Manual for *nemo* version 0.2

April 7th, 2014
Damien Magoni

1. Tutorial

Step 1: generate a map

To generate a map, use the following command:

```
map gen <map_type> <x_size> <y_size> <z_size> <space_step>
```

`<map_type>` : defines the various types of maps:

```
const map_setting_type map_flat = 1; // z = 0, attn = 0
const map_setting_type map_surface = 2; // only one point z1 in z for each
(x, y), attn = a(x, y) < 0 for z <= z1
const map_setting_type map_space = 3; // one or more points zi in z for
each (x, y), attn = a(x, y, z) <= 0
```

Only type 1 is supported for the moment.

`<x_size> <y_size> <z_size>` : x, y, z lengthes in meters
`<space_step>` : granularity in meter

Ex: `map gen 1 1000 1000 0 1`
This generates a flat 1000 m x 1000 m map.

You can save the map with:

```
map save <map_file>
```

And load it later with:

```
map load <map_file>
```

You can show the map description with:

```
show map
```

Step 2: generate a mobility scenario

To generate a mobility scenario, use the following command:

```
mob gen <duration> <nb_of_nodes> <nb_of_events> <max_speed> <max_accel>
```

`<duration>` : duration in seconds
`<nb_of_nodes>` : number of mobile nodes

`<nb_of_events>` : number of mobility events
`<max_speed>` : maximum speed in m/s
`<max_accel>`: maximum acceleration in m/s²

Ex: `mob gen 15.5 10 50 9 2`
This generates a random mobility scenario of 15.5 seconds with 10 nodes and 50 events.

You can save the mobility scenario with:

```
mob save <mob_file>
```

And load it later with:

```
mob load <mob_file>
```

You can show the mobility scenario's description with:

```
show mob
```

Here is an excerpt from a mobility file generated by the above exemple:

```
# time_s mnid pos_x pos_y pos_z vel_x vel_y vel_z acc_x acc_y acc_z
0 1 549 593 715 8 5 0 2 1 0
0 2 847 424 624 6 3 0 1 1 0
0 3 892 57 964 2 3 0 1 2 0
0 4 812 529 480 5 4 0 2 2 0
0 5 71 337 87 6 0 0 1 2 0
0 6 957 778 140 8 8 0 2 1 0
0 7 799 801 461 5 7 0 1 0 0
0 8 721 640 582 1 5 0 2 2 0
0 9 522 106 415 4 2 0 0 2 0
0 10 737 456 217 5 1 0 0 1 0

# time_s mnid vel_x vel_y vel_z acc_x acc_y acc_z
1.000000 3 7 4 0 0 2 0
1.000000 8 1 5 0 1 0 0
2.000000 7 2 6 0 0 2 0
2.000000 2 8 3 0 1 1 0
3.000000 8 3 5 0 1 1 0
3.000000 4 3 4 0 0 1 0
3.000000 7 6 8 0 1 1 0
3.000000 5 5 5 0 0 0 0
3.000000 6 6 2 0 0 2 0
4.000000 7 1 3 0 0 1 0
4.000000 8 5 4 0 1 2 0
…
```

Step 3: generate a connectivity scenario

To generate a connectivity scenario, you have two options:

1. generate it from a previously generated mobility scenario with the following command:

```
mob proc <time_step>
```

`<time_step>` : time resolution in seconds (i.e., connectivity will be measured every `<time_step>` interval)

Ex: `mob proc 0.5`
This generates a connectivity scenario from the previous mobility scenario with a 0.5 second sampling interval.

2. generate it randomly from scratch with the following command:

```
cnn gen <duration> <nb_of_nodes> <nb_of_events> <max_bw>
```

`<duration>` : duration in seconds
`<nb_of_nodes>` : number of mobile nodes
`<nb_of_events>` : number of connectivity events
`<max_bw>` : maximum bandwidth in bits/s

Ex: `cnn gen 15.5 10 50 10000000`
This generates a random connectivity scenario of 15.5 seconds with 10 nodes and 50 events.

You can save the connectivity scenario with:

```
cnn save <cnn_file>
```

And load it later with:

```
cnn load <cnn_file>
```

You can show the connectivity scenario's description with:

```
show cnn
```

Here is an excerpt from a connectivity file generated by the above exemple:

```
# time_s 1st_mid 2nd_mid status bandwidth_bps distance_m
0.000000 1 2 start 24000000 354.466
0.000000 1 4 start 24000000 358.455
0.000000 1 7 start 24000000 412.65
0.000000 1 8 start 36000000 222.445
0.000000 1 9 start 18000000 572.624
0.000000 1 10 start 18000000 549.652
0.000000 2 3 start 18000000 502.309
0.000000 2 4 start 36000000 181.62
0.000000 2 7 start 24000000 413.524
0.000000 2 8 start 36000000 253.567
0.000000 2 9 start 18000000 500.43
0.000000 2 10 start 24000000 422.816
0.000000 4 6 start 24000000 445.675
0.000000 4 7 start 36000000 272.973
0.000000 4 8 start 36000000 176.085
0.000000 4 9 start 18000000 516.966
0.000000 4 10 start 36000000 283.06
0.000000 5 9 start 18000000 603.611
0.000000 6 7 start 24000000 358.516
```

```
0.000000 6 8 start 18000000 519.715
0.000000 6 10 start 24000000 397.508
0.000000 7 8 start 36000000 215.977
0.000000 7 10 start 24000000 427.089
0.000000 8 9 start 18000000 593.84
0.000000 8 10 start 24000000 409.068
0.000000 9 10 start 24000000 455.992
0.500000 1 2 update 54000000 91
0.500000 1 3 start 36000000 249
0.500000 1 4 update 36000000 235
0.500000 1 6 start 18000000 575
0.500000 1 7 stop 0 869.908
0.500000 1 8 update 48000000 133
0.500000 1 9 stop 0 1353.66
0.500000 1 10 stop 0 1410.82
0.500000 2 3 update 24000000 340
0.500000 2 4 update 48000000 144
0.500000 2 5 start 18000000 537
0.500000 2 6 start 18000000 484
0.500000 2 7 stop 0 847.817
0.500000 2 8 update 54000000 42
0.500000 2 9 stop 0 1336.44
0.500000 2 10 stop 0 1381.32
0.500000 3 4 start 18000000 484
0.500000 3 8 start 24000000 382
0.500000 4 5 start 24000000 393
0.500000 4 7 stop 0 832.217
0.500000 4 8 update 54000000 102
0.500000 4 9 stop 0 1321.6
0.500000 4 10 stop 0 1345.95
0.500000 5 6 start 54000000 53
0.500000 5 8 start 18000000 495
0.500000 5 9 stop 0 1360.14
0.500000 6 7 stop 0 891.776
0.500000 6 8 update 24000000 442
0.500000 6 10 stop 0 1322.24
0.500000 7 8 stop 0 840.753
0.500000 7 10 stop 0 1579.29
0.500000 8 9 stop 0 1330.52
0.500000 8 10 stop 0 1369.53
0.500000 9 10 update 36000000 198
1.000000 9 10 update 18000000 548.952
1.500000 1 3 stop 0 1801.29
1.500000 1 7 start 36000000 254
1.500000 1 8 stop 0 1788.95
1.500000 2 3 stop 0 1816.11
…
```

Step 4: run the connectivity scenario in real time

To run the connectivity scenario, use the following command:

```
cnn start
```

You can also use these self meaning commands to alter the flow of the execution:

```
cnn {start|pause|resume|stop|reset}
```

Only `start` and `stop` are supported for the moment.

2. Commands

1.1 Manipulation of the underlying map

`map gen <map_type> <x_size> <y_size> <z_size> <space_step>` : generates a map.

`<map_type>` : defines the various types of maps:
`1 = map_flat` : defines a flat map where z = 0 and attenuation = 0 dB everywhere
`2 = map_surface` : defines a surface map where only one z value is defined for each point (x, y) and where an attenuation = a(x, y) < 0 for z <= z1
`3 = map_space` : defines a volume where the attenuation value is defined for any point in space

Only type 1 is supported for the moment.

`<x_size> <y_size> <z_size>` : x, y, z lengthes in meters
`<space_step>` : granularity in meter

`map {load|save} <map_file>` : loads a previously generated map / saves the current map.

Note: a `map` command must be issued before a `mob` command in order to have correct settings.

1.2 Non-real-time mobility scenario management

`mob gen <duration> <nb_of_nodes> <nb_of_events> <max_speed> <max_accel>` : generates a random scenario (i.e., not based on any mobility model) for testing purposes.

`mob {load|save} {nemo|ns2} <mob_filename>` : loads a previously created mobility file / saves the current mobility scenario in `mob_filename`.
Two formats are currently supported:

- The *nemo* format, for loading (input) or saving (output)
- The *ns-2* format, for loading only (which can then be converted to the *nemo* format)

`mob node {<id>|all} {wic {aironet|xjack|wavelan}|txp <dBm>|margin <dBm>|cfreq <Hz>}` : configures the wireless interface of some or all of the mobile nodes. Three models are currently defined in terms of reception thresholds (i.e., throughtput vs reception level):

- Cisco aironet
- 3com xjack
- Lucent wavelan

The transmission power level can be defined in dBm as well as the gain margin in dBm and the carrier frequency in Hz. The radio propagation model used is the two ray ground reflection model. Default values are:

- Cisco aironet (up to 54Mbps)
- Txp = 20 dBm, margin = 10 dBm, cfreq = 2.4897GHz

`mob proc <time_step>` : process the current mobility scenario and produces the corresponding connectivity scenario with a precision of `time_step` seconds. This value defines the resolution of the scenario and any event thus happens at a multiple of this value.

1.3 Real-time connectivity scenario management and execution

`cnn gen <duration> <nb_of_nodes> <nb_of_events> <max_bw>` : generates a random connectivity scenario for testing purposes (without the need of a prior mobility scenario).

`cnn {load|save} <cnn_filename>` : loads a previously created connectivity file / saves the current connectivity scenario in `cnn_filename`.

`cnn {start|pause|resume|stop|reset}` : starts / stops the real-time engine that executes the connectivity scenario (i.e., sends commands to another management program such as NEmu or *vnd*).

1.4 Miscellaneous commands

`show {map|mob|cnn|ep}` : shows current data related to the map, mobility scenario, connectivity scenario or endpoint.

`conn <ep> <proto> {<laddr>|*} {<lport>|*} <raddr> <rport>` : connects *nemo* to the management console (e.g., Nemu) via a TCP or UDP connection and creates a corresponding endpoint object named `ep`. All commands will be sent into this connection.

`disc <ep>` : disconnects *nemo* from a previously created connection and destroys endpoint object named `ep`.
`msg <ep> <txt>` : sends a message into the connection named `ep`.

Note: several endpoints can be created and simultaneously active. The commands issued by *nemo* will be sent into all endpoints.

`seed {t|<seed>}` : initializes the MT PRNG with either a random value (i.e., current time) with the flag `t` or a fixed `<seed>` value.

`load <command_file>` : instead of typing every command at the *nemo*'s prompt, you can write them in a text file and load them in *nemo* which will execute them sequentially.

`help` : gives the list and the syntax of all commands.

`debug {on|off}` : prints debugging information on screen.

`exit` : exits the program.

That's all folks!