MONASH UNIVERSITY
DEPARTMENT OF ELECTRICAL & COMPUTER SYSTEMS ENGINEERING
Performance of Telecommunication Networks

Experiment V: Introduction to Modeling Wireless Networks and Their Network Layer Protocols

Alexander Senior Y. Ahmet Şekercioğlu

Contents

1 Flooding

Flooding is one of the simplest versions of routing: if a node wishes to send a message, it simply broadcasts the message to all of its neighbours. This experiment will look at three network protocols that use this algorithm:

- A 'naive' protocol, which broadcasts messages and nothing else. To make sure the network is not inundated with old messages, each message can only travel a certain number of hops before it is deleted.
- A 'smarter' protocol, which makes the obvious improvement of not sending messages back to nodes that have just sent them. It also sends acknowledgements straight back to the node which last forwarded the data packet.
- The 'smartest' network protocol[1], where each node maintains a routing table that is built up as messages pass through it. This protocol only floods messages initially, when

[1]That is, the smartest of the ones given here.

it doesn't have any information about the network; when it has learned the 'best'[2] routes, it sends them normally.

In this lab we will graduate to simulating wireless networks, thanks to an extension to OMNeT++ called MiXiM[3] (more information given in the next section). Given that we are simulating a wireless network, you may be wondering how we can choose to *not* send messages to certain nodes (as done in the 'smarter' and 'smartest' protocols): in the message formats for these protocols, a 'ignore' address field is specified; nodes drop packets immediately if their address is in this field.

2 MiXiM

The typical MiXiM simulation functions slightly different from the typical OMNeT++ simulation, so you are advised to both read the sections below, but more importantly investigate the MiXiM source code (at least the code that is referenced in the lab) to get a grasp of how it is set out and links together.

2.1 Template Code

MiXiM, in addition to providing the capacity to simulate wireless networks in detail, also provides a great deal of code to help you write simulations as quickly as possible. It already provides modules that emulate a wireless node running a typical TCP/IP networking stack; the one we use in this lab is `WirelessNode`[4].

MiXiM also defines a base class, `BaseLayer`[5], for the layers that compose a TCP/IP stack; this is extended by the classes `BaseNetwLayer`, `BaseApplLayer` and so forth. These layers are connected solely to the layers above and below them (except for the application layer, which has no layer above it), and there is code within these classes to speed development. Instead of having to define a `handleMessage` method to inspect the source of messages and then take action, `BaseLayer` defines methods such as `handleUpperMsg` and `handleLowerControl` that deal with data messages from the upper layer and control messages from the lower layer (for example). Sub-classing of the appropriate layer and re-implementing the `handle*` methods is highly recommended (and used in the Flooding project) to cut down on the code you need to write.

2.2 Simulation Configuration

The configuration of a MiXiM simulation is also a little different to a typical OMNeT++ simulation. The prime difference is that the nodes are *not* connected in the network module; instead, they are left unconnected, and the MiXiM framework will automatically connect them *if* they theoretically would be able to communicate with each other in the wireless medium. This in turn means that you do not define topologies in the network module; instead, you define the *position* of the nodes in the configuration files (i.e. the `.ini` files),

[2]Here, 'best' means the node with the shortest number of hops to the destination; what other metrics might be a better fit, given the type of network we are simulating in this experiment?

[3]See http://mixim.sourceforge.net/

[4]Defined in `MiXiM/src/modules/node/WirelessNode.ned`

[5]Located in `MiXiM/src/base/modules`.

and MiXiM will form connections as appropriate. Positions are defined by specifying the `node[*].mobility.initialX`, `node[*].mobility.initialY` and `node[*].mobility.initialZ` keys in the configurations files[6].

`omnetpp.ini` will have a few more keys than you might be used to; the majority of them specify settings for the radios the nodes use, so you don't have to be concerned with them for the moment (though feel free to investigate!). Some more important ones are the `**.node[*].<layer>type` keys; these set the exact layers are used for the physical, MAC, network and application layers (they are left as parameters in `WirelessNode`, so you can easily swap them out as needed).

3 Project Layout

The following files and modules are used in the Flooding project:

`FloodingApplLayer` This module is derived from MiXiM's `SensorApplLayer`, which provides a great deal of useful functionality; this application layer will send a set number of messages to random nodes in the network.

`FloodingNetwLayer` This module serves as a base for the three protocols implemented in this lab; it holds as much common code as possible. Note that you won't be able to create one of these in the simulation (nor should you try!), as the C++ class leaves several methods as pure virtuals.

`FloodingNetwLayerNaive` This extends the base class, and implements the most primitive flooding protocol.

`FloodingNetwLayerSmarter` This extends the base class, and implements the slightly smarter flooding protocol.

`FloodingNetwLayerSmartest` This extends the base class, and implements the most sophisticated of the protocols presented here.

`SimpleArp` This extends off of MiXiM's `BaseArp`, and takes a very simple approach to addressing (so it's easier for you to see what's happening in the simulation): both the network and MAC addresses of the nodes are their index in the vector holding all the `WirelessNodes`'s.

`config.xml` This file holds configuration options for the radio model the simulation uses; you don't have to concern yourself with it during this lab.

`Flooding.ned` This holds the network module for the simulation, though as explained earlier this is emptier than you might expect.

`FloodingPkt.msg` This holds the message format used in the 'smarter' and 'smartest' network protocols.

[6]'Initial' is used because the nodes can actually move around during the simulation, by specifying the appropriate mobility module; the simulation can also cope with a 3D playground.

`omnetpp.ini` This holds most of the configuration details of the simulation, with the exception of the node positions; see `default_network.ini` and `diamond.ini` for these. Note that you don't want to use the `General` configuration directly; see `configs.ini`.

`default_network.ini` This specifies the positions of the nodes for the default network MiXiM creates. Note that you don't want to use the `default_network` configuration directly; see `configs.ini`.

`diamond.ini` This specifies the positions of nodes in a simpler network setup. Note that you don't want to use the `diamond` configuration directly; `configs.ini`.

`configs.ini` This file includes the general settings from `omnetpp.ini`, and extends the configurations from `default_network.ini` and `diamond.ini`. It holds the configurations that you actually want to use; they use the positions inherited from either `diamond.ini` or `default_network.ini`, and specify a network protocol.

4 Set-up

1. Download the archive file for MiXiM 2.3 from the MiXiM Website[7]. Note that the plain 2.3 version will suffice, the inet version is *not* required. Import the project into your OMNeT++ workspace (the procedure is the same one you have used before, except you are importing from an archive file) and build it. Make the changes to the source code as described in Appendix A, and build it.

2. Download the 'Flooding' lab file set from the Unit's Website[8]. Import the project into your OMNeT++ workspace and build it. **Note**: both the MiXiM and the Flooding projects should be in the same directory, ideally directly in your OMNeT++ workspace, like so:

```
workspace
├── MiXiM
└── Flooding
```

3. Investigate the operation of the each of the three flooding protocols (starting with the naive version is recommended!) and see if they work as you expect. If the activity in the network is preventing you from seeing how the protocol works, there are a couple of things you can do:

    - Change the simulation so that only one node in the network generates messages. `FloodingAppLayer` contains the parameter `sendingNodeIndex`, which by default is set to -1; this means that all the nodes in the network generate messages. By changing this in the configuration files, you can set only the node you choose to send messages, which will help you determine what is happening.
    - Use the 'diamond' topology. This is much simpler than the default topology, and may also make things clearer.

[7] http://mixim.sourceforge.net/

[8] http://titania.ctie.monash.edu.au/netperf/flooding.tar.gz

5 Exercises

1. Add code to the network layer to count the number of messages that are sent versus the number that are received as a function of time. How does it change between the three protocols?

2. The 'smarter' flooding network protocol introduces 'ignore' addresses, i.e. a packet field that contains the address of a node that should disregard the message (because they have already forwarded it). The logical extension of this is having the messages contain a *list* of nodes that have already forwarded the message. Implement this functionality; a good method is to have an array of fixed size (so the packet length doesn't grow too large) and 'shift out' the older addresses. In the small network given, this might not have a large enough effect, so create a bigger one if little difference is seen.

3. The third of the flooding algorithms introduced above uses its routing table to remember the "best" link to use for any given destination. In `FloodingNetwLayerSmartest`, the size of this routing table grows (without bound) until, eventually, each node knows of every other node. In a small network this has many advantages, but in a network of several hundred or thousand nodes this becomes unrealistic. Extend the 'smartest' network layer so that the maximum size of each routing table is, say, only 4 nodes. Consider the following issues:

   (a) What should now happen when the routing table fills?

   (b) How will this affect certain efficiency metrics of the network as a whole?

A Changes to MiXiM 2.3

If you are downloading MiXiM 2.3 directly from their website, or you need to recompile MiXiM for some reason, note that there are a couple of changes that we have made to the archive on the Unit Website:

- In `MiXiM/src/inet_stub/linklayer/contract`, in `MacAddress.h` on line 28 and `MacAddress.cc` on line 137, the `L`'s on the hex literals needs to be changed to `ULL` (otherwise MiXiM will not compile on the lab machines).

- In `MiXiM/src/modules/messages/`, you need to run (from the command line) `opp_msgc *.msg` (for some reason, on the machines in the lab, the source files are not automatically generated from the `.msg` files, which will lead to compilation failures).