WSNet : How to write a new application module ?

Tutorial

Elyes Ben Hamida, ARES INRIA / CITI - INSA Lyon
Guillaume Chelius, ARES INRIA / CITI - INSA Lyon

WSNet : How to write a new application module ?: Tutorial

by Elyes Ben Hamida and Guillaume Chelius

Abstract

In this tutorial, we will learn the basic steps necessary for writing a new application module for WSNet.

Table of Contents

Chapter 1. Introduction

In this tutorial we will learn the basics of writing a new application module for WSNet. First, we present the requirements for building a new module. Second, we take as an example an epidemic protocol and we implement it as an external module. Finally, we install the new module and we run some simulations.

Chapter 2. Requirements

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

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

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

```
$ ./bootstrap && ./configure && make
```

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

```
# sudo make install
```

In case of compilation errors please read the WSNet installation tutorial [http://wsnet.gforge.inria.fr/installation.html] for a more complete guide.

Setting up the working directory

Once WSNet is correctly installed, one can start the implementation of a new module. This module can be implemented within the WSNet source code archive. For example, if we implement an application protocol, we can add the source code of our module to the **wsnet/model/application/** directory. This method requires the modification of the WSNet distribution to install our new module.

In this tutorial, we consider a more flexible method for implementing a new module. This method consists in working in the **$HOME** directory. An environment variable is then configured to tell the simulator where our modules can be found. For the rest of this tutorial, we will take **$HOME/wsnet-module/** as our home directory for writing new WSNet modules.

First, create and move to the working directory by typing the following commands:

```
$ mkdir $HOME/wsnet-module/
$ cd  $HOME/wsnet-module/
```

Second, download the files needed for the implementation from the SVN repository by typing the following command:

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

This will download the files needed for writing new WSNet modules. Now, the **$HOME/wsnet-module/user_models/** directory should contain the following files:

```
$ ls ./user_models/
bootstrap  configure.ac  ltmain.sh  Makefile.am
```

Finally, we have to setup the **WSNET_MODDIR** environment variable by adding the following lines in $HOME/.bashrc file:

```
export WSNET_MODDIR=$HOME/wsnet-module/lib
```

This variable tells the WSNet simulator where our modules can be found. In our case, the modules will be installed within the **`$HOME/wsnet-module/lib`** directory. In next section we present a case study of the implementation of an application module.

Chapter 3. Implementation

Case study : an epidemic protocol

In this section, we implement an epidemic data dissemination protocol. This protocol operates as follows:

- A source node broadcast periodically a new data.
- A sensor node which receives a new data (not received previously), broadcasts it again according to a given probability **p**, and stores it locally according to a given probability **q**.

This protocol can be seen as a probabilistic flooding algorithm. Each data will be defined by a sequence number and a source number. Two types of nodes have to defined : the source and the sensor.

Before starting the implementation, we present in the next section the architecture of a WSNet application module.

Application module architecture

Each WSNet application module have to follows the following source code architecture:

Note

```
#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);
int rx(call_t *c, packet_t *packet);
application_methods_t methods = {rx};
```

where:

- **`model_t model:`** is a data struture describing the module (authors, version, ...)
- **`init:`** is a function used 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:`** is a function used to bind the application entity to a node and read the node parameters from the xml config file
- **`unsetnode:`** is the function called at the node's death to unbind the application entity from the node
- **`bootstrap:`** is the function called automatically at the node birth
- **`rx:`** is the function called when the application receives a data packet
- **`application_methods_t methods:`** is a data structure describing the exported functions for low layers

The global entity parameters will be the same for all nodes created from this entity. In our case, the probabilities **`p`** and **`q`** will be the same for all nodes. However, the type of the node (i.e., source or sensor) will be defined for each node. For the first case, we use the **`init`** function to read global parameters from the xml config file (i.e., we read the content of the **`init`** markup). For the second case, we use the **`setnode`** function to read node parameters from the xml config file (i.e., we read the content of the **`default`** markup).

Implementation

Please download and take a look at the epidemic protocol implementation's: epidemic.c [http://wsnet.gforge.inria.fr/tutorials/dev-app/epidemic.c]. Then, copy **`epidemic.c`** within the **`$HOME/wsnet-module/user_modules/`** directory.

In the next section, we show you how to compile and install our new application module.

Chapter 4. Compiling and installing the new module

Compiling the epidemic protocol

Copy epidemic.c [http://wsnet.gforge.inria.fr/tutorials/dev-app/epidemic.c] into **$HOME/wsnet-module/user_modules/**, then move to this directory.

Next, modify the content of **Makefile.am**. This file should contain the following entries (download Makefile.am [http://wsnet.gforge.inria.fr/tutorials/dev-app/Makefile.am]):

```
lib_LTLIBRARIES = libapplication_epidemic.la
libapplication_epidemic_la_CFLAGS = $(CFLAGS) $(GLIB_FLAGS) -Wall
libapplication_epidemic_la_SOURCES = epidemic.c
libapplication_epidemic_la_LDFLAGS = -module
```

To compile epidemic.c type the following commands:

```
$ ./bootstrap
$ ./configure --prefix=$HOME/wsnet-module --with-wsnet-dir=/usr/local/wsnet-2.0
$ make
```

The **--prefix** option represent the directory where modules will be installed, and the **--with-wsnet-dir** is the WSNet install directory.

Installing the epidemic protocol

To install the compiled module, type the following command:

```
$ make install
```

This command will install the module within the directory defined by the configure **--prefix** option. In our case, the module is installed within the **$HOME/wsnet-module/lib/** directory:

```
$ ls $HOME/wsnet-module/lib/
libapplication_epidemic.a  libapplication_epidemic.la  libapplication_epidemic.so
libapplication_epidemic.so.0.0.0
```

Finally, to be able to use our module we have to define the **WSNET_MODDIR** environment variable by adding the following lines in $HOME/.bashrc file:

```
export WSNET_MODDIR=$HOME/wsnet-module/lib
```

Chapter 5. Running the simulation

Setting up the config file

To run a simulation we have to setup an xml config file. Take a deeper look at this sample config file: epidemic.xml [http://wsnet.gforge.inria.fr/tutorials/dev-app/epidemic.xml]. In this config, we assume that 100 nodes are distributed randomly over 100x100 area. We use a 802.11 MAC protocol with RTS/CTS, an omnidirectionnal antenna, orthogonal interferences, and circular propagation range.

We define two entities for the nodes: epidemic-sensor and epidemic-source. We assign for the source a static position (50.0, 50.0) and sensors are deployed randomly. By default, nodes act as sensors, and we assign the node 0 the role of source.

Running the simulation

Download and copy epidemic.xml [http://wsnet.gforge.inria.fr/tutorials/dev-app/epidemic.xml] to **`$HOME/wsnet-module/`**. To run the simulation, type the following command:

```
$ wsnet -c epidemic.xml
```

Chapter 6. Credits

Credit for the icons goes to the Tango Desktop Project [http://tango.freedesktop.org/].

The style is inspired from Shaun's Blog [http://blogs.gnome.org/shaunm/].