

OMNET++ Tutorial/Homework 2

Problem: The new problem that we are trying to solve involves a small network with a generator and a sink module. The sink module acts as a destination node. The connection between the nodes is prone to errors, and it is subject to random transmission delays. Since the packets are delayed and some of the packets may thus be lost, the sink will collect statistics on the received packets.

The goal for this tutorial is to teach you how to assign various properties to connections and also how to collect statistics.

Design steps:

Step 1. Create a working directory called my_example3 in the C:\OMNET++\samples directory, and cd to this directory.

Step 2. Copy all your previous files that have related to my_net1 example.

We will keep the Generator1 module (object) unchanged, and we will modify the connection properties, as well as the sink module.

We start with the network description. We rename the file my_net3.ned.

The file my_net3.ned should contain:

```
//
// This file is the network description for my first OMNET++ example
//
//

import "gen1"; // imports all the code from gen1.ned file
                // alternatively, for simple examples, you can put the
                // code in gen.ned at the top of the net1.ned file
import "sink3"; // imports all the code from sink1.ned file

module Network3 //declare network object Network1
  submodules:
    first: Generator; //one instance of the Generator module
                    //object;
                    // The Generator module object is declared
                    // in gen1.ned as a simple module object

    parameters:
```

```

        number_rand = input; //the number of messages and the
                               // probability will be specified
                               // at runtime (or from the config
                               // file)

    sink: Sink3;
        connections:
            first.out --> error 1e-2 delay exponential(5) --> sink.in;
endmodule

network my_net3 : Network3
endnetwork

```

In the previous code, we have added properties for the connection between Generator and sink : a bit error rate probability of 10-2m, and a transmission delay that is exponentially distributed with mean = 5 seconds.

The gen1.ned file remains the same as in the previous example:

```

simple Generator      //declaration of simple module object
    parameters: number_rand; //declare parameters for the module
    gates:
        out: out;
endsimple

```

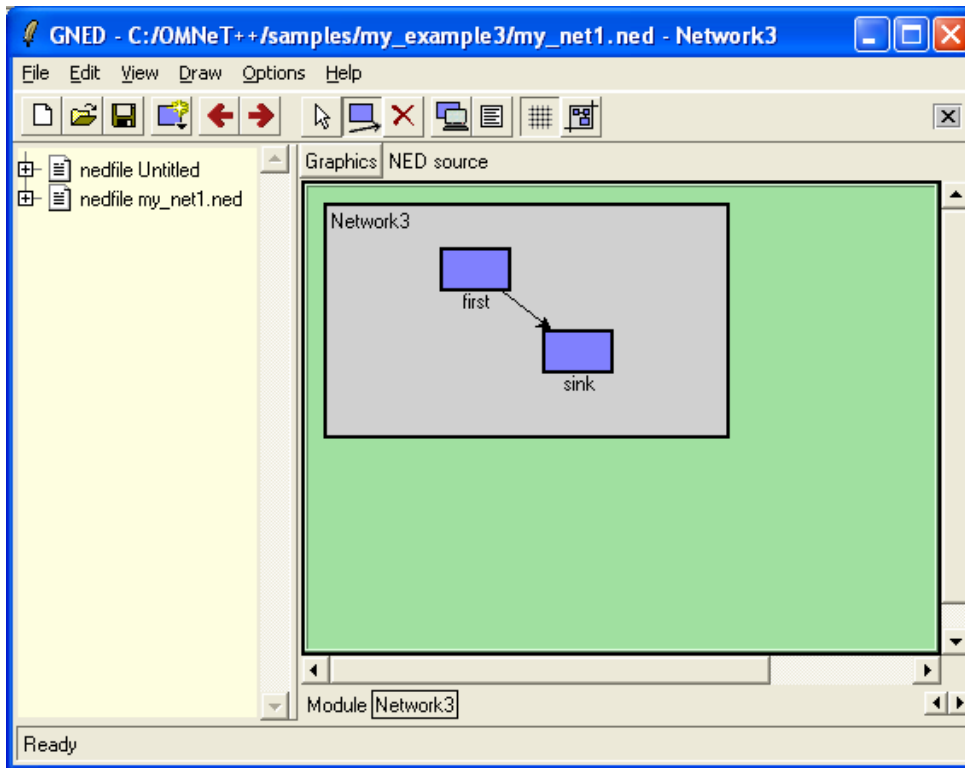
The sink3.ned file should contain

```

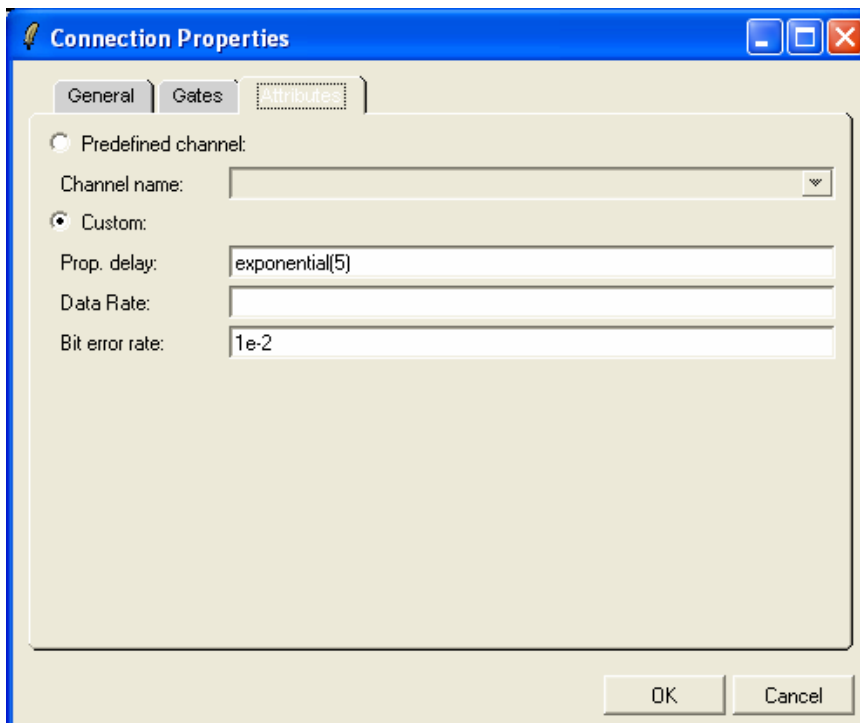
simple Sink3         //declaration of simple module object
    gates:
        in: in;
endsimple

```

Step 3. Visualize and maybe further edit the newly created ned files, using the graphical editor. In windows explorer, double click on my_net1.ned and my_net2.ned and they will be opened with the graphical editor. Here is the visualization for my_net3.ned:



To see the connection properties right click on the link and select “Properties” and then “Attributes”. A new window specifying the connection attributes will open:



Step 4. Add functionality. We now define gen1.cpp and sink3.cpp files.

```
// gen1.cpp code
// This file implements functionality for the generator module

#include <omnetpp.h>

class Generator: public cSimpleModule
{
    // This is a macro; it expands to constructor definition.
    Module_Class_Members(Generator, cSimpleModule, 8192);
                        // 8192 → estimated stack size needs

    // The following redefined virtual function implements the
    // module functionality.
    virtual void activity();
};
// The module class needs to be registered with OMNeT++

Define_Module(Generator);

void Generator::activity()
{
    const ia_time = 120;
    double temp;
    int num_pkts = par("number_rand");
    for (int i=0; i<num_pkts; i++)
    {
        // create message
        cMessage *msg = new cMessage("packet");
        msg->setLength(100); //we set a packet length of 100 bits (if
// no packet length is set, the packet is assumed of zero length, and
// the function that introduces errors does not function properly
        msg->setTimestamp(simTime()); //create time stamp for the
message
        send(msg, "out");
        wait((double)ia_time);
    }
}
```

The sink3.cpp must collect statistics for the received packets. First, it checks for packet errors, and if in error displays a message “packet lost” (using an animated bubble) and then updates the statistics for packets lost. Then checks the delay experienced by the packet, and updates statistics for delays.

We will implement the sink3.cpp file using a handle message approach, to also illustrate this technique.

The sink3.cpp file is given below:

```

// sink3.cpp code
// This file implements functionality for the very simple sink module

#include <omnetpp.h>

class Sink3: public cSimpleModule
{
    // This is a macro; it expands to constructor definition.
    Module_Class_Members(Sink3, cSimpleModule, 0);

    // The following redefined virtual function implements the
    // module functionality.

    cStdDev delayStats;
    cStdDev errStats;
    virtual void handleMessage(cMessage *msg);
    virtual void finish();
};
// The module class needs to be registered with OMNeT++

Define_Module(Sink3);

void Sink3::handleMessage(cMessage *msg)
{
    bool b;
    double delay;
    b = msg->hasBitError();
    if (b)
    {
        bubble("Packet lost!");
        errStats.collect(1);
    }
    else
    {
        errStats.collect(0);
        delay = simTime()-msg->timestamp();
        delayStats.collect(delay);
    }
    delete msg;
}

void Sink3::finish()
{
    ev<<"Average delay time:      "<< delayStats.mean() <<endl;
    ev<<"Maximum delay time:     "<< delayStats.max() <<endl;
    ev<<"Std. dev for delay time:   "<< delayStats.stddev() <<endl;
    ev<<"Average Packet Error Rate: "<< errStats.mean() <<endl;
    ev<<"Std. dev. Packet Error Rate: "<< errStats.stddev()
<<endl;
}

```


Step 5. We now create the Makefile which will help us to compile and link our program to create the executable my_example1. Note that the name of the executable is always the same as the name of the directory that holds the files.

In the command window, type the following sequence of commands:

```
opp_nmake -f
```

This creates a Makefile.vc in the working directory my_example1.

```
nmake -f Makefile.vc depend
nmake -f Makefile.vc
```

Notes:

1) If you get *'nmake' is not recognized as an internal or external command...*, find **vcvars32.bat** somewhere in the MSVC directories, and run it first thing in every command window in which you want to compile.

If there are compilation errors, you need to rectify those and repeat the make until you get an error-free compilation and linking.

2) If depend is not recognized, ignore and continue with next step (newer versions add dependencies automatically).

Step 6. Create the configuration file

If you start the executable now, it will complain that it cannot find the file **omnetpp.ini**, so you have to create one. **omnetpp.ini** tells the simulation program which network you want to simulate (remember that we have defined two different networks, one with one single generator, and the other one with two). In the configuration/initialization file, you can pass parameters to the model, explicitly specify seeds for the random number generators etc.

We will create a very simple **omnetpp.ini** file:

```
# This file is shared by all network descriptions: Network1 and
# Network2
# Lines beginning with '#' are comments

[General]
preload-ned-files=*.ned
network=Network3 # this line is for Cmdenv, Tkenv will still let you
choose from a dialog

[Parameters]
```

```
my_net3.first.number_rand = 100;
```

Step 7. Once you complete the above steps, you launch the simulation by issuing this command:

```
my_example3
```

and hopefully you should now get the OMNeT++ simulation window.

Step 8: Homework assignment:

Run the above tutorial for different values of the parameter “number_rand” (specified in the omnetpp.ini file), e.g. 100, 500, 1000, and determine the stats for the delay and packet error rate. Compare with the theoretical results you would expect. Repeat for different packet lengths: 1, 10, 20, 100, 1000. In order to repeat for different packet lengths, change the code to make the packet length a parameter that can be Initialized at run time.

Hint: You need to compute the packet error rate, under the assumption that each bit in a packet is independently affected by errors.

Note: If you cut and paste the text of the tutorials, you may have to retype in the Microsoft Visual C++ Development Environment some special characters, such as “”, which may not be correctly interpreted.