Large Scale Experiments of Multihop Networks in Mobile Scenarios

Yacine Benchaïb
Sorbonne Universités, UPMC Univ Paris 06,
UMR 7606, LIP6, F-75005, Paris, France
yacine.benchaib@lip6.fr

Claude Chaudet
Institut Mines-Télécom / Télécom ParisTech /
LTCI CNRS UMR 5141
Paris, France
claude.chaudet@telecom-paristech.fr

ABSTRACT

This paper presents the latest advances in our research work focused on VIRMANEL and SILUMOD, a couple of tools developed for research in wireless mobile multihop networks. SILUMOD is a domain specific language dedicated to the definition of mobility models. This language contains keywords and special operators that make it easy to define a mobility model and calculate the positions of a trajectory. These positions are sent to VIRMANEL, a tool that manages virtual machines corresponding to mobile nodes, emulates their movements and the resulting connections and disconnections, and displays the network evolution to the user, thanks to its graphical user interface. The virtualization approach we take here allows to run real code and to test real protocol implementations without deploying an important experimental platform. For the experimentation of a large number of virtual mobile nodes, we defined and implemented a new algorithm for the nearest neighbor search to find the nodes that are within communication range. We then carried out a considerable measurement campaign in order to evaluate the performance of this algorithm. The results show that even with an experiment using a large number of mobile nodes, our algorithm make it possible to evaluate the state of connectivity between mobile nodes within a reasonable time and number of operations.

Categories and Subject Descriptors

C.2 [**COMPUTER-COMMUNICATION NETWORKS**]; I.6.7 [**Simulation and Modeling**]: Simulation Support Systems—*Environments*

General Terms

Testbed

Keywords

Wireless multihop networks, Mobility simulation, Virtualization, Experimentation

1. INTRODUCTION

Multihop networks have attracted researchers attention since the DARPA packet radio networks project in the 1970's. Research has fairly accelerated in the past decade and sensor, mesh and vehicular networks autonomous applications are progressively getting out of the labs. If several algorithms and protocols mostly target static networks or low mobility conditions, the interest in supporting more ambitious mobility models has increased lately.

For research in multihop networks, we developed and released under an LGPL license two tools: SILUMOD [5] and VIRMANEL [6]. SILUMOD is a domain specific language that allows to describe complex mobility models, i.e. how the various nodes in a network move around an environment that can include obstacles. SILUMOD is expressive and easy to use and to understand, making it perfectly suited for user than do not have advanced programming knowledge. And VIRMANEL is a virtualization tool based on OpenVZ [14] whose role is to manages Linux-based virtual machines corresponding to the nodes of a mobile network and emulates their movements by tearing up and down links dynamically, according to the mobility pattern defined in SILUMOD. The main advantage of the virtualization is that it make possible to conduct experiments in a virtual environment even if financial and human resources are limited. In fact, a unique physical machine is often sufficient for the realisation of experiments.

But one of the main difficulties in implementing a simulator for mobile multi-hop networks is the identification for a mobile node to nodes that are sufficiently close, for the establishment of a connection to transfer data. The nodes are frequently on movement so the simulator must at all times be able to evaluate the distance between the mobile nodes and update the status of virtual links between mobile nodes.

The SILUMOD/VIRMANEL couple combine the following advantages, which makes them really convenient for the testing of multihop wireless networks:

An easy language: SILUMOD language is dedicated to the definition of mobility models used to simulate the movement of mobile nodes in the case of multihop wireless networks. Its keywords are specific to mobility models, which makes it a language easy to understand. A user can define mobility models easily, understand models built by others and create and maintain a models database.

TRIDENTCOM 2015, June 24-25, Vancouver, Canada

DOI 10.4108/icst.tridentcom.2015.259821

Use of real code: As VIRMANEL uses virtualization, a user is able to test the real software implementation in a wide range of configurations. Similarly, the installation of a given application of protocol is as easy as on a real machine.

graphical user interface: VIRMANEL provides a graphical user interface that allows experimenters to monitor nodes mobility in real time.

scaling: VIRMANEL uses a very efficient virtualization solution based on containers. It makes possible to run multiple virtual machine instances on a single physical machine. In addition, regulation of the connectivity between the virtual nodes is provided by a controller using an optimized algorithm.

In the rest of this paper, we first presents in section 2 the VIRMANEL key aspects and the SILUMOD language. We then present in section 3 the algorithm that we implemented to reduce the search operations of the mobile nodes present in a coverage radius and evaluate this algorithm in section 4. Section 5 presents the related work. We then conclude the paper in section 6 and present our future research directions.

2. ABOUT VIRMANEL AND SILUMOD

VIRMANEL and SILUMOD are tools that facilitate the research in multihop networks, with a special focus on mobility. These tools were built with ease of use and flexibility in mind, as the user is able to evaluate a classical program running on Linux as well as real code running on an embedded operating system (Contiki, FreeRTOS, ...). We believe that VIRMANEL and SILUMOD provide a way to hide unnecessary technical aspects, as any programming language and operating system can be used, allowing users to focus on the mobility-related aspects.

2.1 VIRMANEL key aspects

VIRMANEL is a tool responsible for managing the states links between virtual machines that model the network nodes and for updating and displaying the network topology. Each mobile node is instantiated as a virtual machine based on OpenVZ[1], an efficient virtualization solution using containers to ensure an isolation of virtual machines. With a prepared template, it is possible to virtualize the network nodes running any Linux distribution or an embedded operating system running on an emulated MSP430 micro-controller. The template can include any software running under this environment.

VIRMANEL receives nodes positions from SILUMOD scripts (see section 2.2), computes the distances between pairs of mobile nodes, compare these distances to a nominal transmission range and update firewall rules dynamically making the topology of the network evolve constantly. In fact, the OpenVZ virtual machines that correspond to a wireless nodes are connected together by a single virtual switch, which theoretically allows them to communicate together directly. To emulate the limited transmission range of wireless interface, firewall rules are set on each virtual machine in order to block trafic coming directly (at the MAC layer) from too distant mobile nodes. By default, for each mobile node, all packets from other mobile nodes are dropped except in two cases. The first case is when two mobile nodes are close enough to establish communication, a firewall rule is added on each node to ensure communication. And the second case is when two mobile nodes are too distant for direct communication, a firewall rule on each node accepts packets from the other node only if they were routed, testing in particular the value of Time To Live (TTL).

Besides the virtualization and the mobility management engines, VIRMANEL includes a graphical user interface that allows to monitor nodes positions and connections, as represented on Figure 2. The GUI allows users to see rough mobility properties such as the repartition of the nodes on the playground, or the network disconnections. In addition, users can get a shell access to the virtual machines by running a manual command in a Linux terminal.

2.2 The SILUMOD language

SILUMOD is an open source (LGPL) internal domain-specific language that allows an easy and expressive description of the nodes movement patterns. Based on the Scala language [17], SILUMOD uses the same syntax and the same libraries as this language and adds a series of keywords and operators specific to the description of mobility. SCALA provides the structure of a classical languages, including iterative and conditional constructs, which allows to develop fairly complex models. As example of a SILUMOD program, the Random Waypoint Mobility Model is represented in Listing 1.

```
import silumod.language._

virmanelNode value "saturne"
virtualization value "no"
name value "Mobile"

playground define(0,0,600,400,"PG","blue","no")
speed value 50
positionX value 100
positionY value 200
distance value 150
coverageRadius value 100

var x = 0
while ( x < 100 )
       {
       toPositionX value ( 1 <-> 599 );
       toPositionY value ( 1 <-> 399 );

       while ( MOBILE calculateNewPosition )
              {
              if ( MOBILE ->| LIMITS )
                     {
                     angle reflexive;
                     }
              else
                     { MOBILE Move }
              }
       x = x + 1
       }
```

Listing 1: Specification of the Random Waypoint Mobility Model in SILUMOD

[1] http://www.openvz.org

With SILUMOD, it is possible to define the environment in which mobile nodes evolve, specifying the playground dimensions, positioning obstacles, and to define each node's movement characteristics through the specification of its movement speed, angle, etc. The user can describe the behavior of a node when it encounters an obstacle such as a wall, form groups of nodes and define zones in the scene so that mobiles change their behavior when they cross these zones limits.

This set of keywords allows to describe most classical mobility models, from the classical random movement patterns (random waypoint, random direction, etc.) to more complex and adaptive mobility patterns. For example, using zones, it is possible to vary speed according to different types of grounds, but also to define attractors and to model the movements of humans in a city, traveling between home and work.

SILUMOD is intuitive so that users can easily make movement parameters vary, implement new mobility models, or simply analyze a provided mobility description. We are currently working on creating a repository for mobility models that should be available online and distributed with future versions of the software.

The mobility model described in the SILUMOD language is then passed to a simulation engine that computes nodes successive positions and communicates these positions to VIRMANEL through classical inter-process communication. The user may easily keep track of the successive positions taken by the nodes by accessing the `positionX current` and `positionY current` variables and writing their values to a file. The nodes effective mobility can then be compared to the algorithm results, replayed and analyzed.

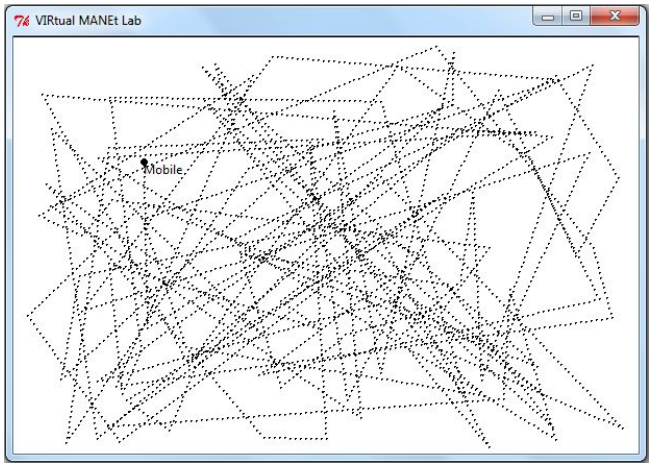


Figure 1: A trace of the Random Waypoint Mobility Model

The trajectory represented of figure 1 corresponds to a single node, whose movements are displayed in VIRMANEL.

Writing a small procedure, it is easy to log the successive positions of the nodes in a file and, consequently, to perform offline analysis of various properties. The successive network topologies can also be saved to a file and input into a dynamic graph analysis tool such as Gephi[2] or exploited with the GraphStream library[3].

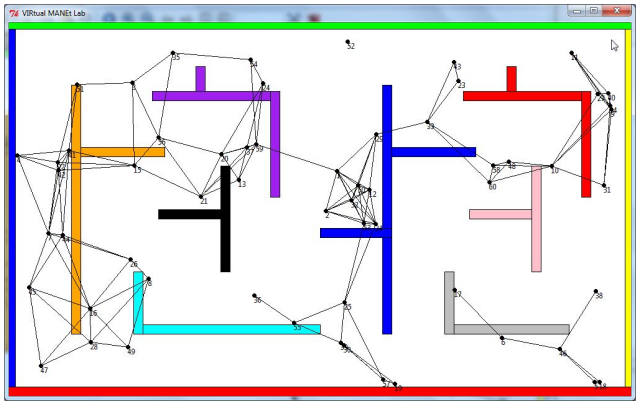


Figure 2: Graphical User Interface of VIRMANEL

Another use case of VIRMANEL allows real-time display of a mobile nodes network moving and the states links between mobile nodes (Fig. 2). The user can then monitor the mobile network and then adjust the parameters of the experiment based on observed facts.

3. EVALUATION OF THE CONNECTIVITY BETWEEN TWO MOBILE NODES

Evaluations conducted continuously between mobile nodes by the usual methods (section 5.1) to know the distance between them are often unnecessary. This section describes the method we have implemented in order to have a minimum number of evaluations of the states links between mobile nodes while ensuring the use of a maximum number of moving mobile nodes.

3.1 Definition of the critical zone

In fact, let M_1 and M_2 be two mobile nodes separated by a distance D_{M1-M2}. Assuming M_1 and M_2 move at a maximum speed V_{max} such that at each movement of M_1 and M_2 the distance between them tends to be the smallest possible. And given that M_1 and M_2 can communicate with another mobile node if the distance to the mobile node is less than $D_{propagation}$, then it is possible to define for how long T, no connection is possible between M_1 and M_2.

Let be a *critical zone* corresponding to a crown delimited by an inner circle with a radius $R_{int.}$ and an outer circle with a radius $R_{ext.}$. The evaluation of the distance between M_1 and M_2 is only necessary when M_1 is in the *critical zone* Z_2 of M_2 (and M_2 in *critical zone* Z_1 of M_1) because the connection status between M_1 and M_2 could change. In fact, taking the case of M_2, if M_2 is in the critical zone Z_1 of M_1, at the text move M_2 can either leave the zone covered by M_1, or stay while if M_2 is outside the critical zone Z_1, no change in the status of connection between M_1 and M_2 is possible at the next move of M_2

[2] `http://gephi.org`
[3] `http://graphstream-project.org`

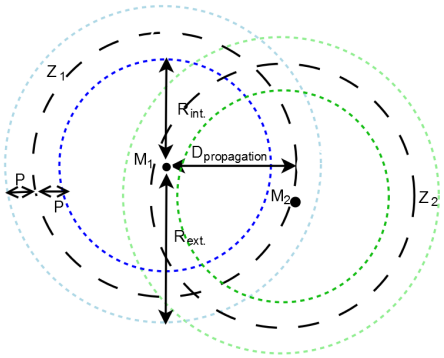


Figure 3: Critical zone of a mobile node

Two successive positions of a mobile node under SILUMOD are separated by a predefined step P. Thus:

$$R_{int.} = D_{propagation} \text{ - } P \; R_{ext.} = D_{propagation} + P$$

3.2 Number of steps before an evaluation of the connection status between two mobile nodes

M_1 moves at a speed V_1 on a distance D_1 during a time T_1 and M_2 moves at a speed V_2 on a distance D_2 during a time T_2. Thus:

$$V_1 = D_1 \; / \; T_1 \; V_2 = D_2 \; / \; T_2$$

Speed, distance and travel time are parameters calculated according to the *mobility model* selected in advance. It is also not necessary that M_1 and M_2 move according to the same *mobility model*. Thus, according to the limits set by the user in the specific SILUMOD script of each mobile node, these parameters can vary considerably from one limit to the other during the movement of a mobile node. These parameters can therefore not be used to predict situations requiring an evaluation of the connection status between two mobile nodes.

Nevertheless, the situation in the worst case can be evaluated assuming that M_1 and M_2 always move with the same maximum speed V_{max}. If the difference between the speed of mobile nodes and V_{max} is important, use this solution may require increased checks of the connection status between M_1 and M_2 relative to an accurate evaluation based on the respective parameters of M_1 and M_2. But, the initial overhead of using this solution is compensated because it avoid to revalue the situation between M_1 and M_2 at each change of parameters of M_1 or M_2. Reported to an important number of nodes connected to M_1 or M_2, the gain in terms of executed operations is substantial. By cons, if the difference between the speed of the mobile nodes and V_{max} is minimal, the additional number of checks the connection status between M_1 and M_2 will be negligible or null.

To evaluate the situation in the worst case, the margin of movement $Margin_{M1-M2}$ between M_1 and M_2 must be defined first. This margin of movement corresponds to the required distance (or remaining) for M_1 and M_2 before a possible change in their common connection status. Therefore:

$$Margin_{M1-M2} = D_1 + D_2$$

Either M_1 et M_2 are already connected and it must then evaluate on which distance D_1 and D_2 the mobiles nodes could move before not to be within reach of each other. Thus:

$$Margin_{M1-M2} = D_{propagation} \text{ - } D_{M1-M2}$$

Or M_1 et M_2 are not yet connected and it must then evaluate on which distance D_1 et D_2 the mobiles nodes could move before to be within reach of each other. Thus:

$$Margin_{M1-M2} = D_{M1-M2} \text{ - } D_{propagation}$$

Then, it must determine the values of distances $D_1 + D_2$ that M_1 and M_2 have to move before to be in the critical zone Z_1 of M_1 with a speed V_{max}. Thus:

$$\frac{D_1}{V_{max}} = \frac{D_2}{V_{max}}$$

$$\Rightarrow$$

$$D_1 = \frac{Margin_{M1-M2}}{2}$$

The number of steps $StepsNumber_{M1-M2}$ before an evaluation of the connectivity status between M_1 a,d M_2 by M_1 is then:

$$StepsNumber_{M1-M2} = \frac{D_1}{P}$$

3.3 Progress of the algorithm

Let be X a value in the range [1, N], N is the number of mobile nodes in the simulation. Each mobile node M_X is represented by a process $Proc_X$ in the application responsible of establishing connectivity between nodes. Each process receives the successive positions calculated by the script of the respective mobile node. The number of positions received by a process $Proc_X$ is defined by the variable $numberOfPositionsReceved_X$.

In an effort to optimize the computing time of the various processes $Proc_X$, every process $Proc_X$ is responsible for managing the connectivity of its associated mobile node M_X with at most $\frac{n}{2}$ mobile nodes. Thus, if $Proc_1$ manages the connectivity between M_1 and M_2, $Proc2$ is exempt of the managing of the connectivity of it's associated mobile node M_2 with M_1 (Fig. 4).

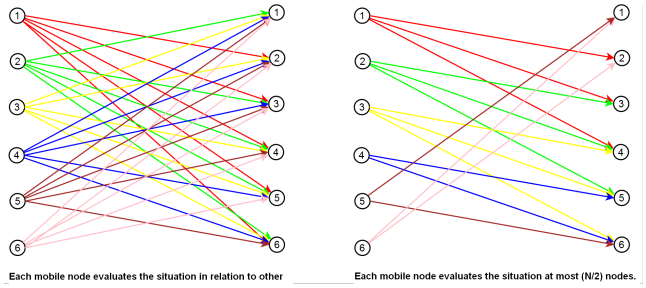


Figure 4: Management relation between processes

VIRMANEL being implemented in Erlang, a language that natively manage parallelism, connectivity management between different mobile nodes is done by processes running in parallel which significantly reduces the computation time required for the processing by each node of all mobile nodes under its management.

When the process $Proc_X$ receives the first position of the mobile X, it evaluates the number of steps $StepsNumber X - Y$ needed before an evaluation of the connectivity status with the Y mobile nodes under its management. The complexity of this initial stage is $O(\frac{n}{2})$.

A table Tab_{X-Y} whose size $maxNumberOfSteps$ is the number of steps between the two most distant points of the simulation area (the ends of the diagonal), contains the evaluation order of the mobile nodes Y by the mobile node X (Fig. 5). The mobile nodes Y are added ($O(1)$ for each) to a list in Tab_{X-Y} as an index I defined as follows:

$$I \equiv (numberOfPositionsReceived_X + numberOfSteps_{X-Y}) \pmod{maxNumberOfSteps}$$

When the mobile node X receives a new position, the index of the array Tab_{XY}, which serves as a moving cursor, is incremented. If the index contains at list with at least one mobile, then the process $Proc_X$ request to each mobile present in the list their current position. If the list is empty, then the $Proc_X$ waits to receive the next position.

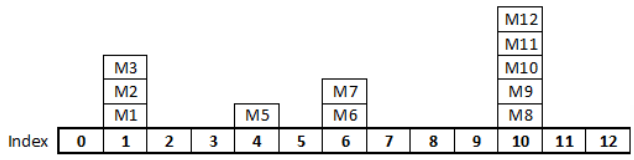


Figure 5: Data structure used for the classification of mobile nodes in the simulation plan

All the nodes positioned in the simulation area are added to an axis of length $maxNumberOfSteps$ depending on the position of a cursor representing the theoretical advancement of the mobile node associated with the process $Proc_X$. Thus, no sorting operations and no change of the data structure that is time consuming is needed to manage the mobile nodes.

Furthermore, by classifying mobile nodes in this way, the process $Proc_X$ only evaluates situations that could tip the connectivity between two mobile nodes from a not connected state to a connected state and inversely. Mobile M_X can thus change positions several times before a revaluation of its situation in relation to other mobile under its management is necessary.

4. EVALUATION OF THE ALGORITHM USED FOR THE EVALUATION OF THE STATES LINKS BETWEEN MOBILE NODES

The goal is to evaluate the cost of the algorithm used to control the mobility that we have just described. In fact, a slowdown of this process would result that the topology configured in the mobile network does not match the expected positions of mobiles nodes movements. This algorithm revaluates the topology at a time interval which dependents on the distance between the mobile nodes and a on a predefined step. The algorithm then performs two main stages:

- Revaluation of the links (connection or disconnection) for which the state is likely to have changed since the last evaluation
- Setting the corresponding firewall rules to allow or block communication between certain mobile nodes

If the system resources of the physical machine used for the experiment are highly stressed in particular by the activity of the virtual machines, this process can take a long time, pushing its next iteration at a time when mobile will have moved to a distance greater than the predefined step. In this case, the maximum travel distance defining which links should be revaluated or not may be insufficient and connections or disconnections could be omitted. It would be desirable, in this case, to increase the radius of the surveillance zone, however this would lead to évaluate more links at each stage and thus to a greater time calculation for the evaluation process.

4.1 Test platform description

The testbed on which we install VIRMANEL and run our evaluation is composed of a single server running the full software suite (SILUMOD, VIRMANEL and the OpenVZ virtualized machines). The server is equipped with 4GB of RAM and a 2.6 GHz QuadCore Intel CPU. It is running Ubuntu Linux 10.04 (The Lucid Lynx), kernel version 2.6.32 patched with the OpenVZ patch (2.6.32-feoktistov.1).

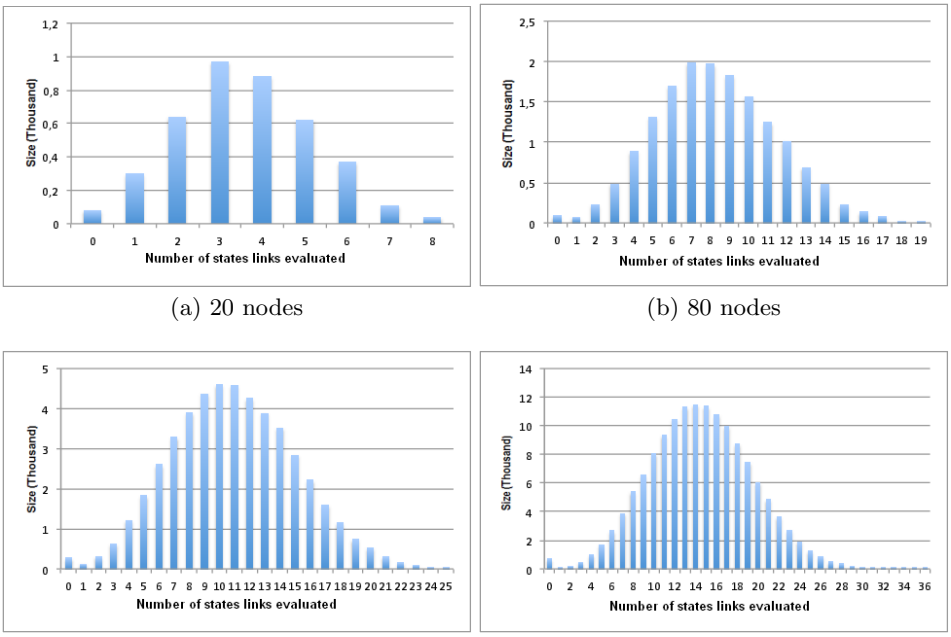
4.2 Execution time of the algorithm

We measured the execution time of the algorithm at different load levels on a physical testbed machine. Table 1 summarizes the results of this measurement campaign.

Figure 7 represents the evolution of the average of the execution time. The red curve represents the result of the linear interpolation of the data series, for which the coefficient of determination (R^2)) equals 98.1 %. The hypothesis of a linear evolution of the execution time seems to be a valid

Number of mobiles	Number of sampling	Average time of execution (ms)	Standard deviation (ms)	Coefficient of variation
20	4020	0,526	0,009	0,017110266
80	16080	11,954	0,178	0,014890413
180	36180	22,726	0,345	0,01518085
245	49245	28,094	0,305	0,010856411
320	64320	30,712	0,29	0,009442563
405	81405	39,289	0,379	0,009646466
500	100500	43,549	0,402	0,009230981
605	121605	53,428	0,634	0,011866437
720	144720	58,608	0,702	0,011977887
845	169845	72,169	0,874	0,012110463

Table 1: Results of the evaluation of the execution time of the algorithm that define the links to reevaluate



(a) 20 nodes (b) 80 nodes

(c) 320 nodes (d) 845 nodes

Figure 6: Average number of states links evaluated from 20 to 845 mobile nodes

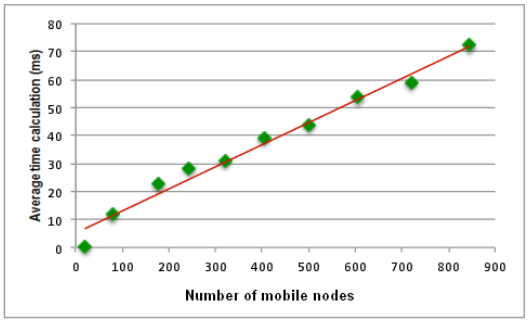


Figure 7: Average execution time of the algorithm from 20 to 845 mobile nodes

approximation. Given the relationship between the standard deviation and average execution time, we can consider that the regression equation of the line provides us with an evaluation of the execution time of the algorithm based on the number of mobile nodes. Therefore, it is assumed that, for the physical machine that we used, the execution time varies depending on the number of nodes, N, using the following equation:

$$T_{algo}(ms) = 0.0789 \cdot N + 5.1913$$

4.3 Number of states links to evaluate

We evaluated, based on the same scenario as previously, the number of states links to evaluate at each stage. Figure 6 represents the distributions of the number of states links evaluated for each stage in 4 cases, 20 nodes, 80 nodes, 320 nodes and 845 nodes. Distributions for other values have the same profile and are therefore not shown here.

We will approach these distributions by normal laws. Table 2 lists the distribution parameters in different scenarios.

Figure 8 represents the evolution of the average number of evaluations and a logarithmic interpolation of these evolution for which the coefficient of determination (R^2) is 99.3%. Figure 9, represents the evolution of the standard deviation and the logarithmic interpolation of the standard deviation

Number of mobiles nodes	Number of samples	Empirical average	Empirical standard deviation
20	4000	3,6	1,63
80	16000	8,32	3,2
180	36000	9,77	3,75
245	49000	10,95	4,15
320	64000	11,85	4,21
405	81000	12,5	4,52
500	100000	13,47	4,71
605	121000	13,76	4,67
720	144000	14,49	4,96
845	169000	15,12	5,18

Table 2: Number of state links to revaluate at each stage

for which the coefficient of determination (R^2) is 98.9%.

Therefore we can model the number of states links to change, or more precisely the firewall rules to execute at each step according to the number of mobile nodes N by a random variable following a normal distribution with parameters μ_{fw} and σ_{fw}, such as:

$$\begin{cases} N_{fw} \sim \mathcal{N}(\mu_{fw}, \sigma_{fw}) \\ \mu_{fw} = 2,9994 \ln(N) - 5,3484 \\ \sigma_{fw} = 0,9034 \ln(N) - 0,9414 \end{cases}$$

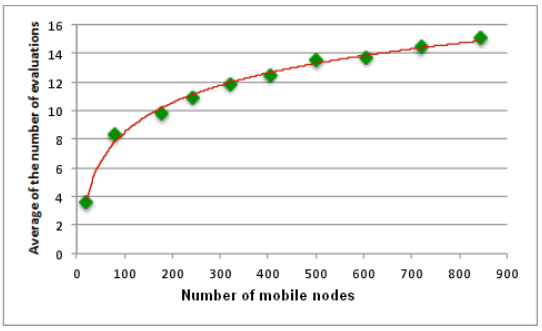


Figure 8: Average number of state links evaluated by the algorithm from 20 to 845 mobile nodes

We modeled the cost of the algorithm that we have implemented based on the number of mobile nodes. The logarithmic law obtained confirm that the number of operations required by the algorithm increases very little, even with a large number of mobile nodes. The measurements also show that despite the number of processes running in parallel and the various processing operations, the execution time of the algorithm is reasonable compared to the number of mobile nodes.

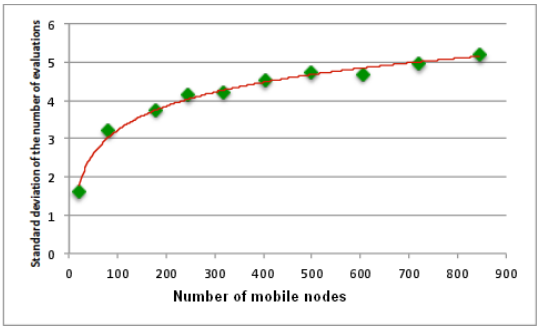


Figure 9: Standard deviation of the number of state links evaluated by the algorithm from 20 to 845 mobiles nodes

5. RELATED WORK

5.1 Methods for the nearest neighbor search problem

Several data structures exist for the nearest neighbor search problem[8]. After a description of the most common structures, this section presents a new algorithm for the nearest neighbor search in the context of mobility simulation.

5.1.1 Linear

The linear method is the most obvious. It consists of evaluating for each mobile node its distance with the other mobile nodes of experimentation at regular time intervals or at every movement of a mobile node. Even if a judicious definition of these two parameters can reduce the computation time of the simulator, this method is extremely costly because it is applied frequently and for each mobile node. Thus, the complexity for a linear search is $O(n)$.

5.1.2 Use of a grid

The principle is to divide the experimental area represented by a grid in several square shaped cells (Fig. 10). Mobile nodes are then assigned to one of these cells based on their position in the grid.

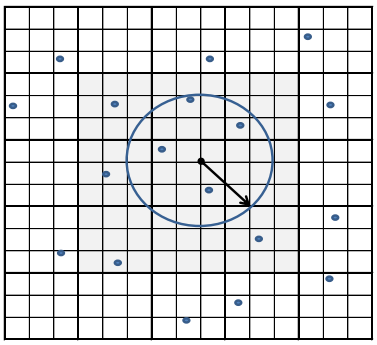


Figure 10: Experimental area division by using a grid

Whenever a mobile node moves, it re-evaluates its distance from the nodes that are in adjacent cells. This method was notably used to optimize the search in neighboring mobile nodes for NS2 [16] and has also a complexity in $O(n)$.

5.1.3 *Use of binaries trees*

K-dimensional trees [8] are data structures that are used to partition data and thus have a better distribution of nodes to limit the number of operations when the searching for nearest neighbors is required. K-dimensional trees are especially used for the nearest neighbor search by the ambient network simulator SimANet [20]. Figure 11 shows the data partitioning with a k-dimensional tree for the set of points (6,3), (5,5), (3,2), (4,8), (8,7) and (7,1).

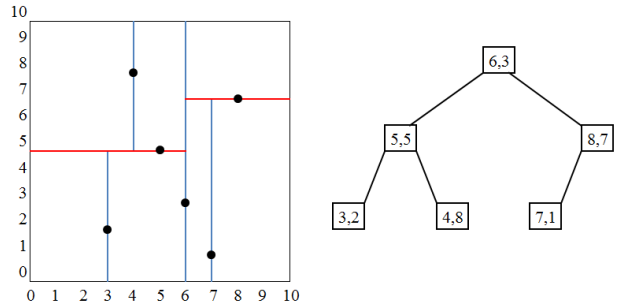


Figure 11: Data partitioning with a k-dimensional tree

But the use of k-dimensional tree requires a preliminary processing of the data, in $O(n.log(n))$ complexity, to construct the data tree. The nearest neighbor search and the insertion and destruction of data have thus a $O(log(n))$ complexity in the best case and $O(n)$ complexity in the worst case.

K-dimensional trees are considered one of the most efficient data structures for the nearest neighbor search even if their complexity has a high amplitude.

But the use of K-dimensional trees in the context of mobility is problematic because each mobile movement may invalidate the data partitioning already established. It would thus be necessary to define a new data partitioning.

5.2 Experimentation tools

For years, researchers conducted their experiments using network simulators. Simulation tools such as NS2 [4] and OPNET [5] are always used for research in mobile networks. These tools include the most common mobility models and all major communication protocols. For cons, the addition of new mobility models, new protocols or other new technology related to networks requires a thorough knowledge of these tools and a lot of programming effort.

Experimentation platforms adapted for the study of multi-hop wireless networks tend to develop according to exceptional financing obtained by the research laboratories. We can quote Senslab [10], a research platform with 1024 nodes distributed over 4 sites on French territory. The deployment of this platform aims to study wireless sensor networks on a large scale. But this kind of experimentation platforms require significant resources for technical management as well as for the purchase of equipment.

[4] `http://www.isi.edu/nsnam/ns/`
[5] `http://www.opnet.com`

Thus, over the years the virtualization has become an essential technology for computer networks experiments. A tool such as VIRCONEL [7] gives users a suitable environment for the study of computer networks but does not include the notion of mobility for nodes networks. Net-Kit [18] and VN-UML [12] both use UML [11] for virtualization network nodes. Their mode of operation is similar. The user defines a topology for the network to virtualize in a file. Besides the fact that performance with UML are disappointing compared to container type virtualization solutions [9], these solutions do not make it possible to manage mobility.

Tools were then developed by the scientific community to support the specificities related to the mobility of nodes in the context of experimentation. MASSIVE [15] uses virtual interfaces to simulate the mobile nodes. Thus virtual interfaces use the same network stack which is problematic for the isolation of the nodes during the execution of applications. Node mobility is defined using an internal script, which calculates the successives positions of a mobile node. These positions are then used by the MASSIVE server to determine which are the nodes that are close enough to communicate in a defined coverage radius. Neman [19] is a tool that, from virtual nodes emulated, simulates a mobile multi-hop network. The nodes are emulated using IP-aliasing [1]. The interface used for the visualization of multi-hop network is that of MobiEmu [2], a tool for MANET [3] emulation, that is coupled with NS3 [13] simulator for the mobility simulation. Topology as well as the motion of the nodes are reproduced by using positions calculated by NS-2 [4].

6. CONCLUSION

VIRMANEL allows to easily study real applications for mobile multi-hop networks and the SILUMOD languages allows to define well-know of new mobility models to obtain the successive positions of a moving node. The use of containers for virtualization ensures the deployment of a large number of virtual machines, each with its own Linux environment (network stack, routing tables, disk space, ...). And the algorithm defined to adapt the nearest neighbor search problem in mobility simulation, allows experiments with a large number of mobile nodes. Indeed, we have observed during our evaluations that the average time needed to evaluate states links between mobile nodes is very low and evolved linearly . In addition, the number of operations necessary to this evaluation evolves according to a logarithmic law. Thus, even an experiment with a large number of mobile nodes will require only a few operations.

We plan for our future research to improve this algorithm by studying the impact of different criteria. These criteria could be directly related to the mobility model and an evaluation of the efficiency of the algorithm according to the mobility model chosen would then to have a more effective management of connectivity between mobile nodes.

As of today, VIRMANEL does not include a wireless link emulator. It is hence limited to the evaluation of distributed applications and protocols in a mobile context. However, the architecture of VIRMANEL was designed to allow the easy integration of such a tool and we are currently evaluating various tools performances to realize this soon.

7. ACKNOWLEDGMENTS

This work is supported in part by the FIT project [6] granted by the French National Research Agency (ANR) and by the FITTING project funded by EIT ICT Labs. Part of this work has been realized at the LINCS laboratory[7].

8. REFERENCES

[1] Ip aliasing - http://www.tldp.org/howto/ip-alias/index.html.

[2] Mobiemu, a framework for emulating manets with lxc and ns-3 - http://code.google.com/p/mobiemu.

[3] Mobile ad-hoc networks - ietf . http://datatracker.ietf.org/wg/manet/documents/.

[4] The network simulator ns-2 . http://www.isi.edu/nsnam/ns/.

[5] Y. Benchaïb and C. Chaudet. Silumod: A simulation language for user mobility models definition in multihop networks. In *First Asia-Pacific Programming Languages and Compilers Workshop (APPLC)*, Beijing, China, June 2012.

[6] Y. Benchaïb and C. Chaudet. Virmanel: a mobile multihop network virtualization tool. In *7th ACM International Workshop on Wireless Network Testbeds, Experimental evaluation and Characterization*, Istanbul, Turkey, Aug. 2012.

[7] Y. Benchaïb and A. Hecker. Virconel: A network virtualizer. In *Proceedings of the 2011 IEEE 19th Annual International Symposium on Modelling, Analysis, and Simulation of Computer and Telecommunication Systems*, Aug. 2011.

[8] J. L. Bentley. Multidimensional binary search trees used for associative searching. *Commun. ACM*, 18(9):509–517, Sept. 1975.

[9] G. Bhanage, I. Seskar, Y. Zhang, D. Raychaudhuri, and S. Jain. Experimental evaluation of openvz from a testbed deployment perspective. In T. Magedanz, A. Gavras, N. Thanh, and J. Chase, editors, *Testbeds and Research Infrastructures. Development of Networks and Communities*, volume 46 of *Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*, pages 103–112. Springer Berlin Heidelberg, 2011.

[10] C. B. des Roziers, G. Chelius, T. Ducrocq, E. Fleury, A. Fraboulet, A. Gallais, N. Mitton, T. Noël, and J. Vandaele. Using senslab as a first class scientific tool for large scale wireless sensor network experiments. In *Proceedings of the 10th international IFIP TC 6 conference on Networking - Volume Part I*, NETWORKING'11, pages 147–159, Berlin, Heidelberg, 2011. Springer-Verlag.

[11] J. Dike. User-mode linux. In *Proceedings of the 5th annual Linux Showcase & Conference - Volume 5*, ALS '01, pages 2–2, Berkeley, CA, USA, 2001. USENIX Association.

[12] F. Galan, D. Fernandez, J. Ruiz, O. Walid, and T. de Miguel. Use of virtualization tools in computer network laboratories. In *Information Technology Based Higher Education and Training, 2004. ITHET 2004. Proceedings of the FIfth International Conference on*, pages 209 – 214, may-2 june 2004.

[13] T. R. Henderson, S. Roy, S. Floyd, and G. F. Riley. ns-3 project goals. In *Proceeding from the 2006 workshop on ns-2: the IP network simulator*, WNS2 '06, New York, NY, USA, 2006. ACM.

[14] K. Kolyshkin. Virtualization in linux. *White paper, OpenVZ*, 2006.

[15] M. Matthes, H. Biehl, M. Lauer, and O. Drobnik. Massive: An emulation environment for mobile ad-hoc networks. In *Second Annual Conference on Wireless On-demand Network Systems and Services (WONS'05)*, St Moritz, Switzerland, Jan. 2005.

[16] V. Naoumov and T. Gross. Simulation of large ad hoc networks. In *Proceedings of the 6th ACM international workshop on Modeling analysis and simulation of wireless and mobile systems*, MSWIM '03, pages 50–57, New York, NY, USA, 2003. ACM.

[17] M. Odersky, S. Micheloud, N. Mihaylov, M. Schinz, E. Stenman, M. Zenger, and et al. An overview of the scala programming language. Technical report, 2004.

[18] M. Pizzonia and M. Rimondini. Netkit: easy emulation of complex networks on inexpensive hardware. In *Proceedings of the 4th International Conference on Testbeds and research infrastructures for the development of networks & communities*, TridentCom '08, pages 7:1–7:10, ICST, Brussels, Belgium, Belgium, 2008. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

[19] M. Puzar and T. Plagemann. Neman: a network emulator for mobile ad-hoc networks. 1:155 – 161, 15-17, 2005.

[20] M. Vodel, M. Sauppe, M. Caspar, and W. Hardt. Simanet–a large scalable, distributed simulation framework for ambient networks. *Journal of Communications*, 3(7):11–19, 2008.

[6] http://fit-equipex.fr/

[7] http://www.lincs.fr/