LAB PROJECT Nº0

OMNET++/INET TUTORIAL

1 INTRODUCTION

This guide is intended to be followed by the students in order to give their first steps in OMNET++/INET network simulation. It assumes the availability of an OMNET++ installation where the INET Framework is included. Follow the described steps and ask the professor or lab assistant if you have any questions.

OMNET++ uses three main kinds of files:

- Source files: C++ files (.h, .cc) and message definition files (.msg).
- Module definition files: Scrips written in NED language (.ned).
- Initialization files: Simulation definition files that parameterize the modules defined with the NED language (.ini).

In this course, we are mostly concerned with the module and **simulation definition files** (i.e., .ned and .ini files), although it will be sometimes useful to look at the **source files** (i.e., .h and .cpp, as well as .ned files corresponding to these modules) in order to better understand the module functionality.

OMNET++ is a generic discrete event simulator, suitable to model any system consisting of modules that communicate by means of messages. However, in this course, we will focus on communication network simulation. The INET framework is an OMNET++ add-on that provides a comprehensive implementation of well-known communication technologies and protocols: e.g., IEEE 802.11, the IP protocol stack, as well as mobile ad-hoc network (MANET) routing protocols.

Note: This project is not intended for evaluation, neither does it require the students to deliver a report. However, the more dedication the students pay understanding the tasks in this lab, the less time they will spend with the other projects, namely LAB1.

2 A SMALL IEEE 802.11 INFRASTRUCTURE WLAN

This task is based on the *hosttohost_LAB0* scenario, which models the communication between one or more client hosts and a server host through an IEEE 802.11 access point. In order to explore this scenario more efficiently, the students should follow the following steps.

1. Download the *hosttohost_LAB0.zip* file from the course's webpage and expand it below *inet/examples/wireless/*.

2. Within the OMNET++ IDE, expand the *inet/examples/wireless/hosttohost_LAB0* folder. You will be able to see the files *SimpleWLAN.ned* and *omnetpp.ini*. *SimpleWLAN.ned* defines the node types and network configuration for this scenario, while *omnetpp.ini* assigns specific values to node modules and network parameters defined in *SimpleWLAN.ned*.

3. Double click on *SimpleWLAN.ned* to open it. You will be able to visualize the contents of the *.ned* file graphically. Starting at the top, you can see the modules that compose the *ClientNode* node type, followed by the *ServerNode* node type. Both comprise a WLAN card, interface table and a mobility module, responsible for calculating the node position as a function of time. The remaining module is the application, which is the element that differs between the two node types. The *ClientNode* runs a client application *cli* (of type *EtherAppClient*), while the *ServerNode* runs a server application *srv* (of type *EtherAppServer*). As you may notice, in this scenario the applications run directly on top of the WLAN card (physical and data link layers) without an IP protocol stack. Below the *ServerNode*, you can see the network definition, which was called *SimpleWLAN*. The network comprises a *srvHost* (a *ServerNode*), one *ap* (an *AccessPoint*), the *radioMedium* (this is the entity responsible for calculating radio propagation effects) and a vector of several *cliHost* nodes (these are *ClientNode* instances). The number of *cliHost* nodes present in the scenario is given by *numCli*, which is a parameter of the *SimpleWLAN* network module.

4. The way the module hierarchy is defined can be simply grasped if you switch to the "Source" view of the *.ned* file. Below the initial package and import definitions, you can see that the modules that you could see in the graphical view are correlated with the textual contents of the *.ned* file. You can see that there is an implicit module hierarchy, since the defined modules have sub-modules, which already appeared in the graphical view. The modules and sub-modules have parameters. As you will see, the values of these parameters (e.g., *numCli*) can be assigned or modified from the *.ini* file. Some of these parameters are special and are called "properties", e.g., ***@display***

and ***@networkNode***. The property ***@display*** controls the graphical representation of the module, namely designating the graphical icon and/or its position. The property ***@networkNode*** simply indicates that the module is a network node. The modules also have *gates* through which can connect with other modules. The connections between a module's *gates* and those of its sub-modules, as well as between submodules, are defined in the section *connections* in the module definition. Although the *SimpleWLAN* network includes a sub-module *ap* of type *AccessPoint*, this module type is not defined in *SimpleWLAN.ned*. This is because *AccessPoint* already makes part of the INET implementation. In fact, you can access its *.ned* file at *inet/src/inet/nodes/wireless/AccessPoint.ned*. You can also look at the application types being run by the *ClientNode* and *ServerNode*, which are *EtherAppCli* and *EtherAppServer*, respectively. The *EtherAppCli* simply sends request frames to a destination MAC address, with a constant period, while the *EtherAppServer* application receives the request frames and responds with the requested number of bytes. Another interesting feature is the *mobility* module types. *ClientNode* nodes have *CircleMobility* (nodes are deployed and move in a circle) while the ServerNode is meant to remain static (*StationaryMobility*)

5. Go back to the graphical representation of *SimpleWLAN.ned*. You can double-click on any sub-module in order to open it. In case a sub-module is itself a compound module with sub-modules, when you open it, it will also display a hierarchical graphical view. For example, the *wlan* sub-modules, which are of type *IEEE80211Nic* include the *mac* and *radio* sub-modules, which implement the physical and MAC protocol layers, respectively. Other modules have no sub-modules, being directly implemented in C++ (e.g., *sink*). These are called simple modules and are defined in a *.ned* file using explicitly the keyword *simple*.

6. Open the *omnetpp.ini* file, which contains the parameter definition. The language employed in *.ini* files makes extensive use of wildcards for sake of abbreviation. For example, the *constraintAreaMinX* parameter belongs to the mobility module, although that is not explicit in its assignment. This wildcard syntax is described in the OMNET++ User Manual. If you read the *.ini* file, you will find interesting parameters, such as radio and MAC parameters, as well as application parameters, namely the destination address of the *EtherAppCli* application run by the *cliHost* nodes, which should correspond to the srvHost's MAC address.

7. The *omnetpp.ini* file describes two configurations, or scenario variants. The *General* configuration forms the basis of the simulation and defines all parameters except *SimpleWLAN.numCli*. The *SimpleWLAN1* configuration fixes the latter with a value of 6, defining a scenario with 6 *cliHost* nodes.

8. While having the *omnetpp.ini* file open, run the simulation, selecting option Run/Run from the main menu. After some processing, a dialog box with appear, asking you

which configuration to run. Select the *General* configuration. Another dialog box will ask you the value of the *numCli* parameter. Whenever a parameter is not defined in either the *.ned* or *.ini* files, the user is prompted to manually enter its value at the beginning of the simulation. Enter the value 6. You will then see that, as expected, the *cliHost* nodes are deployed in a circle around the access point, while *srvHost* is statically deployed at the lower right corner. This setting is derived from the mobility parameters and/or display properties defined in the *.ned* and *.ini* files. You can now run the simulation and watch the message flow. In the simulator's main window (Tkenv) you can watch the scheduled events as well as the trace strings printed from the C++ objects as the simulation runs. You have also the possibility to change the simulation speed, or interrupt/resume the simulation, etc.

9. Run the simulation for 5 seconds, then stop it and select "Simulate/Conclude Simulation" from the TKenv menu and/or terminate the Tkenv. You will now notice that the simulation has generated some statistic files at *inet/examples/wireless/hosttohost_LAB0/results*: *General-0.sca*, *General-0.vec* and *General-0.vci*. The *.sca* file contains scalar statistic output variables, while the *.vec* file contains vector trace variables. The *.vci* file is just an index to the *.vec* file.

10. Open the *.sca* or the *.vec* file in order to access the output statistics variables. An *.anf* file will be created integrating all the statistics. You can now browse the statistics variables, which are organized according to the module hierarchy.

11. Find the following statistics: average end-to-end delay and total number of octets successfully received, both at the application layer. **Challenge: calculate the throughput in bit-per-second, based on the statistics variables that you find appropriate.**

12. At the MAC layer of *cliHost* nodes, check the average number of packets sent/received to/from the higher layer, and number of packets sent/received to/from the lower layer.

13. Do the same at the access point. Check how many packets were incorrectly received. Explain.

14. Check the average signal-to-noise-plus-interference ratio at *srvHost*'s radio.

3 AN IEEE 802.11 INFRASTRUCTURE WLAN WITH IP STACK

This task is based on the *inet/examples/wireless/wiredandwirelesshostswithap* scenario.

1. Copy the *inet/examples/wireless/wiredandwirelesshostswithap* scenario to *inet/examples/wireless/wiredandwirelesshostswithap_LAB0*.

2. Perform the necessary changes to the script files, so that the IDE does not complain about errors in the new scenario folder.

3. Within the OMNET++ IDE, expand the *inet/examples/wireless/wiredandwirelesshostswithap_LAB0* folder. Open the *WiredAndWirelessHostsWithAP.ned* file.

4. Check the types and contents of *wirelessHost1*, *wiredHost1* and *wiredHost2*. The protocol layers are conveniently separated. What is the relationship between *WirelessHost* and *StandardHost*? What are the differences?

5. Run the scenario. Open the wireless host by clicking on its icon. As you can see, the layers of the protocol stack are conveniently identified. What are the supported network layer and transport layer protocols? What kinds of applications are supported?

6. Change the scenario so that the number of wireless nodes is variable and is deployed in a circle around the AP, at a distance of 100 m, moving around the circle at a speed of 5 m/s. **Tips: (1) you had already seen such a behavior in the hosttohost_LAB0 scenario; (2) In *WirelessHost*, the *mobility* module is abstract and its type is defined in parameter **.wirelessHost1.mobility.typename.**

7. Set the WLAN bitrate to 5.5 Mbit/s and a transmit power of 18 dBm.

8. Simulate the scenario during 5 seconds of simulation time.

9. Check the average signal-to-noise-plus-interference ratio at the *ap*.

10. Finish the simulation. Increase the transmit power to 22 dBm.

11. Simulate the scenario during 5 seconds of simulation time.

12. Check the average signal-to-noise-plus-interference ratio at the *ap*.

13. Finish the simulation. Change the mobility model to Random Waypoint mobility, setting the speed at 2 m/s and wait time of 1 s. **Note: this mobility model requires that all minimum and maximum X, Y and Z constraints be initialized, otherwise it crashes.**

14. Simulate the scenario again during 5s and compare the results.