Models and Simulation for CORMORAN

Body Area Networks

CORMORAN – 30 January 2012

Arturo Mauricio JIMENEZ GUIZAR
Claire GOURSAUD
Jean-Marie GORCE
Isabelle AUGE-BLUM

INSA de Lyon
INRIA - CITI Laboratory
Centre of Innovation in Telecommuncations and Integration of services

Key Words: *Game Theory, Wireless Sensor Network, Scheduling, optimization, Body Area Networks*

Project Acronym	CORMORAN (ANR 11-INFR-010)
Document Title	Models and Simulation for CORMORAN
Contractual Date of Delivery to ANR	None
Contractual Date of Delivery to ANR	None
Editor	INSA
Authors	CEA, INSA, TPT, UR1
Participants	CEA, INSA, TPT, UR1
Related Task(s)	T2
Related Sub-Task(s)	T3.4
Security	Private
Nature	Technical Report
Version Number	0.1
Total Number of Pages	13

ST-2.4 Models and Simulation for CORMORAN

1. Introduction

In the context of the CORMORAN project (Subtask 2.4), it **is desirable to build a dedicated inter-WBAN multi-channel simulator** with the following features:

- IR–UWB RAT capabilities
- Realistic short-term and long-term pedestrian mobility models
- On-body and off-body radio simulation capabilities
- Graph-oriented Ray Tracing tool for calculus acceleration and incremental prediction capabilities
- Multi-link and multi-RAT simulation capabilities required within cooperative scenarios

2. Motivations

The idea of this tutorial is **to create a simulation tool with an interfacing between WSNet and PyLayers.** PyLayers should provide the information of the quality channel links associated with network nodes. Then, the data should be processed by WSNet for the upper layers and followed by calculating the performance of the simulated scenario.

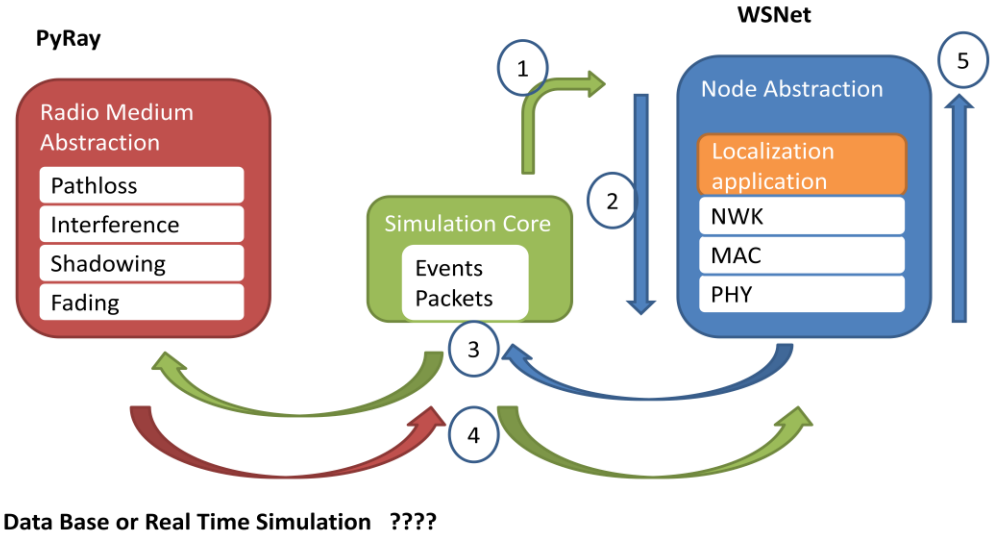
2.1 Discussions and open questions

On the last meeting at Paris, some points were discussed:

- Probably easier to develop routing and location algorithms directly on WSNet
- Current radio models in WSNet are inappropriate for CORMORAN purposes (e.g. on-body SNR and error ranging conditional models missing)

2.2 Architecture of the simulator

Furthermore, we propose two solutions for the simulator platform: **simulator based on a database or real-time data**.

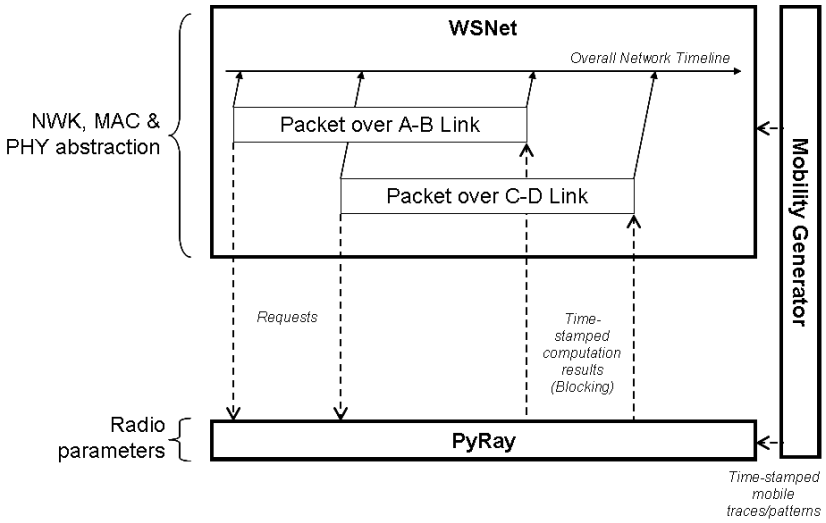


- **Database solution:** In this alternative, there is a database that stocks the results of the UR1 PyRay determinist simulator. The data is specially the quality of channel links and it is used after by WSNet for the upper layers overall network timeline.

  The points to consider in this solution concern the PHY layer on WSNet:
  - The PHY layer can be created by new models (this means to code a PHY layer for WSNet and this take some time).
  - We **use xml files where one would explicitly write traces (csv files) containing the radio metrics** based on the mobility and results of PyRay (apparently easier solution).
  - Mobility can be supported by WSNet

  Another problem for this solution resides in **choice of the master/slave mode of the simulation** for the layers management. It's logical that the upper layers (MAC/NWK) have to been the master to launch the PHY layer as slave. From this approach, WSNet will be the master simulator to launch the simulation and there is a scheduler that will manage the time to launch each process, including the PyRay calculation. However, we have to consider the calculation time and the sampling method (Time Stamp, Node Id, X, Y, Z, Link Quality Indicator, trajectories calculation).

  So one solution is to take PyRay as slave of WSNet:
  - To reduce the generation of several xml files
  - Mobility can be treated separately (e.g. through multi-agent model of UR1 then processed via database with respect to PyRay)
  - **Physical time simulation by request is still an open question** to validate the feasibility of this architecture



- **Real Time solution** : the idea is to simulate on real time the PyLayers mobility feature with statistical radio channel models (PHY layer) and WSNet (upper layers)

  For this solution, the communication of simulators has to be considered:

  Unidirectional: WSNET → PyLayers, in this solution WSNet is the master that calls PyLayers functions when necessary.

  Bidirectional: WSNET ↔ PyLayers, both simulators can be used separately and they can be master or slave depending on the simulated scenario.

  This solution is more complex to develop, that's why we should test the first solution in short-term. The first step is to test the interface of the multi-agent mobility model UR1 for off-body radio & body-to-body to generate the first useful lower-level simulation for the group navigation scenario and cooperative inter-WBAN communication.

3. Tutorial to understand WSNET simulation for CORMORAN

3.1 Goal

The goal of this tutorial is to show the basics to simulate a scenario BAN with WSNET. For this purpose we are going to simulate a BAN with 13 nodes:

- Hip, back, right thigh, right foot, left thigh, left foot, torso, right arm, right hand, left arm, left hand, right ear and left ear
- Simulation parameters: duration= 50s
- The channel status of each link between the nodes is given in a csv file that contains the shadowing of the channel through the time. To read the csv files we need a special library (libcsv) that can be found in the CORMORAN sharing site.

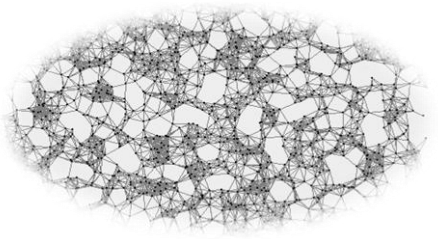
In WSNet there are several modules to simulate a network; however there are not modules to simulate the CORMORAN project needs so we need to create specific models. In this tutorial we will learn the basics of writing a new application module for WSNet.

First, we will see the scope of WSNet. Second, we show the basics to make a simulation with xml files. Then, we present the requirements for building a new module and we take as an example a propagation model to compute the fading on BAN's links. After that, we implement it as an external module. Finally, we install the new module and we run some simulations.

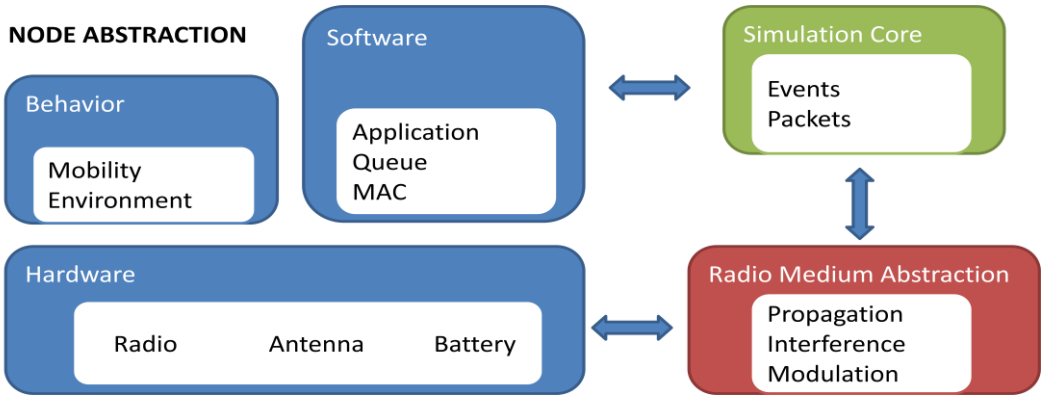
3.2 WSNet scope

WSNet is a simulator for wireless networks on a large scale developed at the CITI laboratory.

- **Written in C**
- Uses **free and accessible libraries**
- Configured using **XML files** to simulate
- Management of a **massive number of nodes**
- Generation of random geometric graphs
- Uses a **"realistic" physical layer to develop algorithms for the upper layers**.



WSNet architecture is represented as blocks modeling the properties of the radio medium, the simulation core and the characteristics of simulated nodes.



- **The simulator** core allows generating and processing **the events and the packets** of the simulation. This is a key element of the simulator performance because it determines the speed degree and the effectiveness of simulations.
- **The radio medium abstraction** consists of three sub-blocks: **propagation, interference and modulation**.
- **The node abstraction** is composed of three sub-blocks:
    - **Behavior** describes the mobility and the environment in which the nodes can evolve (fire, obstacles, etc)
    - **Hardware** defines the different physical properties of each node (TX/RX frequency, TX power, battery)
    - **Software** describes the upper layer of the simulation (MAC, NWK and application layer)

3.3 Installing WSNet

Before starting the implementation of a new application module, we have to make sure that WSNet if correctly installed. If you have already installed the simulator you can skip to the next section. To install WSNet, you have to perform the following steps:

a) First of all, make sure that the software package are installed on your Linux system:
- gcc, libtool and make
- autoconf (>= 2.61), and pkg-config (>= 0.21)
- sun-java5-jdk, or openjdk-6-jdk (for graphical tools)
- libglib2.0-0, and libglib2.0-dev
- libxml2, and libxml2-dev
- libgsl0-dev

b) Downloading the WSNet source code from the SVN repository by typing the following command:

```
$ svn checkout svn://scm.gforge.inria.fr/svn/wsnet
```

c) Compiling the source code by typing the following commands within the WSNet archive directory:

```
$ cd ./wsnet/
$ ./bootstrap
$ ./configure
$ make
```

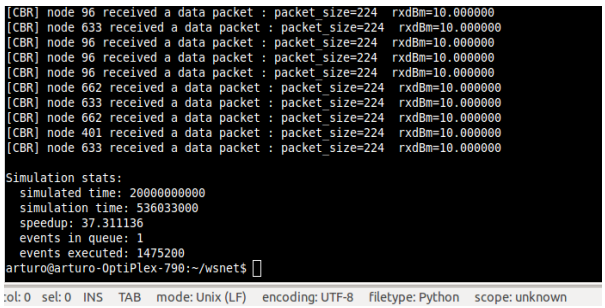
d) Installing WSNet by typing the following command (with root privileges) within the WSNet archive directory:

```
# sudo make install
```

e) Then, once WSNet is properly compiled and installed you can run a simulation by executing the wsnet binary file. For running the the cbr.xml config file, one would type the following command:

```
PATH=$PATH:/usr/local/wsnet-2.0/bin
export PATH
# wsnet -c /usr/local/wsnet-2.0/demo/cbr.xml
```

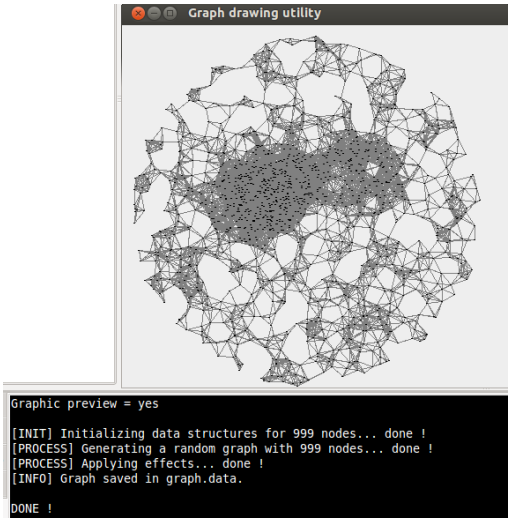
Now, the simulation should work and you can see the simulation stats at the end.



f) We can try the topogen tool to see the topologies of a network simulation. For that you need to install gnuplot and then run topogen.

```
$ apt-get -f install gnuplot-nox
$ apt-get -f install gnuplot-x11
# wsnet-topogen --topo circle 200 20 --distribution uniform 0.008 --effect cluster
4 50 0.05 --output graph.data -gp
```

So you will obtain something like this but may be not the same distribution of nodes.



In case of compilation errors (or for more details) please read the WSNet installation tutorial for a more complete guide.

3.4 Files xml

In WSNet, we use xml files to configure a simulation (in the installation we ran cbr.xml as simulation). This file describes the simulation setup and specifies, for example, the number of nodes to simulate, the libraries used to model the radio medium and the nodes (e.g., for propagation, routing, etc). It is composed of five main sections:

- **Global parameters** allow defining a simulation with a specific syntax. These parameters are ***number of nodes, duration, width (x), length (y) and height (z) of the simulation*****.**

- **Entities** are an instantiation of dynamic libraries. These libraries can be instantiated several times in different entities. Each of these entities may have different **global-parameter values** and **node-parameter default** values. The parameters are ***name of entity, name of the dynamic library, global parameters (init) and node parameters (default)*****.**

- **Environment** defines the radio medium and the physical environment of the simulation. The parameters are ***the name of propagation, the name of interferences, the name of a monitoring entity, the name of one modulation entity (modulation should appear at least once) and the name of one environment entity (with - optional).***

- **Bundles** correspond to the node architecture abstraction. As we said before for the node abstraction, the parameters are ***the name of bundle, default bundle (to specify if this bundle is the default one or not, values true or false), default birth (optional, default value = 0), the mobility entity, the energy entity, the antenna entity.***

Then there are more options that can be repeated several times: ***with to define specific entities in the bundle such as the upper layers, the uplink, the downlink and the default values depending on the entity description(*)(**).***

- **Node** instantiates a bundle and this define the node behavior. The parameters are the ***node id, the bundle name, the birth date of node (optional) and for (optional) that is used to override parameters of an entity defined in the bundle.***

*** It is advisable to see the parameters of libraries on the models WSNet site to configure entities, environment and bundles.**

**** For more details go to the WSNet xml configuration link.**

The simulation configuration file is specified using the command line option -c. Example: wsnet -c config.xml. This allows you to launch the simulation.

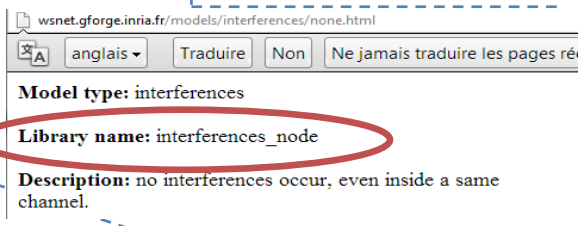
The next example is the one used for the CORMORAN project:

```
<?xml version='1.0' encoding='UTF-8'?>
<worldsens xmlns="http://worldsens.citi.insa-lyon.fr">

<!-- == Worldsens ========================================================= -->
<simulation nodes="13" duration="50s" x="2" y="2" z="2"/>

<!-- == Entities ========================================================== -->

<!-- == PROPAGATION, INTERFERENCES, MODULATION and ENERGY ================= -->
<entity name="ban_propag" library="propagation_ban" >
        <init data_description_file="user_models/ban_links_description.csv" fading_model="rician" />
</entity>

<entity name="mod_oqpsk" library="modulation_oqpsk">
</entity>

<entity name="interf" library="interferences_none">
</entity>

<entity name="bat" library="energy_linear">
        <default energy="100000" tx="0.00468" rx="0.0555" idle="0"/>
</entity>

<!-- == RADIO and ANTENNA ================================================= -->
<entity name="omnidirectionnal" library="antenna_omnidirectionnal" >
  <default loss="0" angle-xy="random" angle-z="random"/>
</entity>

<entity name="radio" library="radio_half1d" >
   <default sensibility="-92"  T_s="90" dBm="0" channel="0" modulation="mod_oqpsk"/>
</entity>

<!-- == MAC =============================================================== -->

<entity name="mac" library="mac_802_15_4_2400_oqpsk_u_csma_ca" >
</entity>

<!-- == Application ======================================================= -->

<entity name="hip_app" library="application_cbr" ></entity>
<entity name="back_app" library="application_cbr" ></entity>
<entity name="rthigh_app" library="application_cbr" ></entity>
<entity name="rfoot_app" library="application_cbr" > </entity>
<entity name="lthigh_app" library="application_cbr" ></entity>
<entity name="lfoot_app" library="application_cbr" ></entity>
```

All xml files for WSNET start and finish with <worldsens> instance. See at the end of the example.

This part corresponds to the **global parameters.** We assume that 13 nodes are deployed over an area of 2x2x2 during 50 seconds.

After global parameters, we define the **entities** and the instances to the libraries and models. In the first entity, **we found the instance of our model ban_propag** explained in 3.5. **We configure our propagation model with a rician fading**



Then we define modulation, interferences and energy. Be careful with the name of the library instance. This is show in the models table WSNet site. The name is up to the user to define and this is the name to call the entity after.

Next, we define the radio and antenna entities of the PHY layer. In the example we have an omnidirectionnal antenna and the radio is half duplex.

After that, we define the MAC layer as the IEEE 802.15.4 at 2.4 GHz using QPSK and CSMA.

```
<entity name="torso_app" library="application_cbr" ></entity>
<entity name="rarm_app" library="application_cbr" ></entity>
<entity name="rhand_app" library="application_cbr" ></entity>
<entity name="larm_app" library="application_cbr" ></entity>
<entity name="lhand_app" library="application_cbr" ></entity>
<entity name="rear_app" library="application_cbr" ></entity>
<entity name="lear_app" library="application_cbr" ></entity>

<!-- == MOBILITY ======================================================= -->
<!--
<entity name="static-pnc" library="mobility_static" >
  <default x="0.433" y="0.433" z="0"/>
</entity>
<entity name="static-dev1" library="mobility_static" >
  <default x="0.683" y="0.866" z="0"/>
</entity>
<entity name="static-dev2" library="mobility_static" >
  <default x="0.183" y="0.866" z="0"/>
</entity> -->

<entity name="static" library="mobility_static">
        <default x="random" y="random" z="random"/>
</entity>
```

Subsequently, the application layer is defined for each node with a simple Content Based Routing.

Then you can define the mobility of the simulation. In the example with put this feature as comments and we define a static mobility entity instead.

```
<!-- == Environment ==================================================== -->

<environment>
        <propagation entity="ban_propag"/>
        <interferences entity="interf"/>
        <modulation entity="mod_oqpsk"/>
</environment>
```

After entities, we define the **environment**, in our example, the propagation is the ban_propag entity created for BANET. Then we define the environment without interferences and with a QPSK modulation.

```
<!-- == Bundle ========================================================= -->
<bundle name="hip" worldsens="false" default="true" birth="0">
        <mobility entity="static"/>
        <antenna entity="omnidirectionnal">
                <up entity="radio"/>
        </antenna>
        <with entity="radio">
                <up entity="mac"/>
                <down entity="omnidirectionnal"/>
        </with>
        <with entity="mac">
                <up entity="hip_app"/>
                <down entity="radio"/>
        </with>
        <with entity="hip_app">
                <down entity="mac"/>
        </with>
</bundle>

……

<bundle name="lear" worldsens="false" default="false" birth="0">
        <mobility entity="static"/>
        <antenna entity="omnidirectionnal">
                <up entity="radio"/>
        </antenna>
        <with entity="radio">
                <up entity="mac"/>
                <down entity="omnidirectionnal"/>
        </with>
        <with entity="mac">
                <up entity="lear_app"/>
                <down entity="radio"/>
        </with>
        <with entity="lear_app">
```

After the environment, we define the **bundles**, in our example we define a bundle for each node. In this tutorial, we only show two of the bundles.

The first one is for the hip node. It is composed by the static mobility entity, followed by an omnidirectional antenna entity and remark the up entity that is the radio.

Up and down entities, we define the chain of the PHY, MAC, NWK and application layers. You have to define each layer with <with> in the correct order.

The last with entity correspond to the application layer hip_app defined before with the Content Based Routing.

Then we do the same for the left ear node.

```
                    <down entity="mac"/>
            </with>
</bundle>
```

```
<!-- == Nodes ================================================== -->

<node id="0" as="hip"/>
<node id="1" as="back"/>
<node id="2" as="rthigh"/>
<node id="3" as="rfoot"/>
<node id="4" as="lthigh"/>
<node id="5" as="lfoot"/>
<node id="6" as="torso"/>
<node id="7" as="rarm"/>
<node id="8" as="rhand"/>
<node id="9" as="larm"/>
<node id="10" as="lhand"/>
<node id="11" as="rear"/>
<node id="12" as="lear"/>
```

At the end, we define each node with their corresponding bundle.

```
</worldsens>
```

This define the end of the simulation xml file

3.5 *Models* (creation, structure, compilation)

During a simulation with WSNet, the entities on the xml file have an instance with a module. The modules are implementations of network models (radio propagation, interferences, mobility, MAC, application, etc).

There are a certain number of models included in the online version of WSNet. However, for the CORMORAN goal, we need to create our own models to support the different algorithms. The Fig XX shows the different models existing in WSNet, you can find it on the WSNet site. There you can click on each model to have the details to configure it correctly such as *model type, library name, description and entity parameters,* these parameters are used in the xml file configuration. It is important to know the library name because is the name needed to put in the xml files to call the functions of the model. Then it is preferable to see the entity parameters description to know the values you can use for the simulation.

Models	Included in WSNet distribution
Radio propagation	file static, disk model (range), free space, tworay ground, lognormal shadowing, rayleigh fading, ITU indoor model, nakagami fading
Interferences	none, orthogonal, factor
Modulation	none, step, bpsk, oqpsk, mqam
Antenna	omnidirectional
Mobility	static, file static, billiard, torus central, torus plane, teleport
Battery/energy	linear
Environment	fire
Monitor	nodes, nrj
Radio	half1d, 802.15.4 868MHz bpsk, 802.15.4 902MHz bpsk, 802.15.4 2400MHz oqpsk
MAC	802.11 DCF, 802.15.4 868MHz bpsk, 802.15.4 902MHz bpsk, 802.15.4 2400MHz oqpsk, B-MAC, Ideal MAC
Routing	greedy geographic, file static
Application	CBR, CBR_v2, Hello protocol, GHT, LBDD, XY, Data_Sink, Data_Source, GOSSIP, B-MAC application sample

Creation: The implementation of a new module is made by adding the source code of our module to the */wsnet/user_models/* directory. Normally this directory has been installed before, if not please follow the instructions to set up the working directory.

Structure: Each module has a different architecture depending on the methods and the application of the model. But the basic architecture includes the following source code:

```
#include <include/modelutils.h>

model_t model = {
        "MODULE DESCRIPTION",
        "AUTHORS",
        "0.1",
        MODELTYPE_APPLICATION,
        {NULL, 0}
};

int init(call_t *c, void *params);

int destroy(call_t *c);

int setnode(call_t *c, void *params);

int unsetnode(call_t *c);

int bootstrap(call_t *c);

void rx(call_t *c, packet_t *packet);

/* ************************************************** */
application_methods_t methods = {rx};
```

This library is always present in the implemented modules. You can also include other libraries needed for your model.

This is a data structure describing the module (authors, version, etc). It is necessary to call the model with the xml files. Make sure there is only one model_t model structure in the code.

You can also create more structures for your model as another C program.

The init fuction is necessary to initialize the application entity and the global entity parameters (values taken from the xml config file). **Destroy is a function called at the end of the simulation to destroy the entity**.

Setnode, unsetnode, bootstrap are methods used to make the interaction of the network nodes. These methods are useful for application implementation; however we are not using them in our example for BAN because we use csv files to take some nodes parameters for the simulation.

Finally**, the rx method is the one who contains your application model** and make the calculations when a packet is received. This function must be called at the end as an application method. **The names can change between the different modules**. In our example, this method is called propagation.

Case Study – Deterministic BAN channel propagation layer: As we said before, the goal is to simulate a BAN with 13 nodes. First, we need to implement a specific propagation model to compute the status of each channel link between the nodes. For that, the module will take entities and global parameters from the xml file (ban_cea.xml) presented in section 3.4. Then, the module will use csv files to take the shadowing values of the channel links of each BAN node and then compute the fading of these links. Finally, at the end of the simulation we can see all the status of links.

```
/**
 * \file   ban-propag.c
 * \brief  Deterministic BAN channel propagation layer
 * \author Paul Ferrand, Javier Cuadrado
 * \date   2010
 **/
```

The first comments are a description of the module

```
#include <include/modelutils.h>
#include <math.h> // Use fmod(x,y)
#include "libcsv/csv.h"
```

Here we call the modelutils.h library that is necessary for all implementation. Then libcsv/csv.h is a library that contains the methods to manipulate the csv files

```
/* ************************************************** */
/* ************************************************** */
model_t model =  {
   "BAN propagation model",
   "Paul Ferrand, Javier Cuadrado",
   "0.1",
   MODELTYPE_PROPAGATION,
   {NULL, 0}
};
```

Data structure to describe the model: modeltype propagation

```
#define
...
#endif

/* ************************************************ */
/* ************************************************ */
typedef struct samples
{       ...         } t_samples;

struct entitydata
 {      ...         };

typedef struct csv_params
{       ...         } t_csv_params;
```

After we create some structures to manipulate the csv files data. We want to parse the shadowing into tables to compute the fading.

```
/* ************************************************ */
/* ************** Normal distribution ************** */
/* ************************************************ */
double normal (double avg, double deviation)
{                   ...                 }
```

This function generates a distributed random number (Normal distribution Box Muller transform) given a source of uniformly distributed random numbers. **It is used to compute the fading** (Rayleigh and Rice)

```
/* ************************************************ */
/* ************ Manipulate sample tables ************ */
/* ************************************************ */

int init_tables(struct entitydata * entitydata)
{       ...         }

__inline__
t_samples * get_samples_t( struct entitydata * entitydata, int src, int dest)
{       ...         }

int free_tables(struct entitydata * entitydata)
{       ...         }

void reset_csv_params( t_csv_params * csv_params )
{       ...       }

/* ************************************************ */
/* ************** Callbacks for libcsv ************** */
/* ************************************************ */

void _links_field_read(void *s, size_t i, void *p)
{       ...         }

void _links_entry_read(int i, void *p)
{       ...         }

void _config_field_read(void *s, size_t i, void *p)
{       ...         }

void _config_entry_read(int i, void *p)
{       ...         }
```

These are methods to manipulate the tables and csv files.

```
/* ************************************************ */
/* ************************************************ */
int init(call_t *c, void *params)
{       ...         }

int destroy(call_t *c)
{       ...         }
```

We can find aswell the init and destroy functions to create and destroy the entity.

```
/* ************************************************** */
/* ************************************************** */
double compute_ban_fading(double pathloss, int fading_model)
{
  ...
  {
        case BF_NONE:
            ...
        case BF_RAYLEIGH:
            ...
        case BF_RICE:
            ...
        case BF_NAKAGAMI:
            ...
  }
 ...
}
```

This function computes the fading, it takes into account the fading model chosen in the xml file in the entity propagation configuration. We call the normal method to compute normal random values. Then we compute the fading.

In the example we choose Rice fading that is computed according to the standard IEEE 802.15.6

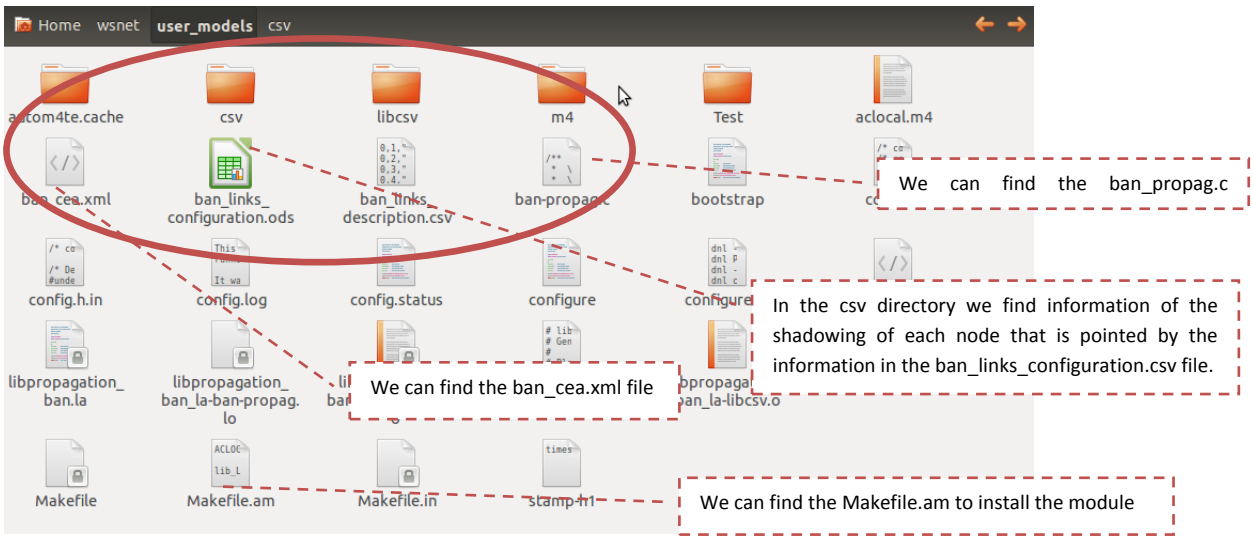
```
/* ************************************************** */
/* ************************************************** */
double propagation(call_t *c, packet_t *packet, nodeid_t src, nodeid_t dst, double rxdBm)
{
  ...
    fprintf(stdout,"[ban-propag] Time called : %f s (propagation).\n", sim_time);
  ...

  VERB(fprintf(stdout,"[ban-propag] P_tx (dBm) : %f.\n", rxdBm));
  VERB(fprintf(stdout,"[ban-propag] Pathloss (shadowing) (dB) : %f.\n", pathloss));
  VERB(fprintf(stdout,"[ban-propag] Fading (dB) : %f.\n", fading));
  VERB(fprintf(stdout,"[ban-propag] P_rx (dBm) : %f.\n", rx_dBm));
  return rx_dBm;
}


/* ************************************************** */
/* ************************************************** */
propagation_methods_t methods = {propagation};
```

At the end we find the propagation method called by init to compute the receiver power and fading using the compute_ban_fading method.

This is also the responsible of the results showed during the simulation.

We see that the propagation method is defined at the end to be used by the xml file.

Implementation: First, download the zip file (ban_propag) and take a look at the propagation ban model implementation: ban-propag.c and ban_cea.xml. Then, copy all the files (csv libraries included) within the ***$HOME/wsnet-module/user_modules/*** directory.



After this, we need to compile and install our new application module. For this in the user_models directory, we have to modify the content of Makefile.am. At the end, the file should contain the following entries:

```
ACLOCAL_AMFLAGS=-I m4
lib_LTLIBRARIES = libpropagation_ban.la
libpropagation_ban_la_CFLAGS = $(CFLAGS) $(GLIB_FLAGS) $(GSL_FLAGS) -Wall
libpropagation_ban_la_SOURCES = ban-propag.c libcsv/libcsv.c
libpropagation_ban_la_LDFLAGS = -module
```

Then, to compile ban-propag.c type the following commands at /user_modules/ directory:

```
$ cd ./wsnet/user_models/
/user_models$ ./bootstrap
/user_models$ ./configure
/user_models$ make
```

After that, we have to install the module:

```
/user_models$ make install
```

If you type ls you will notice that some files have been created

```
arturo@arturo-OptiPlex-790:~/wsnet$ cd user_models
arturo@arturo-OptiPlex-790:~/wsnet/user_models$ ls
aclocal.m4                  ban-propag.c   config.status  libcsv                              libtool       resultats_ban.txt
autom4te.cache              ban-propag.c~  configure      libpropagation_ban.la               m4            stamp-h1
ban_cea.xml                 bootstrap      configure.ac   libpropagation_ban_la-ban-propag.lo  Makefile      Test
ban_cea.xml~                config.h       csv            libpropagation_ban_la-ban-propag.o  Makefile.am
ban_links_configuration.ods config.h.in    epidemic.xml   libpropagation_ban_la-libcsv.lo     Makefile.am~
ban_links_description.csv   config.log     epidemic.xml~  libpropagation_ban_la-libcsv.o      Makefile.in
arturo@arturo-OptiPlex-790:~/wsnet/user_models$
```

Simulation: Finally, we can simulate our module by typing in the wsnet directory:

```
~/wsnet$ wsnet -c user_models/ban_cea.xml
```

At the end of the simulation you will find the Simulation stats:

```
ban-propag] Pathloss (shadowing) (dB) : 20.633161.
ban-propag] Fading (dB) : -2.396227.
ban-propag] P_rx (dBm) : -23.029388.
ban-propag] Time called : 49.000000 s (propagation).
ban-propag] Fading enveloppe r.v. : 0.683211.
ban-propag] Source : 8
ban-propag] Destination : 10
ban-propag] P_tx (dBm) : 0.000000.
ban-propag] Pathloss (shadowing) (dB) : 33.554896.
ban-propag] Fading (dB) : -3.308901.
ban-propag] P_rx (dBm) : -36.863797.
ban-propag] Time called : 49.000000 s (propagation).
ban-propag] Fading enveloppe r.v. : 0.778638.
ban-propag] Source : 8
ban-propag] Destination : 3
ban-propag] P_tx (dBm) : 0.000000.
ban-propag] Pathloss (shadowing) (dB) : 29.109701.
ban-propag] Fading (dB) : -2.173284.
ban-propag] P_rx (dBm) : -31.282985.

imulation stats:
 simulated time: 50000000000
 simulation time: 232014000
 speedup: 215.504237
 events in queue: 1
 events executed: 19539
ban-propag] Destroying the propagation framework!
ban-propag] Freeing tables (init_tables).
rturo@arturo-OptiPlex-790:~/wsnet$ cd user_models
```

You should find this message to show that the simulation has finished successfully.

To analyze the results it is possible to parse the simulation results into a text file:

```
~/wsnet$ wsnet -c user_models/ban_cea.xml > user_models/results_ban.txt
```