ELEC3500 Simulation Experiment 1 Report – Introduction to OMNeT++ Modelling

Introduction

The purpose of the simulation was to provide a beginner-level base of knowledge about the features of OMNeT++. The topics covered included:

- Proper installation of OMNeT++.
- Setting up debug/run environments to run the simulation, including default run options.
- Updating the 2-node "network" to test the available features, parameters and variables. This includes concepts like altering colours and other graphics, default variables, setting states, timers and NED file inheritance.
- Adding nodes, connections and channels to the 2-node system to make a full network. This includes simplifying pathways with 2-way connections and setting connection definitions. Further extending message functionality by defining a Message class is also included in this section.
- Collecting statistics, including packet sent/received counters, adding figures and adding detailed statistics with and without modifying the model.
- Producing a visual representation of the results. Results can be output as scalar (.sca) and vector (.vec) files, and there are a variety of plots available.

Simulation Model and File Structure

The model used in part 4.4 is a 6-module simulation, with module 1 and 4 serving as central points. The modules are connected as follows:

	tic[0]	*tic[1]*	*tic[2]*	*tic[3]*	*tic[4]*	*tic[5]*
tic[0]	x	out	x	x	x	x
tic[1]	out	x	out	x	out	x
tic[2]	x	out	x	x	x	x
tic[3]	x	x	x	x	out	x
tic[4]	x	out	x	x	x	out
tic[5]	x	x	x	x	out	x

Table 1.

Every time a message enters a module (on the x-axis), a message will then be sent **out** to a random module. For example, a message leaving *tic[0]* will be sent to *tic[1]*, but a message leaving *tic[1]* will go to either *tic[0]*, *tic[2]* or *tic[4]*.

Module and network information is stored in *tictoc13.ned*. Modules are generated with simple **inout** gates. This allows modules to have connections to multiple modules. The network contains 6 submodules of type *Txc13* and the channel has a delay of 100ms. The individual connections are listed in table 1.

Functionality is stored in the *Txc13* class in the *txc13.cc* file and includes 4 functions: initialize(), *generateMessage(), forwardMessage() and handleMessage(). initialize() boots the process and sends a message for *tic[0]*. generateMessage() determines what message will be sent, including source and destination. handleMessage() shows when messages are made, sent and received. forwardMessage() shows how the message is sent.

Messages are a separate class called *TicTocMsg13* which is stored in *tictoc13.msg*. The class has three attributes: source (where the message is from), destination (where the message is going) and hopCount (how many times a message has hopped to another module).

The *omnetpp.ini* file imports network *Tictoc13*. *Tictoc13* imports modules *Txc13*. *Txc13* imports message *TicTocMsg13*.

Model Modification

The model was modified in part 4.1 to add more than 2 modules. This was achieved by initializing an array of modules. Using this method, a theoretically infinite number of modules can be added to a model, although building the connections between module becomes exponentially more difficult, or at least tedious.

The function forwardMessage() was added to redirect messages whenever the simulation was intended to continue. For each call of handleMessage(), if the simulation is not on module 3, forwardMessage would be called to choose a random “out” gate. A forward message would then be directed to that gate.

Finally, the simulation was updated to end when module 3 is reached. An arrival message is sent and the simulation stops itself.

Results

1.
```
Tictoc7.tic.delayTime = exponential(3s)
Tictoc7.toc.delayTime = truncnormal(3s,1s)
```

Tictoc7.tic:Wait time, min: 2.86967
Tictoc7.tic:Wait time, max: 6.77057
Tictoc7.tic:Wait time, mean: 4.22375

2.
```
Tictoc7.tic.delayTime = exponential(7s)
Tictoc7.toc.delayTime = truncnormal(4s,8s)
```

Tictoc7.toc:Wait time, min: 1.77297
Tictoc7.toc:Wait time, max: 9.85909
Tictoc7.toc:Wait time, mean: 6.98666

3.
```
Tictoc7.tic.delayTime = exponential(3s)
```

Tictoc7.tic:Wait time, min: 1.55538
Tictoc7.tic:Wait time, max: 6.77057
Tictoc7.tic:Wait time, mean: 3.98031

Knowledge

1. The .ned files define the components that make up a network, including channels, submodules and connections. The .ini files show which network to launch, potential seeds and the initial parameters of a given simulation. The .cc files implement the behaviour of the simulation.

2. The initialize() function tells the simulation what to do when it first starts. The handleMessage() function defines the what happens when a module receives a message.

3. Random number generators are used throughout the simulation to randomly select:

    - a gate for a message to be passed to – this applies to models with more than 2 modules.

    - the length of time delay, for example how long a message a held before being sent.

    - whether a packet (message) is “lost”.

    The random number generator will always produce the same sequence of events. This is because the seed value will always be the same, unless a seed value is provided in the .ini file.

4. In this case, exponential distribution increases message wait time by an unbounded amount over a period of time. By contrast, truncated normal distribution bounds the variable to a certain minimum or maximum value.

5. Vector files present data as a function of time, for example showing how many messages were sent after 5 seconds. Scalar files show total aggregate data, such as the maximum delay before sending a message from module 2.

6. The statistical distribution functions can be very useful. If we were to apply a mean (average) operation to a message count, we can see how much certain modules are favoured, helping reduce bottlenecks and evenly distribute resources.