REPORT ON OMNET++ SIMULATOR

The report contains two parts : the first one talks about the omnet++ simulator itself and the second concerns the INET framework for wireless simulations.

Part 1 : OMNET++5.2.1

Introduction :

This introduction is tentirely taken from the omnet++ website.

OMNeT++ is an extensible, modular, component-based C++ simulation library and framework, primarily for building network simulators originally developed by András Varga**.** "Network" is meant in a broader sense that includes wired and wireless communication networks, on-chip networks, queueing networks, and so on. Domain-specific functionality such as support for sensor networks, wireless ad-hoc networks, Internet protocols, performance modeling, photonic networks, etc., is provided by model frameworks (INET is an example), developed as independent projects. OMNeT++ offers an Eclipse-based IDE, a graphical runtime environment, and a host of other tools. There are extensions for real-time simulation, network emulation, database integration, SystemC integration, and several other functions.

The use of it is very popular in the scientific community as well as in industrial settings, and building up a large user community.

OMNeT++ provides a component architecture for models. Components *(modules)* are programmed in C++, then assembled into larger components and models using a high-level language *(NED)*. Reusability of models comes for free. OMNeT++ has extensive GUI support, and due to its modular architecture, the simulation kernel (and models) can be embedded easily into your applications. These components are :

- simulation kernel library ;
- NED topology description langage ;
- OMNeT++ IDE based on the Eclipse platform ;
- GUI for simulation execution, links into simulation executable (Tkenv) ;
- command-line user interface for simulation execution (Cmdenv) ;
- utilities (makefile creation tool, etc.) ;
- documentation, sample simulations, etc.

How to use the omnet++ simulator for wired simulations ?

Omnet++ runs on Windows, Linux, MAC OS X and other Unix-like systems. This report concerns only Linux and the simulations have been done with Ubuntu16.04. The following will just show some examples of how to create a new project, run, debug, add log, visualize results, add statistics and analyze them.

i. Create a new project

After downloading and installing omnet++, we can either run some examples projects (by locating the project and selecting the omnetpp.ini file and then clicking « run ») or create new projects. The next section shows how to create a new project from scratch and the main files needed for that. Three main files are required in an omnet++ project : the Ned file which is used to define components and assemble them into a larger units like networks ; the C++ file which is used to implement the functionnality of the simple module defined in the NED and the omnetpp.ini file which tells the simulattion program which networkhe has to simulate (because the NED file can contain several networks).

◆ **The NED file :**

This file, as mentioned earlier, describes the topology (the structure of the simulation) of the network we would like to simulate. After the new project has been created, the NED file must follow. Omnet++ IDE's NED editor is used to edit NED files and this can be done in two ways :

- In the design mode : where the topology is edited graphically
- In the source mode : edited as text

Simple modules, compounds modules, channels, … are components that a NED file uses.

(a) Create a new project : *File* → *New* → *Omnet++ Project…* → Give the project name and click « *finish* ».

(b) Add a NED file : Right click on the project → *New* → *Network Description File (NED)* → Give the name of the NED file and Ok

```
simple Txc1
{
    gates:
        input in;
        output out;
}

//
// Two instances (tic and toc) of Txc1 connected both ways.
// Tic and toc will pass messages to one another.
//
network Tictoc1
{
    @display("bgb=383,242");
    submodules:
        tic: Txc1;
        toc: Txc1 {
            @display("p=173,134");
        }
    connections:
        tic.out --> {  delay = 100ms; } --> toc.in;
        tic.in <-- {  delay = 100ms; } <-- toc.out;
}
```

Figure 1 : a NED File

- **The C++ file**

As previously mentioned, all the modules defined in the NED file are implemented in the C++ file. To create a C++ file, just right-clich on the project and proceed as with the NED file but here, choose « *Source File* » instead of NED file nd follow the same instructions. Figure 2 shows the implementation of the NED file created above.

```
/*
 * Txc1.cc
 *
 *  Created on: Mar 10, 2018
 *      Author: zongo
 */

#include <string.h>
#include <omnetpp.h>

using namespace omnetpp;

/**
 * Derive the Txc1 class from cSimpleModule. In the Tictoc1 network,
 * both the `tic' and `toc' modules are Txc1 objects, created by OMNeT++
 * at the beginning of the simulation.
 */
class Txc1 : public cSimpleModule
{
  protected:
    // The following redefined virtual function holds the algorithm.
    virtual void initialize() override;
    virtual void handleMessage(cMessage *msg) override;
};

// The module class needs to be registered with OMNeT++
Define_Module(Txc1);

void Txc1::initialize()
{
    // Initialize is called at the beginning of the simulation.
    // To bootstrap the tic-toc-tic-toc process, one of the modules needs
    // to send the first message. Let this be `tic'.

    // Am I Tic or Toc?
    if (strcmp("tic", getName()) == 0) {
        // create and send first message on gate "out". "tictocMsg" is an
```

Figure 2 : the C++ file

The Simple module **Txc1** defined in the NED file is implemented here by the *public* class *cSimpleModule.*

- **The omnetpp.ini file**

It is used to tell the simulation program which network to simulate. It can be edited in the Inifile editor which has two modes as well : the form mode and the source mode.

To create an omnetpp.ini file, proceed as with the first two files but instead of choosing *NED* or *Source* file, choose *Initialiazation File (ini)* and then follow the instructions.

```
#[General]
#network = Tictoc1
[General]
    # nothing here

    [Config Tictoc1]
    network = Tictoc1
```

Figure 3 : omnetpp.ini file

Tictoc1 is the name of the network topology created in the NED file and the one we are going to simulate.

In order to run the simulation, two steps are required. The first is to launch the simulation and the second is to run it.

- Launch : select omnetpp.ini in the project we are working on and click run. The IDE will build the project automatically. In case the omnetpp.ini contains several network, the IDE will ask you which network we would like to launch

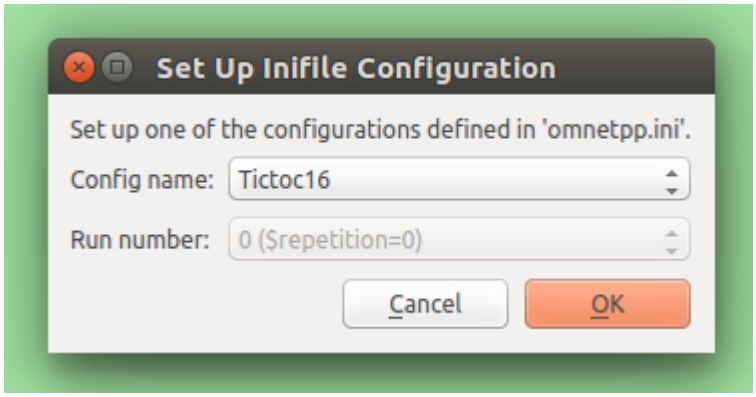

Figure 4 : choosing the simulation to be ran

- Run : once in the run GUI, click on run. We can play with the differents ways of running a simulation. Run fast, express, etc.

Figure 5 : A part of the toolbar of the execution window

ii. Debug

In order to track runtime errors, we need to debug the simulations. This is done by clicking the debug button and this activates the breakIntoDebuggerIfRequested() method.Like any C code. The important thing to know here is the crash case caused by a segfault seen where a variable is not initialized. The exit code is 139.

iii. Add log statements

The logging are done to see what is happening during our simulation and it is implemented in the C++ file. The one that we use here is the simplest one : « EV ». Syntax : EV << " Sending message 1 \n''

iv. Some parameters definition

Many parameters could be defined in order to refine the simulation. These parameters could be Connections (how modules are interconnected) channels (they are used to specify the delay parameter of connections in the network), message definition (used to create message files) and many more.

v. Adding statistics and visualizing results

It is possible to get an overview at runtime of how many messages each node sent or received by adding two counters the module class **numSent** and **numReceived**. In the same way, it is possible to add other statistics such as the hop count a message has passed before it reaches his destination. For this purpose, we need to record in the hop count of every message upon arrival into an output vector (a sequence of (time,value) pairs, sort of a time series). It is also possible to calculate mean, standard deviation, minimum, maximum values per node, and write them into a file at the end of the simulation. Then we'll use tools from the OMNeT++ IDE to analyse the output files.

For that, we add an output vector object (which will record datas into a file with the #0.vec suffix) and an histogram object (which also calculates mean, ...) to the class in the C++ file like this :

```
class ClassName : public cSimpleModule {
                Private :
                     long numSent ;
                     long numReceived ;
                     cLongHistogram hopCountStats ;
                     cOutVector hopCountVector ;
                Protected :
…
}
```

When a message arrives at the destination node, the statistics are update because of the following code addes to handleMessage() method :

```
hopCountVector.record(hopcount) ;
hopCountStats.collect(hopcount) ;
```

The `hopCountVector.record()` call writes the data into `*#0.vec`. Scalar data (collected by the histogram object in this simulation) have to be recorded manually, in the `finish()` function like in the figure

below. `finish()` is invoked on successful completion of the simulation, i.e. not when it's stopped with an error. The `recordScalar()` calls in the code below write into the `*-#0.sca` file.

```
void Txc15::finish()
{
    // This function is called by OMNeT++ at the end of the simulation.
    EV << "Sent:     " << numSent << endl;
    EV << "Received: " << numReceived << endl;
    EV << "Hop count, min:    " << hopCountStats.getMin() << endl;
    EV << "Hop count, max:    " << hopCountStats.getMax() << endl;
    EV << "Hop count, mean:   " << hopCountStats.getMean() << endl;
    EV << "Hop count, stddev: " << hopCountStats.getStddev() << endl;

    recordScalar("#sent", numSent);
    recordScalar("#received", numReceived);

    hopCountStats.recordAs("hop count");
}
```

In order to view the data during the simulation, we have to choose the specific node u'd like to view datas and right-click on that node and choose « open details for nodeName ». This action opens a new window where we have to locate what kind of diagram we want. Either a vector (cOutVector) or an histogram (cLongHistogram). Once chosen, right-click on it and click « open graphical view » and we will wait until something starts to appear on the window. But by running the simulation normally we will have to wait for long. So, it is wise to click on « Fast » or « Express » in the toolbar instead of « run » in order to see something.

There are two methods to have the statistics (by recording values and events) : one by modifying the c++ (the previous method) file and another by not modifying the c++ file but by manipulating signals. The late one is better because usually, people don't know yet at the time of writing the model what data the end user will need.

To use this method, three steps are required :

(a) Define the signal with an identifier

(b) Register the signal before any usage

(c) Emit the signal where ever the user would like.

To visualize a global simulation, go to the results directory of the project and double-click on the filename.anf file. Then play with the toolbar, specially the Browse data tool.

Below are the two types of ploting the data recorded : a vector and an histogram.

The difference between the two is that a scalar result (histogram) has only one output value of simulation (eg : the number of received packets) while a vector result saves many pairs of time-value within a specific period (eg : response times durant the simulation).

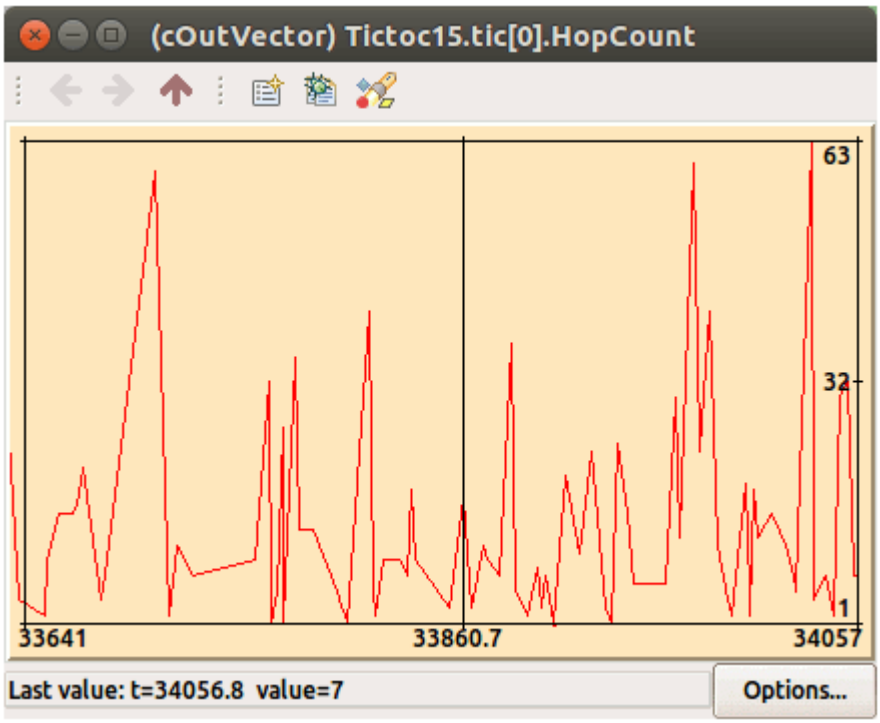

Figure 6 : an output vector example

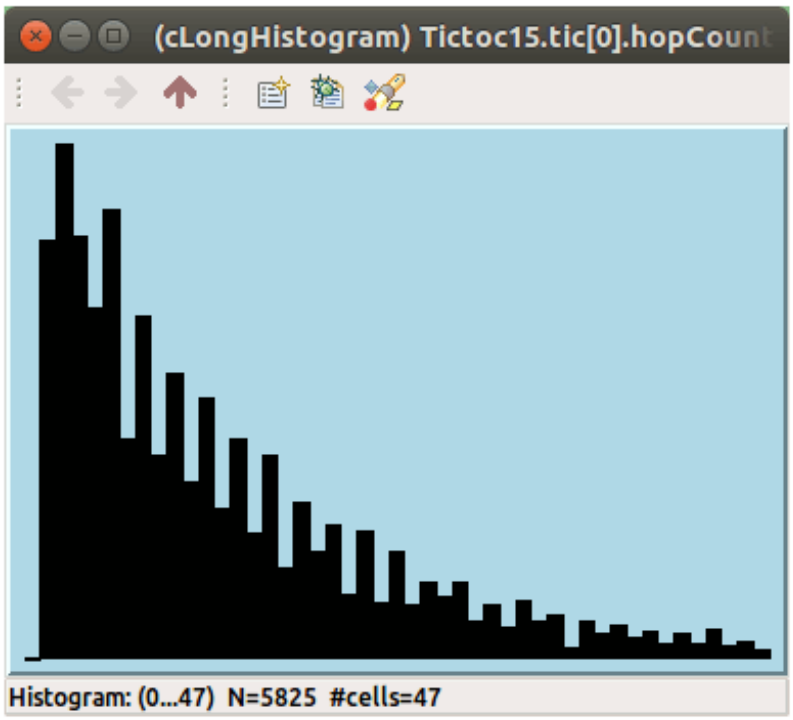

Figure 7 : an scalar ouput example

Part 2 : The INET framework

Introduction

INET Framework is an open-source model library for the OMNeT++ simulation environment. It provides protocols, agents and other models for researchers and students working with communication networks. INET is especially useful when designing and validating new protocols, or exploring new scenarios.

INET contains models for the Internet stack (TCP, UDP, IPv4, IPv6, OSPF, BGP, etc.), wired and wireless link layer protocols (Ethernet, PPP, IEEE 802.11, etc), support for mobility, MANET protocols, DiffServ, MPLS with LDP and RSVP-TE signalling, several application models, and many other protocols and components.

INET is built around the concept of modules that communicate by message passing. Agents and network protocols are represented by components, which can be freely combined to form hosts, routers, switches, and other networking devices. New components can be programmed by the user, and existing components have been written so that they are easy to understand and modify.

INET benefits from the infrastructure provided by OMNeT++. Beyond making use of the services provided by the OMNeT++ simulation kernel and library (component model,

parameterization, result recording, etc.), this also means that models may be developed, assembled, parameterized, run, and their results evaluted from the comfort of the OMNeT++ Simulation IDE, or from the command line. The great advantage of INET is that it is maintained by the OMNeT++ team for the community, utilizing patches and new models contributed by members of the community.

Some features:

- OSI layers implemented (physical, link-layer, network, transport, application)
- Pluggable protocol implementations for various layers
- IPv4/IPv6 network stack (or build your own network layer)
- Transport layer protocols: TCP, UDP, SCTP
- Routing protocols (ad-hoc and wired)
- Wired/wireless interfaces (Ethernet, PPP, IEEE 802.11, etc.)
- Physical layer with scalable level of detail (unit disc radio to detailed propagation models, frame level to bit/symbol level representation, etc.)
- Wide range of application models
- Network emulation support
- Mobility support
- Supports the modeling of the physical environment (obstacles for radio propagation, etc.)
- Separation of concerns
- Visualization support

OMNet++ Architecture model for IEEE 802.11 in the INET framework

The INET framework provides various modules that can be used for developing simulation models. In the inetconfig, the ROOT parameter is adjusted to the directory of the framework before one can build it usng the ./makemake and make command.

As for the OMNet++ IDE, simulations with INET can be done using the OMNeT++ IDE. Before been able to use INET, one must install it. There are two types of installations :

- **Manual installation :**

  (a) download the latest INET sources corresponding to your OMNeT++ version. For our case, it is ***INET 3.6.4*** for ***OMNeT++ 5.2.1*** which can be found here : https://github-production-release-asset-2e65be.s3.amazonaws.com/112633/5ecc1ce6-3279-11e8-81ff-

de6c42d6cb2a?X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-Credential=AKIAIWNJYAX4CSVEH53A%2F20180502%2Fus-east-1%2Fs3%2Faws4_request&X-Amz-Date=20180502T090308Z&X-Amz-Expires=300&X-Amz-Signature=956483543375fdee9d88f7b850749c802b1a5a795a821d660fe8e1919b2d187e&X-Amz-SignedHeaders=host&actor_id=0&response-content-disposition=attachment%3B%20filename%3Dinet-3.6.4-src.tgz&response-content-type=application%2Foctet-stream

(b) Unpack it into the directory of your choice: `tar xvfz inet-<version>.tgz`

(c) Start the OMNeT++ IDE, and import the project via `File -> Import -> Existing Projects` to the Workspace. A project named INET should appear. But if it doesn't apear (as it was for our case), locate it in your computer and then click on it

(d) Build with `Project -> Build`, or hit Ctrl+B from your IDE GUI

(e) And then launch some example simulations located in `inet→ examples` subdirectory by locating the `omnetpp.ini` file and clicking `run`.

In order to create a new project, the procedure is quite the same as for an OMNeT++ project by with a slight difference. Below is the complete procedure to create a new inet project :

i. File -> New -> OMNeT++ Project

ii. Create a name and keep the default location, select Empty Project and click Finish

iii. Link inet project to your new project. Right-click on your project, then click Properties. This will open the properties window. Select 'Project References' and select 'inet' from the list

- **Automatic installation :**

(a) Open the OMNeT++ IDE (omnetpp)

(b) Go to the workbench (dismiss the Welcome screen). The first time you do this, a prompt will ask if you want to install INET

(c) Keep the boxes checked and proceed

(d) In the OMNeT++ IDE GUI, Go to Help -> Install Simulation Models

(e) A dialog will appear with the available simulation models. Currently only INET is listed there, simply select it and follow the prompts

(f) Both ways, the IDE will download, unzip, and automatically build INET from the sources

Contrarily to OMNet++ projects, to create a simulation with all the needed modules already in the existing packages, just 2 files are needed :

1. A topology file (.ned), which defines package for your project. A NED file contains the network, i.e. routers, hosts and other network devices connected together. You can use a text editor or the IDE's graphical editor to create the network.
2. A configuration file (.ini). Modules in the network contain a lot of unassigned parameters, which need to be assigned before the simulation can be run. The name of the network to be simulated, parameter values and other configuration option need to be specified in the omnetpp.ini file

Modeling layers for an IEEE 802.11 simulation Model

I. Physical layer :

All wireless simulations in INET need a radio medium module. The module called “radioMedium “ is used to represent the shared physical medium where communication takes place. It is responsible for taking signal propagation, attenuation, interference and other phenomena into account. INET can model the wireless physical layer at various levels of detail, realized with different radio medium modules : “ IdealRadioMedium“ (the simplest one which ignores some phenomena and it os often used for modeling ad hoc routing protocols).

NB : In hosts, Network Interface Cards (NICs) are represented by the NIC module.

The NIC module description.

NICs module consists of four layers : Agent, management, MAC and physical (Radio). As our work is based on IEEE 802.11 ad hoc networks, the NIC we studied is IEEE 802.11 interface which is Ieee80211NicAdhoc. The descriptions belows are based on it.

- The Physical layer :Ieee80211Radio models transmissions and receptions of frames ; characterizes the model of the radio channel and determines if a frames was received correctly and only if it is the case, the frame is passed up to the Mac layer.
- The Mac layer : Ieee802Mac performs transmissions according to the CSMA/CA protocol. It receives datas and management frames from the upper layers and transmits them.
- The Management layer : It performs encapsulation and decapsulation of data packets for the Mac , and exchanges management frames via the Mac with its peers management entities in other STAs and Aps. Beacons, Probe Request/Response etc. frames are generated and interpreted by management entities and transmitted/received via the Mac layer that

periodically switches channels and collects information from received beacons and probe responsaes. It has several implementations which differ from their role (STA/AP/Ad hoc) and level of detail : Ieee80211MgmtAdhoc, Ieee80211MgmtAP, Ieee80211MgmtSTA, Ieee80211APSimplified, Ieee80211STASimplified.

- The agent is what instructs the management layer to perform scanning, authentication and association. Tha management layer itself just carries out these commands by performing the scanning, authentication and association procedure and reports back the results to the agent. The agent module is only present in the Ieee80211NicSTANic module. The management entity in other NIC variants do not have much freedom as to need an agent to control them.

The diagram below is the representation of the internal architecture of the Ieee80211NicAdhoc

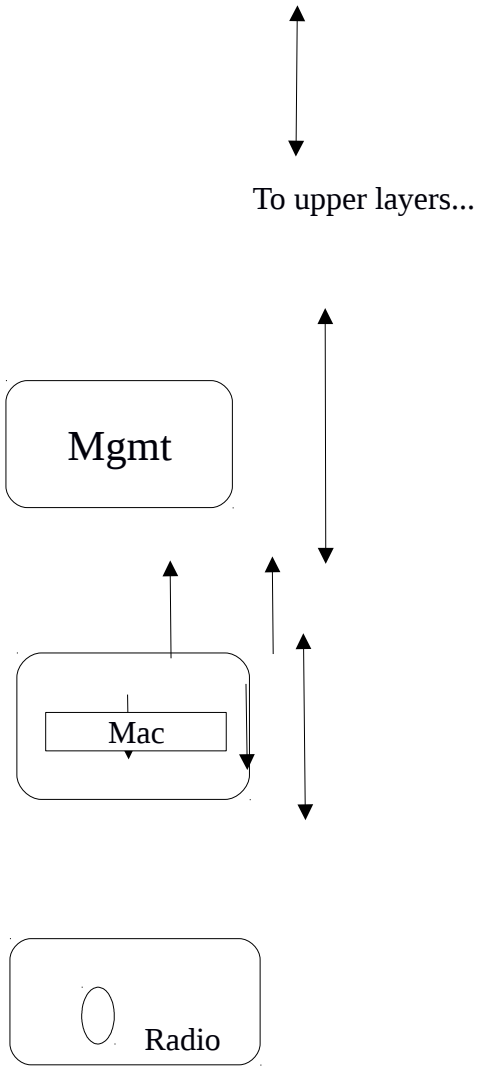

Figure : Internal architecture of Ieee80211NicAdhoc

II. MAC layer

As already stated, NICs modules contain also an L2 protocol and the Mac protocol is also configurable.

As my work is mainly based on the Mac sublayer of the 802.11 standard, the following describes the internal architecture of the Mac « module » present in the inet framework. The biggest part of it has been taken from the *Ieee80211Doc.ned* source file of inet.

The Ieee80211 interface card (NIC) is the module that contains the Mac sublayer. In fact, as we have already seen, the Ieee80211Nic is a compound module that contains four layers : the Physical layer, the Mac layer, the Management layer and the Agent (from bottom to the top). Our interest here is in the Mac sublayer of the real IEEE 802.11 implemented in the inet framework through the Ieee80211Nic.

Nic is has several flavours differing in their roles. We have one for ad hoc network and another for infrastructure networks. Namely :

- *Ieee80211Nic* : a generic (configurable NIC)
- *Ieee80211Nic* with mgmtType :
  - *Ieee80211Mgmt* : it is a module interface, not a concrete module type but a prototype for all IEEE 802.11 management module types. It helps to specify what gate a management module should have in order to be usable within Ieee80211Nic.
  - For ad hoc mode : *Ieee80211MgmtAdhoc* : it is a simple module and completely relies on the Mac layer for transmission and reception of frames.
  - For infrastructure mode : *Ieee80211MgmtAP* or *Ieee80211MgmtAPSimplified* for use in access Point and *Ieee80211MgmtSTA* or *Ieee80211MgmtSTASimplified* for use in an infrastructure-mode station.

Actually, the module of interest for our work in the Management module as it has been said early in this report, this module is the one in charge of switching channel and the Mac module is just in charge of transmitting/receiving frames according to the CSMA/CA protocol.

In order to design a simulation model within the inet framework, we need :

(a) **Modules** : two types of modules exist.

- Simple modules : they are implemented in C++ and represent active components of omnet++ where events take placeand behaviours of the model are definied. In the inet context, simple modules represent a protocol, an application. A simple module's external

interface (gates [connectors] and parameters) is described in a NED file, and the implementation is contained in a C++ class with the same name.

- Compound modules : they are made up of simple modules or even compound modules and are used as containers to structure a model. In the inet context, they contain hosts, routers and other modules which form the network.

Note : A network is a compound module which is made up of assembled simple or compound modules. Modules are organized into and packages are organized according to the OSI layers.

Common modules are :

i. Hosts : such as
   - *WirelessHost*: represents a host in wireless network within infrastructure mode
   - *AdhocHost* : represents a host in a MANET with a routing table included
   - *AccessPoint* : represents an access point.
   - Etc.

ii. Protocols and other devices : such as
   - *inet.linklayer.ieee80211.Ieee80211Nic*: implementation of a wireless NIC
   - `inet.linklayer.ieee80211.Ieee80211.mgmt.*`: modules that configure the wireless interface to work as an AP or a STA or an ad hoc host.

(b) **Parameters :** they are module variables which are used to configure submodules. They can be strings, numerics, boolean values, or contain XML data trees.

(c) Connections : they define a connection between two gates or submodules. For example, two wireless stations are connected through a NIC. For example :

```
radio.upperLayerIn <-- mac.lowerLayerOut;
radio.upperLayerOut --> mac.lowerLayerIn;

mac.upperLayerOut --> mgmt.macIn;
mac.upperLayerIn <-- mgmt.macOut;
```

(d) **Gates :** they are the connection points of a module. For example :

```
input upperLayerIn;  // to upper layers
output upperLayerOut; // from upper layers
```

After building a model, we will need to evaluate the model by interpreting experimental results. For this end, it is suitable to compare multiple results from more than one set of simulation outputwith different parameters. The output of the simulation is written into results files (just like with omnet++ projects) : output vector file, output scalar files and if presents, the user's own output file. Output files are line-oriented text files which can be processed by a variety of tools such as Matlab, R, Perl, Python and spreadsheets.