MANET extensions to ns2

Andrés Lagar Cavilla
andreslc@cs.toronto.edu

Department of Computer Science
University of Toronto

This is a set of software modules providing additional models for the simulation of multihop mobile ad hoc networks (MANETs) in the ns2 simulator. We provide implementations of the CM and Shell mobility models, and the AF and LOS radio propagation models described in [1]. We also provide instrumentation tools for the ns2 simulator, as well as template simulation scripts and results parsers.

The software presented here builds on the Monarch ns2 wireless extensions [2], as well as on some pre-existing mobility and radio propagation models, such as Random WayPoint [3], Free Space and Two-Ray Ground. We assume that the reader is familiar with the concept of MANETs, with the models aforementioned, and with the internals of ns2. All the software is based on version 2.26 of the ns2 distribution.

This documentation is split in two sections. The first part is a walk-through around the provided software, explaining how to conduct a simulation similar to the ones we carried out in [1]; we have bundled in the tarball the necessary scenario files. In the second part we go deeper into the fundamentals of each model and the particular implementation at hand.

The software is provided "AS IS", i.e. we do not take any responsibility whatsoever on any damage it may cause to your system or data. As long as you give proper credit, you can use the software freely for whatever legal purposes you may wish, and you can also modify it at will.

1 The Walk-Through

1.1 Mobility Pattern Generation

If you are reading this, then you have successfully decompressed the tarball, and are now wondering where to start among all those directories. The first thing you want to do is go to the `setdest` directory and build the four flavors of setdest we have there. You need to edit `Makefile` to have the `NSDIR` variable point to the root of your ns2 source tree. Once you've done that, `make` should do its job with no problems.

A typical simulation scenario in [1] was a network of 40 nodes moving at 3 m/s during 1200 s, following the CM mobility model, with no pause time in an area of 88 m by 113 m, and with a sensitivity threshold of -81 dBm assuming AF propagation. To randomly generate a mobility pattern according to those specifications, you have to type

```
./setdest_cm_af -n 40 -r -81 -p 0 -s 3 -t 1200 -x 88 -y 113
-f bahen_cm_graph.txt -g bahen_af_pic.pnm -d bahen_af_defs.txt
> mobility-40-cmaf-81.1
```

This will take a while, and the mobility pattern will be written to the `mobility-40-cmaf-81.1` file. You may want to generate 4 more mobility pattern files with the same specifications, labeled `mobility-40-cmaf-81.2` to `mobility-40-cmaf-81.5`. This way you can run 5 different experiments and average the results.

`setdest` does not only generate a mobility pattern, but also computes some topology metrics derived from that pattern, and appends them as comments at the end of the output file. The next step is then to go to the `parsers` directory and compile the Java program `MobilityParser` by typing `javac MobilityParser.java`. Then you can move the resulting `.class` file to where the mobility pattern files are located, and run `java MobilityParser mobility-40-cmaf-81 5`. This will average the mobility metrics in each of the 5 files you have generated, and dump the results to `mobility-40-cmaf-81.csv`. This is a comma-separated-values file, readable by Matlab and any spreadsheet program. This `csv` file in particular will contain the values for three metrics:

Link Changes: a global count of how many times the connectivity between each pair of nodes is interrupted and restored.

Neighbor Density: the average number of peers within connectivity range for any given node at any given time.

Optimal Path Length: the average length in hops of the optimal path between every pair of nodes for which a path actually exists, at any given time.

Now you have five mobility pattern files for a given scenario and the corresponding topology metric. You can create mobility patterns for roughly the same scenario, but now with FS propagation (in this case we'll specify a connectivity range of 35 m):

```
./setdest_cm -n 40 -r 35 -p 0 -s 3 -t 1200 -x 88 -y 113
-f bahen_cm_graph.txt > mobility-40-cm-35.1
```

This will take quite less time. You can also create five of these and average their topology metrics.

1.2 Generating a traffic pattern

In the `cbr` dir you'll find `cbrgen.tcl`, a CBR traffic pattern script generator. To generate a traffic pattern suitable for the scenarios we've been working with above, type

```
ns cbrgen.tcl -nn 40 -mc 20 -rate 4 -size 64 -seed <seed> > cbr-40
```

where `seed` is some random floating point number between zero and one.

1.3 Integrating the propagation models to ns2

In the `propagation` directory you'll find 4 files with the implementation for ns2 of the Attenuation Factor and Line of Sight propagation models: `propagation_af.cc` and `.h`, and `propagation_los.cc` and `.h`. Move this files to the `mobile` directory in your ns2 source tree, and add them to the list of object files in the ns2 `Makefile`: somewhere around line 216 in your ns2 `Makefile` you should be able to add the following line

```
mobile/propagation_af.o mobile/propagation_los.o \
```

Then recompile the whole simulator. You have now added both propagation models to ns2. However, before recompiling ns2 you may want to read the following section.

1.4 Integrating our instrumentation extensions to ns2

We have developed a set of instrumentation extensions to ns2. These extensions track down the values of several performance metrics during a MANET simulation, and pour the final values to `stdout` once the simulation is finished. The metrics reported are:

packets sent

packets received

packet delivery rate

packet delivery latency: in miliseconds

routing packets transmitted

routing bytes transmitted

normalized routing load: The ratio of routing packets transmitted to application-layer packets delivered.

Routing optimality: expressed as the number of additional hops a packet traversed with respect to the optimal route at the time of arrival.

You may want to give a try to our instrumentation extensions. In such a case, go to the `instrumentation` directory and copy each file there to its proper place in the ns2 source tree (**Always do backups!!!**): `cmu-trace.cc` to `trace`, `god.cc` and `god.h` to `mobile`, `dsdv.cc` to `dsdv`, and `packet.h` to `common`. Note that the modified modules are based on version 2.26 of ns2. Now is a good time to recompile the whole simulator from scratch.

1.5 Running the simulations

Now you have all the components you need to run a simulation such as the ones we ran in [1]. In the `script` directory you'll find a tcl script (`wireless_af_inst.tcl`) properly set up to run a simulation using AF propagation. But first some preparations (all paths relative to the location of the `wireless_af_inst.tcl` script):

- Create a `mobility` directory and move the mobility pattern files there.
- Create a `cbr` directory and move the traffic pattern file there.
- Copy (or link) the `bahen_af_defs.txt` and `bahen_af_pic.pnm` file from the `setdest` directory to where `wireless_af_inst.tcl` is.

Now you're ready to go. Type

```
ns wireless_af_inst.tcl -nn 40 -range -81 -proto dsdv -mobility cmaf
-option 1 > results-40-81-dsdv-cmaf.1
```

To run the simulations using the other mobility patterns you generated type for example

```
ns wireless_af_inst.tcl -nn 40 -range -81 -proto dsdv -mobility cmaf
-option 3 > results-40-81-dsdv-cmaf.3
```

And to run the same scenario but with dsr routing type

```
ns wireless_af_inst.tcl -nn 40 -range -81 -proto dsr -mobility cmaf
-option 1 > results-40-81-dsr-cmaf.1
```

Note: If you chose not to use our instrumentation extensions then you can use instead the script `wireless_af.tcl` with the same parameters. Note that now you don't need to pipe `stdout` to any file. The following section is of no interest to those who chose not to use our instrumentation extensions.

1.6 Parsing the results

Now you have five result files generated by our instrumentation extensions, `results-40-81-dsdv-cmaf.1` to `results-40-81-dsdv-cmaf.5` (you may also have five similar files for dsr routing). If you want to average the results for the five runs, go to the `parsers` directory and compile `OutputParser.java`. Then copy the resulting `.class` file to where your results file are and type

```
java OutputParser results-40-81-dsdv-cmaf 5 1200 64
```

This will generate `results-40-81-dsdv-cmaf.csv` a (hopefully) nicely labelled and easily understandable digest of the simulation results.

2 Detailed Description

2.1 Starting with AutoCAD

which is, by the way, a registered trademark. In order to work with your instance of any of the models we provide, you will need to get hold of the AutoCAD blueprint of your floor plan, and convert it to a picture, preferably in png format. There's plenty of programs out there that can do that; one example is the opendwg utilities (www.opendwg.org).

2.2 AF propagation

Attenuation Factor is a radio propagation model that obeys the following equation:

$$P_{AF}(r, m_1, ..., m_\sigma) = P_o(r_o) - 10n \log_{10}\left(\frac{r}{r_o}\right) - \sum_{i=1}^{\sigma} m_i \cdot PF_i,$$

where P_o is the power at some nearby reference distance r_o, n is the path loss exponent that determines the rate at which power decreases with the distance r, m_i is the number of times the dominant ray (the straight-line trajectory) between transmitter and receiver collides with an obstacle of material type i, $1 \leq i \leq \sigma$, and PF_i is the attenuation due to obstacles of type i. In a nutshell, in AF propagation the dominant ray is used to count the number of obstacles (walls) between transmitter and receiver. Each wall material is tagged with a different attenuation factor, and the summation of the attenuation factors, plus a traditional log-distance path-loss component, constitutes the total signal attenuation with respect to the reference power P_0.

The AF propagation model is a module that can be integrated to ns2, as we have explained in subsection 1.3. The AF model needs a list of materials and their corresponding attenuation factors, as well as a means to detect intersections with obstacles. For the latter, we use a pnm picture of the floor plan, derived from the original AutoCAD file, in which each material has a different color. pnm is an easily-readable graphic format provided in the `netpbm` graphics package. If you have this package installed in your Linux distribution (you probably do), just run `pngtopnm` to convert a png to a pnm. You'll notice pnm's are bulky, however they compress fairly well. The `bahen_af_pic.pnm` file in the `setdest` directory is the one we used for our project.

To run the AF model we need to load the pnm picture and a text file specifying the color and attenuation factor of each material that should be considered, as well as the remaining AF parameters. The file `bahen_af_defs.txt` that you can find in the `setdest` directory will serve as an explanation:

```
-31.4627 1.96651 7
4.7727 255 255 255
4.7727 224 224 224
2.479 128 128 128
2.479 160 160 160
3.11104 160 160 255
3.11104 128 128 255
6.50076 64 64 64
```

where the first value is P_0, the second is n, the third is σ (the number of different materials), and the correspondingly following seven lines are the attenuation factor for each material and the rgb specification of the color of the material in the pnm picture. Inside your ns simulation script, you should set up the AF propagation command with the following commands:

```
set prop [new Propagation/AF]
$prop topography $topo
$prop load-defs <definitions file>
$prop load-pic <pnm picture>
```

where topo is a properly setup topography object with the simulation area dimensions. You can find these lines in the any of the scripts in the `scripts` directory.

2.2.1 Line-Of-Sight Propagation

The LOS propagation model is derived from AF. It simply prevents propagation once an obstacle in the direct path between transmitter and receiver is found; otherwise, it falls back to conventional Two-Ray Ground propagation. It therefore uses the same pnm picture as AF to detect obstacles. However, it does not need a definitions file; the model assumes by convention that anything that is not black is an obstacle. The script command sequence is thus the following:

```
set prop [new Propagation/LOS]
$prop topography $topo
$prop load-pic <pnm picture>
```

2.3 CM Mobility and TOOL

In Constrained Mobility, node movement is constrained according to the edges of a graph we call a Mobility Graph. A Mobility Graph is drawn on top of a floor plan, and its structure will constrain node movement in such a way as to mimic human movement. Nodes choose destinations from the set of leaf vertices of the graph – located in offices or conference rooms – and move to their new destination over the shortest path on the mobility graph, which involves going through the door, walking on a hallway, and so on.

TOOL is originally a multi-purpose program written by Tom Hart. We have tailored it for the generation of CM mobility graphs. Its entirely written in Java, so the first thing to do is move to the `TOOL` directory and compile everything: `javac *.java` should suffice. You then execute TOOL with the following command line:

```
java CoverGenerator <picture> <width in meters> <height in meters>
<width in pixels> <height in pixels>
```

where you provide a png picture of your floor plan, the width and height of the floor plan in meters, and the width and height of the picture in pixels.

TOOL's online help is pretty self explanatory, so we will not elaborate much. Anyway, here's is short list of what you need to do

1. Select the option "File/New Canvas".
2. In the "File/Options" box, unmark "Enforce Ordered Set" and set "Export Format" to "Graph Adjacency Matrix"
3. In your canvas window, press "a" to add vertices. You can drag them around to their proper place.
4. To draw an edge, choose the two end-point vertices (they turn red) and press "c". If you made a mistake, undo it by pressing "d".

5. To specify that a vertex is suitable as a node destination, choose it and press "e". You can undo this by pressing "e" again. Once the vertex is no longer selected, it should be blue.

6. Once you are done, save your work using "File/Save Canvas".

7. And finally, export your work usgin "File/Export". We will use this second file in the following stage.

To serve as an example, you can run

```
java CoverGenerator bahen.pnm 88 113 1464 1920
```

and after loading `bahen_cm_canvas.txt` you'll be able to export `bahen_cm_graph.txt`.

2.4 Setdest

In the `setdest` directory you'll find a set of variants of the original `setdest` program written by the Monarch group [2]. This program randomly generates a RWP mobility scenario and dumps it to `stdout` in the form of a TCL script that ns2 will later use. The arguments passed are the number of nodes, simulation time, width and height of the simulation area, pause time and node speed. It is the assumption that you're familiar with this program.

2.4.1 setdest_rwp

`setdest_rwp` is quite similar to the original `setdest`, but it takes an additional range argument: the radius in meters of the connectivity range of a single node, for free space or two-ray ground propagation. It uses this parameter to produce the three topology metrics we described in section 1.1. The following is an example of how to run `setdest_rwp` to get a rwp mobility pattern for 20 nodes moving at 3 m/s max speed in a 88 by 113 m rectangle during 1200 seconds. Transmission range is set to 25 m, and pause time is set to zero

```
./setdest_rwp -n 20 -r 25 -p 0 -s 3 -t 1200 -x 88 -y 113
> test.rwp
```

2.4.2 setdest_shell

`setdest_shell` generates a shell mobility pattern assuming also free space propagation. Shell mobility is a middle-ground between CM and RWP: it considers outer walls but not inner walls. Node mobility is therefore constrained to happen within the shell outlined by the outer perimeter of the floor plan under consideration. Functionality of `setdest_shell` is identical to that of `setdest_rwp`. However, we need an additional parameter specifying a picture of the floor plan in pnm format.

As in AF, the pnm picture is meant to depict the floor plan, for the purpose of distinguishing its boundaries, in this case. By convention, the exterior of the floor plan (those places outside of the shell, where you don't want your nodes to go to) has to be in red color. Look at `bahen_af_pic.pnm` for an example. To generate a shell mobility pattern with the remaining parameters the same as in the previous example, just type

```
./setdest_shell -n 20 -r 25 -p 0 -s 3 -t 1200 -x 88 -y 113
-f bahen_af_pic.pnm > test.shell
```

2.4.3 setdest_cm

`setdest_cm` generates a CM mobility pattern assuming free space radio propagation. It has the same parameters as `setdest_rwp`, but you also need to specify where the CM mobility graph is. This is the file you generated using TOOL. A simple example:

```
./setdest_shell -n 20 -r 25 -p 0 -s 3 -t 1200 -x 88 -y 113
-f bahen_cm_graph.txt > test.cm
```

2.4.4 setdest_cm_af

The final and most complex variant is that which generates a CM mobility pattern assuming AF propagation. We need to load the proper AF specification, as you would do in an ns2 simulation, in order to be able to compute the topology metrics. Apart from the CM graph, we need to load the pnm picture, as well as the definitions text file detailing the AF model parameters. The following will exemplify:

```
/setdest_cm_af -n 20 -r -71 -p 0 -s 3 -t 1200 -x 88 -y 113
-f bahen_cm_graph.txt -g bahen_af_pic.pnm -d bahen_af_defs.txt
> test.cmaf
```

Note that the range is now specified in as the sensitivity threshold that you will use in the simulation, in units of dBm.

2.5 Instrumentation

We also provide our own instrumentation mechanism to record performance metrics. It consists of a set of modified ns modules which you can find in the `instrumentation` directory. The mechanism is simple: a set of counters and tables are initialized inside the GOD (Global Operations Director) object, and every time an event is written to the trace file, the cmu-trace module communicates the event to GOD, where the proper counter/table is updated. By the end of the simulation, counter contents in GOD are poured to `stdout`.

Besides modifying the `cmu-trace.cc`, `god.cc` and `god.h` files, we also need to customize a few other modules. `packet.h` is changed to add a new timestamp field to the common packet header; this new timestamp will be used to record packet delivery latency. Since DSDV routing packets are usually not recorded as trace events, we modified `dsdv.cc` to directly tell GOD whenever it broadcasts a routing packet. Moreover, we include in the new version of `dsdv.cc` a fix for a well-known infinite loop bug.

If you want to use our instrumentation tools, just replace the modules in the correct places of the ns2 source tree and recompile. Note that our modified code is based on version 2.26. Also, two commands need to be added to your simulation script file (after the creation of the `god_` object):

```
$god_ set-rp $opt(proto)
$ns_ at $opt(stop) ‘‘$god_ dump-counters’’
```

2.6 Parsers

We provide two small Java[1] programs in the `parsers` directory. The idea is to automate the averaging of the results for a series of experiments targeting the same scenario, but with different randomly generated mobility patterns.

For example, assume that `results-cmaf-20-dsr-91.1` represents the performance results (obtained through our instrumentation extensions) of the first run of an experiment using CM mobility, AF propagation with a sensitivity threshold of -91 dBm, DSR routing and 20 nodes. Furthermore,
`results-cmaf-20-dsr-91.2`, `.3`, etc... are the files containing the results for subsequent runs of the same experimental configuration. Executing

```
java OutputParser results-cmaf-20-dsr-91 5 1200 64
```

will average the results of 5 of those files – Note that we require different runs to be identified by a different number after a period at the end of the file name: `.1`, `.2`, etc... – and write the averages to
`results-cmaf-20-dsr-91.csv`, the comma-separated-values files readable by Matlab and most spreadsheet programs. A csv output example is provided in the `parsers` directory. The third and four arguments passed to `OutputParser` are simulation time and packet size, respectively, used to compute overall throughput.

While `OutputParser` allows us to average the performance metrics for several runs, obtained through our instrumentation extensions, `MobilityParser` is applied to the mobility pattern files generated by our different flavors of `setdest` to average the topology metrics: link changes, neighbor densities and path length. The syntax is similar:

```
java MobilityParser mobility-cmaf-20-91 5
```

will average `mobility-cmaf-20-91.1` to `mobility-cmaf-20-91.5`, and output the results to `mobility-cmaf-20-91.csv`. Once again, a csv example for mobility averaging is provided in the `parsers` directory.

2.7 CBR

Traffic patterns are modeled after Constant Bit Rate sources. We slightly modified the original `cbrgen.tcl` script and placed it in the `cbr` directory. The new syntax is

```
ns cbrgen.tcl -nn <nodes> -mc <sources> -rate <pkts/second>
-size <pktsize> -seed <seed> > <destination file>
```

where the optional `seed` is a floating point number between zero and one. This will pour to `stdout` a tcl script that ns2 will interpret in order to set up the traffic pattern.

[1] Yes, we could have used Python or something else.

2.8 Scripts

We finally provide two template script (`wireless_af_inst.tcl` and `wireless_af.tcl`) used to run a ns2 simulation using AF propagation with or without our instrumentation extensions. The scripts are properly documented, and you can cut&paste whatever parts you need into your own scripts.

References

[1] A. Lagar Cavilla, G. S. Baron, T. E. Hart, L. Litty, and E. de Lara, "Simplified simulation models for indoor manet evaluation are not robust," in *Proceedings of the First IEEE Conference on Sensor and Ad-Hoc Communications and Networks*, Oct. 2004.

[2] The Monarch Project. [Online]. Available: http://www.monarch.cs.rice.edu

[3] J. Broch, D. A. Maltz, D. B. Johnson, Y.-C. Hu, and J. Jetcheva, "A performance comparison of multi-hop wireless ad hoc network routing protocols," in *Mobile Computing and Networking*, 1998, pp. 85–97.