, group models have become helpful in simulating the motion patterns of a group as a whole.

Package modules:

- Module sim2net.mobility._mobility
- Module sim2net.mobility.gauss_markov
- Module sim2net.mobility.nomadic_community
- Module sim2net.mobility.random_direction
- Module sim2net.mobility.random_waypont

Module sim2net.mobility._mobility

Contains an abstract class that should be implemented by all mobility model classes.

class sim2net.mobility._mobility.**Mobility**(*name*)

Bases: object

This class is an abstract class that should be implemented by all mobility model classes.

Parameters: - **name** (*str*): a name of the implemented mobility model.

get_current_position(*node_id*, *node_speed*, *node_coordinates*)

Calculates and returns a node's position at the current simulation step in accordance with the implemented mobility model. It is assumed that this method is called at each step of the simulation.

***Parameters*:**

- **node_id** (*int*): an identifier of the node;
- **node_speed**: an object representing the node's speed;
- **node_coordinates** (*list*): values of the node's horizontal and vertical coordinates at the previous simulation step.

***Returns*:** A tuple containing current values of the node's horizontal and vertical coordinates.

***Raises*:**

- **NotImplementedError**: this method is an abstract method.

logger

(*Property*) A logger object of the `logging.Logger` class with an appropriate channel name.

See also:

`sim2net.utility.logger`

random_generator

(*Property*) An object representing the `sim2net.utility.randomness._Randomness` pseudo-random number generator.

Module sim2net.mobility.gauss_markov

This module provides an implementation of the Gauss-Markov mobility model.

In the Gauss-Markov mobility model ([LH99]), motion of a single node is modelled in the form of a Gauss-Markov stochastic process. At the beginning, each node is assigned with an initial speed and direction, as well as mean values of these parameters. Then, at set intervals of time (e.g. *simulation steps*), a new speed and direction are calculated for each node, which follow the new course until the next time step. This is repeated through the duration of the simulation. The new speed (v) and direction (d), at time interval n, are evaluated in the following manner:

- $v_n = \alpha \times v_{n-1} + (1 - \alpha) \times \overline{v} + \sqrt{(1 - \alpha^2)} \times v_x$,
- $d_n = \alpha \times d_{n-1} + (1 - \alpha) \times \overline{d} + \sqrt{(1 - \alpha^2)} \times d_x$;

where:

- $0\alpha1$ is a tuning parameter used to vary the randomness;
- $\overline{v}$ is constant representing the mean value of speed;
- $\overline{d}$ is constant representing the mean value of direction;
- v_x and d_x are random variables from a normal (Gaussian) distribution.

Consequently, at time interval n, node's horizontal (x) and vertical (y) coordinates in the simulation area are given by the following equations:

- $x_n = x_{n-1} + v_{n-1} \times \cos d_{n-1}$;
- $y_n = y_{n-1} + v_{n-1} \times \sin d_{n-1}$.

It is worth to note that when α is equal to 1, movement becomes predictable, losing all randomness. On the other hand, if α is equal to 0, the model becomes memoryless: the new speed and direction are based completely upon the mean speed and direction constants ($\overline{v}$ and $\overline{d}$) and the Gaussian random variables (v_x and d_x).

class `sim2net.mobility.gauss_markov.`**GaussMarkov**(*area*, *time*, *initial_coordinates*, *initial_speed*, ***kwargs*)

Bases: `sim2net.mobility._mobility.Mobility`

This class implements the Gauss-Markov mobility model, in which motion of each node is modelled in the form of a Gauss-Markov stochastic process.

Note:

- Due to the characteristics of this model, it is expected that each node has assigned the normal speed distribution (see: `sim2net.speed.normal`) – the speed is used as random variable v_x when a new speed is calculated.
- All direction values used in this implementation are expressed in radians.
- The `get_current_position()` method computes a position of a node at the current *simulation step* (see: `sim2net._time`), so it is presumed that the method is called at each step of the simulation.

***Parameters*:**

- **area**: an object representing the simulation area;
- **time**: a simulation time object of the `sim2net._time.Time` class;
- **initial_coordinates** (*list*): initial coordinates of all nodes; each element of this parameter should be a tuple of two coordinates: horizontal and vertical (respectively) of type *float*;
- **initial_speed** (*float*): a value of the initial speed that is assigned to each node at the beginning of the simulation;
- **kwargs** (*dict*): a dictionary of (optional) keyword parameters related to the Gauss-Markov mobility model; the following parameters are accepted:

  alpha (*float*) The tuning parameter $0\alpha1$ used to vary the randomness of movements (default: *0.75*).

  direction_deviation (*float*) Constant representing the standard deviation of direction random variable d_x (it defaults to $\frac{\pi}{2}$).

  direction_margin (*float*) Constant used to change direction mean $\overline{d}$ to ensure that nodes do not remain near a border of the simulation area for a long period of time (it defaults to *0.15*, or *15%* of the simulation area width/height, and cannot be less than zero and greater than one; see: `_GaussMarkov__velocity_recalculation()`).

  direction_mean (*float*) Constant representing mean value $\overline{d}$ of direction (it defaults to $\frac{\pi}{6}$). The same value is used as mean of direction random variable d_x.

  recalculation_interval (*int*) Velocity (i.e. speed and direction) recalculation time interval (it defaults to the *simulation frequency*; see: `sim2net._time`). It determines how often, counting in simulation steps, new values of velocity are recalculated.

***Raises*:**

- **ValueError**: raised when the given value of the *area*, *time*, *initial_coordinates* or *initial_speed* parameter is *None*; or when the given value of the keyword parameter *alpha* is less than zero or greater that one; or when the given value of the (optional) keyword parameter *direction_margin* is less than zero or greater than one.

Example:

```
>>> gm = GaussMarkov(area, time, coordinates, 10.0, alpha=0.35)
```

_GaussMarkov__get_new_direction()
Randomizes a new direction with the normal (Gaussian) distribution.

***Returns*:** (*float*) a newly randomized direction value.

_GaussMarkov__step_move(*node_id*, *node_coordinates*)
Computes a node's position at the current simulation step.

***Parameters*:**

- **node_id** (*int*): an identifier of the node;
- **node_coordinates** (*list*): values of the node's horizontal and vertical coordinates at the previous simulation step.

***Returns*:** (*tuple*) current values of the node's horizontal and vertical coordinates.

_GaussMarkov__velocity_recalculation(*node_id*, *node_speed*, *node_coordinates*)

Recalculates a node's velocity, i.e. its speed and direction, as a Gauss-Markov stochastic process.

To ensure that a node does not remain near a border of the simulation area for a long period of time, the node is forced away from the border when it moves within certain distance of the edge. This is done by modifying mean direction $\overline{d}$. For example, when a node is near the right border of the simulation area, the value of $\overline{d}$ changes to 180 degrees (π). The distance that is used in this method is calculated as a product of the **direction margin** and area width or height.

***Parameters*:**

- **node_id** (*int*): an identifier of the node;
- **node_speed**: an object representing the node's speed;
- **node_coordinates** (*list*): values of the node's horizontal and vertical coordinates at the previous simulation step.

get_current_position(*node_id*, *node_speed*, *node_coordinates*)

Calculates and returns a node's position at the current simulation step in accordance with the Gauss-Markov mobility model.

A distance of the route traveled by the node, between the current and previous simulation steps, is calculated as the product of the current node's speed and the *simulation period* (see: `sim2net._time` module). Therefore, it is assumed that this method is called at every simulation step.

***Parameters*:**

- **node_id** (*int*): an identifier of the node;
- **node_speed**: an object representing the node's speed;
- **node_coordinates** (*list*): values of the node's horizontal and vertical coordinates at the previous simulation step.

***Returns*:** A tuple containing current values of the node's horizontal and vertical coordinates.

Module `sim2net.mobility.nomadic_community`

This module provides an implementation of the Nomadic Community mobility model.

The Nomadic Community ([CBD02]) is a group mobility model, in which a group of nodes collectively moves from one destination to another. Destinations for the group are determined by the so-called *reference point* that is selected at random within the simulation area. Moreover, each node uses an entity mobility model to roam, within a fixed range, around the current reference point. But when the reference point changes, all nodes travel to the new area defined by new coordinates of the reference point (and its range of free roam) and then begin roaming around it. The whole process is repeated again and again until simulation ends.

class `sim2net.mobility.nomadic_community.`**NomadicCommunity**(*area*, *time*, *initial_coordinates*, *pause_time=0.0*, *area_factor=0.25*)

Bases: `sim2net.mobility.random_waypoint.RandomWaypoint`,

sim2net.mobility._mobility.Mobility

This class implements the Nomadic Community mobility model, in which a group of nodes travels together from one location to another.

In this implementation, coordinates of the reference point are uniformly selected at random within the simulation area once every $x + y \times pause\_time$ *simulation time* units (see: sim2net._time module), where x is uniformly picked at random from the range $[100, 200]$, and y from the range $[1, 10]$. Nodes roam around reference points in accordance with the **Random Waypoint** mobility model (see: sim2net.mobility.random_waypoint module). The width and height of the (square or rectangular) free roam area around the reference point are computed as a product of the *area_factor* parameter and the width and height (respectively) of the simulation area.

Note: The get_current_position() method computes a position of a node at the current *simulation step* (see: sim2net._time), so it is presumed that the method is called at each step of the simulation.

***Parameters*:**

- **area**: an object representing the simulation area;
- **time**: a simulation time object of the sim2net._time.Time class;
- **initial_coordinates** (*list*): initial coordinates of all nodes; each element of this parameter should be a tuple of two coordinates: horizontal and vertical (respectively) of type *float*;
- **pause_time** (*float*): a maximum value of the pause time in the *simulation time* units (default: *0.0*, see also: sim2net._time);
- **area_factor** (*float*): a factor used to determine the width and height of the free roam area around the reference point (default: *0.25*).

***Raises*:**

- **ValueError**: raised when the given value of the *area*, *time* or *initial_coordinates* parameter is *None*; or when the given value of the *pause_time* parameter is less that zero; or when the given value of the *area_factor* parameter is less than zero or greater than one.

(At the beginning, nodes' destination points are set to be equal to its initial coordinates passed by the *initial_coordinates* parameter.)

_NomadicCommunity__get_free_roam_area_edges(*reference_point*)
Computes boundaries of a free roam area around a given reference point.

***Parameter*:**

- **reference_point** (*tuple*) containing horizontal and vertical coordinates (respectively) of the reference point.

***Returns*:** A *tuple* containing values of the top, right, bottom and left boundaries (respectively) in the simulation area.

_NomadicCommunity__get_new_reference_point()
Uniformly randomizes new coordinates of the reference point. The vertical and horizontal coordinates are returned (respectively) as a *tuple*.

_NomadicCommunity__get_new_relocation_time()
Randomizes and returns a new relocation time of type *float*, after which coordinates of the reference point will be changed.

_NomadicCommunity__reference_point_relocation()
Relocates the reference point by picking its new coordinates. The relocation takes place only if all nodes

are within the current area of free roam and the relocation time has expired. Otherwise, the current coordinates of the reference point are preserved.

_get_new_destination()
Uniformly randomizes a new waypoint within the range of free roam and returns its coordinates as a *tuple*.

get_current_position(*node_id*, *node_speed*, *node_coordinates*)
Calculates and returns a node's position at the current simulation step in accordance with the Nomadic Community mobility model (and Random Waypoint model within the area of free roam).

A distance of the route traveled by the node, between the current and previous simulation steps, is calculated as the product of the current node's speed and the *simulation period* (see: sim2net._time module). Therefore, it is assumed that this method is called at every simulation step.

***Parameters*:**

- **node_id** (*int*): an identifier of the node;
- **node_speed**: an object representing the node's speed;
- **node_coordinates** (*list*): values of the node's horizontal and vertical coordinates at the previous simulation step.

***Returns*:** A tuple containing current values of the node's horizontal and vertical coordinates.

Module sim2net.mobility.random_direction

This module provides an implementation of the Random Direction mobility model.

At the beginning of the simulation, with the use of the Random Direction mobility model ([RMM01]), a node first stops for some random *pause time*, and then randomly selects a *direction* in which to move. The direction is measured in degrees, and at first, the node selects a degree between 0 and 359. Next, it finds a destination point on the boundary of the simulation area in this direction of travel and moves with a constant, but randomly selected (between the *minimum* and *maximum* values), speed to its destination. Once it reaches the destination, it pauses, and then selects a new direction between 0 and 180 degree (the degree is limited because the node is already on the boundary of the simulation area). The node then identifies the destination on the boundary in this line of direction, selects a new speed, and resumes travel. The whole process is repeated again and again until simulation ends. The speed and destination of each node are chosen independently of other nodes.

class sim2net.mobility.random_direction.**RandomDirection**(*area*, *time*, *initial_coordinates*, *pause_time=0.0*)
Bases: sim2net.mobility.random_waypoint.RandomWaypoint, sim2net.mobility._mobility.Mobility

This class implements the Random Direction mobility model, in which each node moves along straight lines from one destination point, on the boundary of the simulation area, to another.

The nodes may also have *pause times* when they reach their destination points, and their speeds are selected at random between the *minimum* and *maximum speed* values. (All random picks are uniformly distributed).

Note: The get_current_position() method computes a position of a node at the current *simulation step* (see: sim2net._time), so it is presumed that the method is called at each step of the simulation.

See also:

sim2net.mobility.random_waypoint

***Parameters*:**

- **area**: an object representing the simulation area;
- **time**: a simulation time object of the `sim2net._time.Time` class;
- **initial_coordinates** (*list*): initial coordinates of all nodes; each element of this parameter should be a tuple of two coordinates: horizontal and vertical (respectively) of type *float*;
- **pause_time** (*float*): a maximum value of the pause time in the *simulation time* units (default: *0.0*, see also: `sim2net._time`).

***Raises*:**

- **ValueError**: raised when the given value of the *area*, *time* or *initial_coordinates* parameter is *None* or when the given value of the *pause_time* parameter is less that zero.

(At the beginning, nodes' destination points are set to be equal to its initial coordinates passed by the *initial_coordinates* parameter.)

_get_new_destination()
Randomizes a new destination point on the boundary of the simulation area and returns its coordinates as a *tuple*.

Module `sim2net.mobility.random_waypont`

This module provides an implementation of the Random Waypoint mobility model.

In this model ([JM96], [BMJ+98]), a node first stops for some random *pause time*. Then, the node randomly picks a point within the simulation area and starts moving toward it with a constant, but randomly selected, speed that is uniformly distributed between the *minimum* and *maximum speed* values. Upon reaching the destination point (or waypoint), the node pauses again and then moves toward a newly randomized point. (If the *pause time* is equal to zero, this leads to continuous mobility.) The whole process is repeated again and again until simulation ends. The speed and destination of each node are chosen independently of other nodes.

class sim2net.mobility.random_waypoint.**RandomWaypoint**(*area*, *time*, *initial_coordinates*, *pause_time=0.0*)
Bases: `sim2net.mobility._mobility.Mobility`

This class implements the Random Waypoint mobility model, in which each node moves along straight lines from one waypoint to another.

The waypoints are randomly picked within the simulation area. The nodes may also have *pause times* when they reach waypoints, and their speeds are selected at random between the *minimum* and *maximum speed* values. (All random picks are uniformly distributed).

Note: The `get_current_position()` method computes a position of a node at the current *simulation step* (see: `sim2net._time`), so it is presumed that the method is called at each step of the simulation.

***Parameters*:**

- **area**: an object representing the simulation area;
- **time**: a simulation time object of the `sim2net._time.Time` class;
- **initial_coordinates** (*list*): initial coordinates of all nodes; each element of this parameter should be a tuple of two coordinates: horizontal and vertical (respectively) of type *float*;
- **pause_time** (*float*): a maximum value of the pause time in the *simulation time* units (default: *0.0*, see also: `sim2net._time`).

***Raises*:**

- **ValueError**: raised when the given value of the *area*, *time* or *initial_coordinates* parameter is *None* or when the given value of the *pause_time* parameter is less that zero.

(At the beginning, nodes' destination points are set to be equal to its initial coordinates passed by the *initial_coordinates* parameter.)

_assign_new_destination(*node_id*, *node_speed*)
Assigns a new destination point for a node of a given ID and picks its new speed value. (See also: `_get_new_destination()`)

***Parameters*:**

- **node_id** (*int*): an identifier of the node;
- **node_speed**: an object representing the node's speed.

_assign_new_pause_time(*node_id*)
Assigns a new pause time for a node of a given ID and returns the value. If the maximum pause time is set to *0*, *None* value is assigned and returned.

***Parameters*:**

- **node_id** (*int*): an identifier of the node.

***Returns*:** (*float*) a newly randomized pause time.

_diagonal_trajectory(*node_id*, *node_coordinates*, *step_distance*)
Computes the current position of a node if its trajectory is not parallel to the horizontal or vertical axis of the simulation area. (See also: `_parallel_trajectory()`.)

***Parameters*:**

- **node_id** (*int*): an identifier of the node;
- **node_coordinates** (*list*): values of the node's horizontal and vertical coordinates at the previous simulation step.
- **step_distance** (*float*): a distance that the node has moved between the previous and current simulation step.

***Returns*:** (*tuple*) current values of the node's horizontal and vertical coordinates.

_get_new_destination()
Randomizes a new waypoint and returns its coordinates as a *tuple*.

_get_new_pause_time()
Randomizes a new pause time and returns its value of type *float*.

_parallel_trajectory(*coordinate*, *destination*, *step_distance*)
Computes the current position of a node when one of its coordinates is equal to the corresponding destination coordinate. In such a case, the node moves on a straight line that is parallel to the horizontal or vertical axis of the simulation area. (See also: `_diagonal_trajectory()`.)

***Parameters*:**

- **coordinate** (*float*): a value of the previous node's coordinate that is not equal to its corresponding destination coordinate;
- **destination** (*float*): a value of the destination coordinate;
- **step_distance** (*float*): a distance that the node has moved between the previous and current simulation steps.

***Returns*:** (*float*) a current value of the node's coordinate.

_pause (*node_id*, *node_coordinates*)
Decreases the current value of a node's pause time and returns the result of type *float*, or *None* if the pause time has expired.

***Parameters*:**

- **node_id** (*int*): an identifier of the node;
- **node_coordinates** (*list*): values of the node's horizontal and vertical coordinates at the previous simulation step.

_step_move (*node_id*, *node_speed*, *node_coordinates*)
Computes a node's position at the current simulation step. If its trajectory is parallel to the horizontal or vertical axis of the simulation area, the `_steady_trajectory()` method is used, otherwise the `_diagonal_trajectory()` method is used.

***Parameters*:**

- **node_id** (*int*): an identifier of the node;
- **node_speed**: an object representing the node's speed;
- **node_coordinates** (*list*): values of the node's horizontal and vertical coordinates at the previous simulation step.

***Returns*:** (*tuple*) current values of the node's horizontal and vertical coordinates.

get_current_position (*node_id*, *node_speed*, *node_coordinates*)
Calculates and returns a node's position at the current simulation step in accordance with the Random Waypoint mobility model.

A distance of the route traveled by the node, between the current and previous simulation steps, is calculated as the product of the current node's speed and the *simulation period* (see: `sim2net._time` module). Therefore, it is assumed that this method is called at every simulation step.

***Parameters*:**

- **node_id** (*int*): an identifier of the node;
- **node_speed**: an object representing the node's speed;
- **node_coordinates** (*list*): values of the node's horizontal and vertical coordinates at the previous simulation step.

***Returns*:** A tuple containing current values of the node's horizontal and vertical coordinates.

3.3.5 Package `sim2net.packet_loss`

This package provides a collection of packet loss model classes.

Packet loss occurs when a packet of data (or message) traveling across a computer network fails to reach its destination(s). In wireless communication, the loss may be caused by wireless channel properties (e.g. signal degradation due to multi-path *fading* or *shadowing*), packet collisions or faulty networking hardware. Thus, the purpose of packet loss models is to simulate (potential) transmission failures in wireless communication.

Package modules:

- Module `sim2net.packet_loss._packet_loss`
- Module `sim2net.packet_loss.gilbert_elliott`

See also:

`sim2net.propagation`

Module `sim2net.packet_loss._packet_loss`

Contains an abstract class that should be implemented by all packet loss model classes.

class `sim2net.packet_loss._packet_loss.`**`PacketLoss`**(*name*)

Bases: `object`

This class is an abstract class that should be implemented by all packet loss model classes.

Parameters: - **name** (*str*): a name of the implemented placement model.

`logger`

(*Property*) A logger object of the `logging.Logger` class with an appropriate channel name.

See also:

`sim2net.utility.logger`

`packet_loss`()

Returns information about whether a transmitted packet has been lost or can be successfully received by destination nodes according to the implemented packet loss model.

***Returns*:** (*bool*) *True* if the packet has been lost, or *False* otherwise.

***Raises*:**

- **NotImplementedError**: this method is an abstract method.

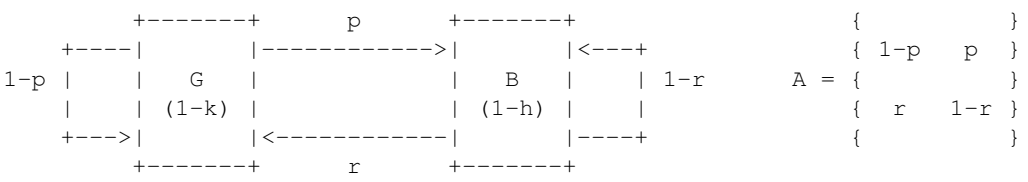
`random_generator`

(*Property*) An object representing the `sim2net.utility.randomness._Randomness` pseudo-random number generator.

Module `sim2net.packet_loss.gilbert_elliott`

This module provides an implementation of the Gilbert-Elliott packet loss model.

The Gilbert-Elliott model ([Gil60], [Ell63]) describes error patterns in communication channels ([HH08]). The model is based on a simple *Markov chain* with two states: `G` (for *good* or *gap*) and `B` (for *bad* or *burst*). Each of them may generate errors (packet losses) as independent events at a state dependent error rate: $1-k$ in the *good* state and $1-h$ in the *bad* state. The chain is shown in the figure below along with the transition matrix A that uses two transitions: $p = P(q_t = B|q_{t-1} = G)$ and $r = P(q_t = G|q_{t-1} = B)$ (q_t denotes the state at time t):



Then, error rate p_E is obtained (in steady mode) for the model as follows: $p_E = (1-k) \times \frac{r}{p+r} + (1-h) \times \frac{p}{p+r}$ (assuming: $0 < p, r < 1$).

It is worth to note that when $q = 1 - p$ (and $k = 1, h = 0$), this model reduces to the Bernoulli model – a very simple loss model, characterized by a single parameter, the loss rate r, used for modeling packet loss.

Finally, p equal to 0 means that no losses are possible, whereas r equal to 0 means that no transmission is successful (once the `B` state is reached).

class sim2net.packet_loss.gilbert_elliott.**GilbertElliott**(*prhk=None*)
Bases: sim2net.packet_loss._packet_loss.PacketLoss

This class implements the Gilbert-Elliott packet loss model.

***Parameters*:**

- **prhk** (*tuple*): a *tuple* that contains four model parameters: $0p, r, h, k1$, respectively (each of type *float*). The parameters default to the following values:
  - $p = 0.00001333$,
  - $r = 0.00601795$,
  - $h = 0.55494900$,
  - $k = 0.99999900$;

  (which leads to error rate equal to 0.098% and the mean packet loss rate equal to 0.1% ([HH08])).

***Raises*:**

- **ValueError**: raised when the given value any model parameter is less than zero or greater that one.

(At the beginning the model is in the G state.)

packet_loss()
Returns information about whether a transmitted packet has been lost or can be successfully received by destination node(s) according to the Gilbert-Elliott packet loss model.

***Returns*:** (*bool*) *True* if the packet has been lost, or *False* otherwise.

3.3.6 Package sim2net.placement

This package provides a collections of placement model classes.

A placement (or deployment) model describes a simulation area and a given number of nodes deployed in the area. It provides also node positions in case of static networks or initial node positions for mobile environments.

Package modules:

- Module sim2net.placement._placement
- Module sim2net.placement.grid
- Module sim2net.placement.normal
- Module placement.uniform

See also:

sim2net.area

Module sim2net.placement._placement

Contains an abstract class that should be implemented by all placement classes.

class sim2net.placement._placement.**Placement**(*name*)
Bases: object

This class is an abstract class that should be implemented by all placement model classes.

Parameters: - **name** (*str*): a name of the implemented placement model.

get_placement()
Generates placement positions and returns the result as a dictionary.

***Returns*:** A dictionary containing the placement information.

***Raises*:**

- **NotImplementedError**: this method is an abstract method.

logger
(*Property*) A logger object of the `logging.Logger` class with an appropriate channel name.

See also:

`sim2net.utility.logger`

static position_conflict(*horizontal_coordinates*, *vertical_coordinates*, *index=-1*)
If *index* is less than 0, checks whether the given coordinates are unique, that is, if no two points have the same horizontal and vertical coordinates. Otherwise, checks if there is a point that has the same coordinates as these at the *index* position.

***Parameters*:**

- **horizontal_coordinates** (*list*): a list of horizontal coordinates;
- **vertical_coordinates** (*list*): a list of vertical coordinates;
- **index** (*int*): an index of the coordinate lists; if greater than -1, it is checked whether there is a point with the same horizontal and vertical coordinates as at *index*.

***Returns*:** (*int*) an index of the coordinate that is in conflict, or -1 if the given coordinates are unique.

***Raises*:**

- **ValueError**: if given coordinate lists have different lengths, or if a given value of the *index* parameter is greater than the total number of coordinates.

Examples:

```
>>> Placement.position_conflict([1, 2, 2, 4], [5, 6, 6, 7])
2
>>> Placement.position_conflict([1, 2, 2, 4], [5, 6, 6, 7], 1)
2
>>> Placement.position_conflict([1, 2, 2, 4], [5, 6, 6, 7], 0)
-1
```

random_generator
(*Property*) An object representing the `sim2net.utility.randomness._Randomness` pseudo-random number generator.

Module sim2net.placement.grid

Provides an implementation of the grid placement model.

In the grid placement model nodes are placed at intersections of a square or rectangular grid. Usually, the grid has quadratic-shaped cells with edge length that is close to the communication radius of a node. It creates networks that are regular in shape and provides excellent connectivity at a startup.

class sim2net.placement.grid.**Grid**(*area*, *nodes_number*, *transmission_range*)
Bases: `sim2net.placement._placement.Placement`

This class implements the grid placement model, in which a given number of nodes are placed at intersections of a square or rectangular grid within a simulator area.

***Parameters*:**

- **area**: an object representing the simulation area;
- **nodes_number** (*int*): a number of nodes to place within the simulation area;
- **transmission_range** (*float*): a value of the transmission (or communication) radius of nodes, that is, the distance from a transmitter at which the signal strength remains above the minimum usable level.

***Raises*:**

- **ValueError**: raised when: the given number of nodes or transmission range is less or equal to 0, or when the given value of the *area* parameter is *None*.

_Grid__adjust_grid_dimensions(*columns*, *rows*)

Adjusts the given grid dimensions to the size of the simulation area. If the area shape is square and the grid shape is rectangular, the longer side of the grid is placed along the horizontal x-axis of the simulation area. If both shapes are rectangular, the longer side of the grid is placed along the longer size of the simulation area.

***Parameters*:**

- **columns** (*int*): a number of grid columns;
- **rows** (*int*): a number of grid rows.

***Returns*:** A number of grid columns and rows as a tuple.

_Grid__get_grid_dimensions()

Calculates dimensions of the grid based on the number of nodes. If the number has a square root, the grid shape will be a square, otherwise it will be a rectangular. In the worst case if the number of nodes is prime, the number of rows (or columns) will be equal to one.

***Returns*:** A number of grid columns and rows as a tuple.

_Grid__get_horizontal_coordinates(*columns*, *rows*, *distance*)

Generates horizontal coordinates of nodes based on the number of columns, rows and the distance between nodes.

***Returns*:** A list of horizontal coordinates.

_Grid__get_nodes_distance(*columns*, *rows*)

Calculates a distance between nodes in the same row and column based on the their transmission ranges. The distance is also adjust to fit the dimensions of the simulation area.

***Returns*:** A distance between nodes in the grid of type *float*.

_Grid__get_vertical_coordinates(*columns*, *rows*, *distance*)

Generates vertical coordinates of nodes based on the number of columns, rows and the distance between nodes.

***Returns*:** A list of vertical coordinates.

get_placement()

Generates grid placement coordinates for the given number of nodes and its transmission ranges and returns the result as a dictionary.

***Returns*:** A list of tuples of horizontal and vertical coordinates for each host.

Module sim2net.placement.normal

Provides an implementation of the normal placement model.

In the normal placement model, a simulation area of a given size is chosen and a given number of nodes are placed over it with the normal, i.e. Gaussian, probability distribution.

class sim2net.placement.normal.**Normal**(*area*, *nodes_number*, *standard_deviation=0.2*)
Bases: sim2net.placement._placement.Placement

This class implements the normal placement model, in which a given number of nodes are placed over a simulation area with the normal probability distribution.

***Parameters*:**

- **area**: an object representing the simulation area;
- **nodes_number** (*int*): a number of nodes to place over the simulation area;
- **standard_deviation** (*float*): a value of the standard deviation (default: *0.2*).

***Raises*:**

- **ValueError**: raised when the number of nodes is less or equal to 0, or when the given value of the *area* parameter is *None*.

get_placement()
Generates normal (Gaussian) placement coordinates for the given number of nodes and returns the result as a dictionary.

The means used here are computed as follows: $\frac{1}{2} \times area\ width$ and $\frac{1}{2} \times area\ height$.

***Returns*:** A list of tuples of horizontal and vertical coordinates for each host.

Module placement.uniform

Provides an implementation of the uniform placement model.

In the uniform placement model, a simulation area of a given size is chosen and a given number of nodes are placed over it with the uniform probability distribution.

class sim2net.placement.uniform.**Uniform**(*area*, *nodes_number*)
Bases: sim2net.placement._placement.Placement

This class implements implements the uniform placement model, in which a given number of nodes are placed over a simulation area with the uniform probability distribution.

***Parameters*:**

- **area**: an object representing the simulation area;
- **nodes_number** (*int*): a number of nodes to place over the simulation area.

***Raises*:**

- **ValueError**: raised when the number of nodes is less or equal to 0, or when the given value of the *area* parameter is *None*.

get_placement()
Generates uniform placement coordinates for the given number of nodes and returns the result as a dictionary.

***Returns*:** A list of tuples of horizontal and vertical coordinates for each host.

3.3.7 Package `sim2net.propagation`

This package provides a collection of wireless signal propagation model classes.

A wireless transmission may be distorted by many effects such as free-space loss, refraction, diffraction, reflection or absorption. Therefore, wireless propagation models describe the influence of environment on signal quality (mainly as a function of frequency, distance or other conditions) and calculate the **signal-to-noise ratio** (*SNR*) at the receiver. Then, it is assumed that if the SNR value is higher than some prescribed threshold, the signal can be received, and the packet that is carried by the signal can be successfully received if the receiving node remains connected in this way with the sending node at least for the duration of that packet transmission.

Package modules:

- Module `sim2net.propagation._propagation`
- Module `sim2net.propagation.path_loss`

See also:

`sim2net.packet_loss`

Module `sim2net.propagation._propagation`

Contains an abstract class that should be implemented by all wireless signal propagation model classes.

class `sim2net.propagation._propagation.`**`Propagation`**(*name*)

Bases: `object`

This class is an abstract class that should be implemented by all wireless signal propagation model classes.

Parameters: - **name** (*str*): a name of the implemented placement model.

`get_neighbors`(*coordinates*)

Calculates identifiers of all nodes in a network that would be able to receive a wireless signal transmitted from a source node, according to the implemented propagation model. All nodes in the network are considered, one by one, as the source node.

***Parameters*:**

- **coordinates** (*list*): a list of coordinates of all nodes in the simulated network at the current simulation step.

***Returns*:** A *list* that in position `i` is a list of all nodes that would be able to receive a wireless signal transmitted by a node whose identifier is equal to `i`.

***Raises*:**

- **NotImplementedError**: this method is an abstract method.

`logger`

(*Property*) A logger object of the `logging.Logger` class with an appropriate channel name.

See also:

`sim2net.utility.logger`

`random_generator`

(*Property*) An object representing the `sim2net.utility.randomness._Randomness` pseudo-random number generator.

Module sim2net.propagation.path_loss

This module provides an implementation of the simplified path loss model.

The path loss model predicts the reduction in attenuation (power density) a signal encounters as it propagates through space. In this simplified implementation, it is presumed that for all nodes that are within transmission range of each other, the **signal-to-noise ratio** (*SNR*) is above the minimal usable level, and hence, the nodes are able to communicate directly.

class sim2net.propagation.path_loss.**PathLoss**(*transmission_range*)

Bases: sim2net.propagation._propagation.Propagation

This class implements simplified path loss model in which the signal-to-noise ration is calculated on the given value of the transmission range of nodes.

***Parameters*:**

- **transmission_range** (*float*): a value of the transmission (or communication) radius of nodes, that is, the distance from a transmitter at which the signal strength remains above the minimum usable level.

***Raises*:**

- **ValueError**: raised when the given transmission range is less or equal to 0.

_PathLoss__distance(*source_coordinates*, *destination_coordinates*)

Calculates the distance between source and destination nodes in Cartesian space.

***Parameters*:**

- **source_coordinates** (*list*): values of the source node's horizontal and vertical coordinates at the current simulation step;
- **destination_coordinates** (*list*): values of the destination node's horizontal and vertical coordinates at the current simulation step.

***Returns*:** The distance between source and destination nodes in Cartesian space of type *float*.

get_neighbors(*coordinates*)

Calculates identifiers of all nodes in a network that would be able to receive a wireless signal transmitted from a source node, according to the implemented propagation model. All nodes in the network are considered, one by one, as the source node.

***Parameters*:**

- **coordinates** (*list*): a list of coordinates of all nodes in the simulated network at the current simulation step.

***Returns*:** A *list* that in position `i` is a list of all nodes that would be able to receive a wireless signal transmitted by a node whose identifier is equal to `i`.

Examples:

```
>>> pathloss = PathLoss(1.0)
>>> coordinates = [[1.0, 2.0], [1.5, 2.5], [2.0, 3.0], [2.5, 3.5]]
>>> print pathloss.get_neighbors(coordinates)
[[1], [0, 2], [1, 3], [2]]
>>> coordinates = [[1.0, 2.0], [1.1, 2.1], [1.2, 2.2], [1.3, 2.3]]
>>> print pathloss.get_neighbors(coordinates)
[[1, 2, 3], [0, 2, 3], [0, 1, 3], [0, 1, 2]]
```

3.3.8 Package `sim2net.speed`

This package provides a collection of speed distribution classes.

Speed is a scalar quantity that describes the rate of change of a node position in a simulation area (see: `sim2net.area`).

Note: In all speed distribution classes the quantity of speed should be considered as simulation area units per one *simulation time* unit (see: `sim2net._time`).

For example, the value of speed equal to 5 would mean *five units of simulation area per one unit of simulation time*.

Package modules:

- Module `sim2net.speed._speed`
- Module `sim2net.speed.constant`
- Module `sim2net.speed.normal`
- Module `sim2net.speed.uniform`

See also:

`sim2net.placement`, `sim2net._time`

Module `sim2net.speed._speed`

Contains an abstract class that should be implemented by all speed distribution classes.

class `sim2net.speed._speed.`**Speed**(*name*)

Bases: `object`

This class is an abstract class that should be implemented by all speed distribution classes.

Parameters: - **name** (*str*): a name of the implemented speed distribution.

current

(*Property*) A value of the current speed of type *float*.

***Raises*:**

- **NotImplementedError**: this property is an abstract property.

get_new()

Assigns a new speed value.

***Returns*:** (*float*) a new speed value.

***Raises*:**

- **NotImplementedError**: this method is an abstract method.

logger

(*Property*) A logger object of the `logging.Logger` class with an appropriate channel name.

See also:

`sim2net.utility.logger`

random_generator

(*Property*) An object representing the `sim2net.utility.randomness._Randomness` pseudo-random number generator.

Module sim2net.speed.constant

Provides an implementation of a constant node speed. In this case a speed of a node is constant at a given value.

class sim2net.speed.constant.**Constant**(*speed*)
Bases: sim2net.speed._speed.Speed

This class implements a constant node speed fixed at a given value.

***Parameters*:**

- **speed** (*float*): a value of the node speed.

Example:

```
>>> speed = Constant(5.0)
>>> speed.current
5.0
>>> speed.get_new()
5.0
>>> speed = Constant(-5.0)
>>> speed.current
5.0
>>> speed.get_new()
5.0
```

current
(*Property*) The absolute value of the current speed of type *float*.

get_new()
Returns the absolute value of the given node speed of type *float*.

Module sim2net.speed.normal

Provides an implementation of the normal speed distribution. In this case a speed of a node is assigned at random with the normal, i.e. Gaussian, probability distribution.

class sim2net.speed.normal.**Normal**(*mean=0.0*, *standard_deviation=0.2*)
Bases: sim2net.speed._speed.Speed

This class implements the normal speed distribution that assigns node's speeds with the Gaussian probability distribution.

(Defaults to **standard normal distribution**.)

***Parameters*:**

- **mean** (*float*): a value of the expectation (default: *0.0*);
- **standard_deviation** (*float*): a value of the standard deviation (default: *0.2*).

current
(*Property*) A value of the current speed of type *float* (or *None* if the value has yet not been assigned).

get_new()
Assigns a new speed value.

> **Warning:** Depending on distribution parameters, negative values may be randomly selected.

***Returns*:** (*float*) the absolute value of a new speed.

mean
(*Property*) A value of the expectation of type *float*.

Module sim2net.speed.uniform

Provides an implementation of the uniform speed distribution. In this case a speed of a node is assigned at random with the uniform probability distribution.

class sim2net.speed.uniform.**Uniform**(*minimal_speed*, *maximal_speed*)
Bases: sim2net.speed._speed.Speed

This class implements the uniform speed distribution that assigns node's speeds from a given range with equal probability.

Parameters: - **minimal_speed** (*float*): a value of a node's minimal speed; - **maximal_speed** (*float*): a value of a node's maximal speed.

current
(*Property*) A value of the current speed of type *float* (or *None* if the value has yet not been assigned).

get_new()
Assigns a new speed value.

> **Warning:** Depending on distribution parameters, negative values may be randomly selected.

***Returns*:** (*float*) the absolute value of a new speed.

3.3.9 Package sim2net.utility

This package contains miscellaneous utility modules and classes.

Package modules:

- Module sim2net.utility.logger
- Module sim2net.utility.randomness
- Module sim2net.utility.validation

Module sim2net.utility.logger

Provides functions which implement an event logging system with the use of the logging module from the standard library.

class sim2net.utility.logger.**Sim2NetFormatter**(*time=None*)
Bases: logging.Formatter

Implements a custom logging.Formatter that can also log simulation steps and time (see: sim2net._time).

Parameters: - **time**: a simulation time object of the

sim2net._time.Time class to log simulation steps and time.

format(*record*)
Formats the specified record as text and adds the current simulations step and time if the time object is present.

sim2net.utility.logger.**__channel_string**(*channel*)
Returns a logging channel string for a given string.

sim2net.utility.logger.**create_logger**(*time=None*, *level=None*, *handler=None*, *formatter=None*)
Creates and configures a logger for the main logging channel.

If no *handler* is passed, the sim2net.utility.logger.Sim2NetFormatter formatter is used.

***Parameters*:**

- **time**: a simulation time object of the sim2net._time.Time class to log simulation steps and time;
- **level**: a logging level that will be set to the logger (and its handler if the handler is not passed as an argument); the level can be passed as a string or a logging module's level;
- **handler**: an object representing the handler to be used with the logger (see logging.handlers in the standard library);
- **formatter**: an object representing the log format to be used with the logger's handler (see logging.Formatter class in the standard library).

***Returns*:** A logging.Logger object for a newly created logger.

sim2net.utility.logger.**get_logger**(*channel=None*)
Returns a logger object. Multiple calls to this function with the same channel string will return the same object.

***Parameters*:**

- **channel** (*str*): a string that represents a logging channel.

***Returns*:** A logging.Logger object for the given logging **channel** or the main channel logger if **channel** argument is *None*.

Examples:

```
>>> main_channel_logger = logger.create_logger()
>>> main_channel_logger = logger.get_logger()
>>> new_channel_logger = logger.get_logger('my_channel')
```

Module sim2net.utility.randomness

Provides a pseudo-random number generator.

class sim2net.utility.randomness.**_Randomness**
Bases: object

This class provides a pseudo-random number generator with the use of the random module from the standard library that produces a sequence of numbers that meet certain statistical requirements for randomness.

get_state()
Returns an object capturing the current internal state of the generator.

This object can be passed to set_state() to restore the state.

normal(*mikro*, *sigma*)
Returns a random floating point number with the normal (i.e. Gaussian) distribution.

***Parameters*:**

- **mikro** (*float*): a value of the mean to be used by the generator;
- **sigma** (*float*): a value of the standard deviation to be used by the generator.

random_order(*sequence*)
Shuffles the given **sequence** *in place*.

set_state(*generator_state*)
Sets a new internal state of the generator.

The state can be obtained from a call to `get_state()` method.

***Parameters*:**

- **generator_state**: an internal state of the generator to set.

***Raises*:**

- **ValueError**: raised when a given value of the *generator_state* parameter is *None*.

uniform(*begin*, *end*)
Returns a random floating point number N such that $beginNend$ for $beginend$ and $endNbegin$ for $end < begin$.

sim2net.utility.randomness.**get_random_generator**()
Returns an object representing the `_Randomness` pseudo-random number generator. Multiple calls to this function will return the same object.

Module sim2net.utility.validation

Contains a collection of source code validation functions.

sim2net.utility.validation.**check_argument_type**(*function*, *parameter*, *expected_type*, *argument*, *logger=None*)
Checks whether a given argument is of a given type and raises an exception or reports a log message if the argument's type is inappropriate.

Checks whether a value of the *argument* parameter is of the *expected_type* type. If not, it raises an exception (if *logger* object is *None*) or reports a log message (if *logger* object is passed) indicating an inappropriate type of the *parameter* parameter in the *function* function (or method).

***Parameters*:**

- **function** (*str*): a name of the function which argument is to be checked;
- **parameter** (*str*): a name of the parameter which argument is to be checked;
- **expected_type**: an expected type of the *argument* parameter;
- **argument**: a value of the argument that is to be checked;
- **logger** (`logging.Logger`): a logger object that will be used to write the log message.

***Raises*:**

- **TypeError**: raised when the value of *argument* is not of the *expected_type* type and *logger* object is not passed.

Example:

```
>>> check_argument_type('function_name', 'parameter_name', str, 'argument')
```

CHAPTER 4

Indices and tables

- *genindex*
- *modindex*
- *search*

CHAPTER 5

Links

Repository https://github.com/mkalewski/sim2net

Bug reports https://github.com/mkalewski/sim2net/issues

Documentation https://sim2net.readthedocs.org/en/latest/

CHAPTER 6

Copyright

Bibliography

[CGR11] Christian Cachin, Rachid Guerraoui, Luís Rodrigues. Introduction to Reliable and Secure Distributed Programming, 2ed Edition. Springer-Verlag, 2011.

[LNR04] Guolong Lin, Guevara Noubir, Rajmohan Rajamaran. Mobility Models for Ad-Hoc Network Simulation. In Proceedings of the *23rd Conference of the IEEE Communications Society* (INFOCOM 2004), pp. 463-473. Hong Kong, March 2004.

[CBD02] Tracy Camp, Jeff Boleng, Vanessa Davies. A Survey of Mobility Models for Ad-Hoc Network Research. In *Wireless Communications Mobile Computing. Special Issue on Mobile Ad Hoc Networking: Research, Trends and Applications*, vol. 2(5), 483–502. John Wiley & Sons, 2002.

[LH99] Ben Liang, Zygmunt J. Haas. Predictive Distance-Based Mobility Management for PCS Networks. In Proceedings of the *18th Annual Joint Conference of the IEEE Computer and Communications Societies* (INFOCOM 1999), pp. 1377–1384, vol. 3. New York, NY, United States, March 1999.

[RMM01] Elizabeth M. Royer, P. Michael Melliar-Smithy, Louise E. Moser. An Analysis of the Optimum Node Density for Ad Hoc Mobile Networks. In Proceedings of the *IEEE International Conference on Communications* (ICC 2001), pp. 857–861, vol. 3. Helsinki, Finland, June 2001.

[JM96] David B. Johnson and David A. Maltz. Dynamic Source Routing in Ad Hoc Wireless Networks. In *Mobile Computing*, edited by Tomasz Imielinski and Hank Korth, chapter 5, pp. 153–181. Kluwer Academic Publishers, 1996.

[BMJ+98] Josh Broch, David A. Maltz, David B. Johnson, Yih-Chun Hu, Jorjeta Jetcheva. A Performance Comparison of Multi-hop Wireless Ad Hoc Network Routing Protocols. In Proceedings of the *4th Annual ACM/IEEE International Conference on Mobile Computing and Networking* (MobiCom 1998), pp. 85–97. Dallas, Texas, United States, October 1998.

[Ell63] E. O. Elliott. Estimates of Error Rates for Codes on Burst-Noise Channels. In *Bell System Technical Journal*, vol. 42(5), 1977–1997. Bell Laboratories, September 1963.

[Gil60] Edgar Nelson Gilbert. Capacity of a Burst-Noise Channel. In *Bell System Technical Journal*, vol. 39(5), 1253–1265. Bell Laboratories, September 1960.

[HH08] Gerhard Haßlinger, Oliver Hohlfeld. The Gilbert-Elliott Model for Packet Loss in Real Time Services on the Internet. In Proceedings of the *14th GI/ITG Conference on Measurement, Modelling and Evaluation of Computer and Communication Systems* (MMB 2008), pp. 269–286. Dortmund, Germany, April 2008.

S